Case No. 78803 (120-2 US)

COMMISSIONER FOR PATENTS
PO BOX 1450
ARLINGTON, VA 22313-1450

Transmitted herewith for filing is the patent application of:

I HEREBY CERTIFY THIS PAPER OR FEE IS BEING DEPOSITED WITH THE U.S. POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 CFR 1.10 ON THE DATE INDICATED BELOW AND IS ADDRESSED TO: COMMISSIONER FOR PATENTS, PO BOX 1450, ALEXANDRIA, VA 22313-1450.

EXPRESS MAIL NO.: EL 322695237 US

DATE OF DEPOSIT: September 21, 2004

NAME: Kristen Ferguson

SIGNATURE: Kristen Ferguson

22141 U.S. PTO 10/946536 092104

Inventors: Donn ROCHETTE, Paul O'LEARY, Dean HUFFMAN

For: A COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

Please note the following:

[X] Patent Application: 22 pages, 20 claims.
[X] 7 sheets of drawings.
[X] The suggested drawing figure to be published is FIG. 3
[X] A declaration and power of attorney.
[ ] Citation Under 37 CFR 1.97 and PTO-1449.
[X] Assignee info:
Name: Trigence Corp.
Address: 750 Palladium Drive, Ste. 210
Ottawa, Ontario, CANADA K2V 1C7
State of Incorporation:
[X] Applicant qualifies as a small entity under 37 CFR ' 1.27.

[X] Applicant claims priority benefit to the following U.S. application(s):
Application No.: 60/504,213
Filing Date: SEPTEMBER 22, 2003

The filing fee has been calculated as shown below:

	(Col. 1)		(Col. 2)		SMALL ENTITY			LARGE ENTITY	
FOR:	# FILED		# EXTRA		RATE	FEE		RATE	FEE
BASIC FEE						$ 385	OR		$
TOTAL CLAIMS	20	-20			X 9	$	OR	X 18	$
INDEP CLAIMS	1	- 3			X 43	$	OR	X 86	$
* If the difference in Col. 1 is less than "0", enter "0" in Col. 2.					TOTAL	$ 385		TOTAL	$

[X] Authorization is given to charge the Filing Fee in the amount of $385.00 to Deposit Account 50-2810.

[X] Authorization is given to charge the Assignment Recordal Fee in the amount of $40.00 to Deposit Account 50-2810.

[X] The Commissioner is authorized to charge or credit any discrepancies in fee amounts to Deposit Account No. 50-2810.

[X] PLEASE ADDRESS ALL CORRESPONDENCE TO ATTORNEY OF RECORD:
CHARLES E. WANDS

[X] **Please associate this application with Customer No. 27975.**

CUSTOMER NO. 27975

Telephone: (321) 725-4760

September 21, 2004

CHARLES E. WANDS
Reg. No. 25,649

Google Exhibit 1002
Google v. VirtaMove

Case No. 78803 (120-2 US)

COMMISSIONER FOR PATENTS
PO BOX 1450
ARLINGTON, VA 22313-1450

Transmitted herewith for filing is the patent application of:

I HEREBY CERTIFY THIS PAPER OR FEE IS BEING DEPOSITED WITH THE U.S. POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 CFR 1.10 ON THE DATE INDICATED BELOW AND IS ADDRESSED TO: COMMISSIONER FOR PATENTS, PO BOX 1450, ALEXANDRIA, VA 22313-1450.

EXPRESS MAIL NO.: EL 322695237 US

DATE OF DEPOSIT: September 21, 2004

NAME: Kristen Ferguson

SIGNATURE: Kristen Ferguson

22141 U.S. PTO 10/946536 092104

Inventors: Donn ROCHETTE, Paul O'LEARY, Dean HUFFMAN

For: A COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

Please note the following:

[X] Patent Application: 22 pages, 20 claims.
[X] 7 sheets of drawings.
[X] The suggested drawing figure to be published is FIG. 3
[X] A declaration and power of attorney.
[ ] Citation Under 37 CFR 1.97 and PTO-1449.
[X] Assignee info:
Name: Trigence Corp.
Address: 750 Palladium Drive, Ste. 210
Ottawa, Ontario, CANADA K2V 1C7
State of Incorporation:
[X] Applicant qualifies as a small entity under 37 CFR ' 1.27.

[X] Applicant claims priority benefit to the following U.S. application(s):
Application No.: 60/504,213
Filing Date: SEPTEMBER 22, 2003

The filing fee has been calculated as shown below:

	(Col. 1)		(Col. 2)		SMALL ENTITY			LARGE ENTITY	
FOR:	# FILED		# EXTRA		RATE	FEE		RATE	FEE
BASIC FEE						$ 385	OR		$
TOTAL CLAIMS	20	-20			X 9	$	OR	X 18	$
INDEP CLAIMS	1	- 3			X 43	$	OR	X 86	$
* If the difference in Col. 1 is less than "0", enter "0" in Col. 2.					TOTAL	$ 385		TOTAL	$

[X] Authorization is given to charge the Filing Fee in the amount of $385.00 to Deposit Account 50-2810.

[X] Authorization is given to charge the Assignment Recordal Fee in the amount of $40.00 to Deposit Account 50-2810.

[X] The Commissioner is authorized to charge or credit any discrepancies in fee amounts to Deposit Account No. 50-2810.

[X] PLEASE ADDRESS ALL CORRESPONDENCE TO ATTORNEY OF RECORD:
CHARLES E. WANDS

[X] **Please associate this application with Customer No. 27975.**

CUSTOMER NO. 27975

Telephone: (321) 725-4760

September 21, 2004

CHARLES E. WANDS
Reg. No. 25,649

A COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

Cross-Reference to Related Applications

[1] This application claims priority of U.S. Provisional Patent Application No: 60/504,213 filed September 22, 2003, entitled "User Mode Critical System Element as Shared Libs", which is incorporated herein by reference.

Field of the Invention

[2] This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

Background of the Invention

[3] Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

[4] For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

[5] In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

[6] In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

[7] An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

Summary of the Invention

[8] In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

[9] The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

[10] In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

[11] an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

[12] a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of

the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

[13] In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.
2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

[14] In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

[15] Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

[16] Typically, the critical system elements are not removed from operating system kernel.

[17] In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

[18] In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

[19] In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

[20] In some embodiments, the interrupt handling capabilities are initialized through a system call.

[21] In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

[22] In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

[23] In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

[24] In some embodiments, function overlays are used to intercept software application accesses to operating system services.

[25] In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

[26] In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

[27] In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

[28] In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

[29] In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

[30] In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

[31] In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application

libraries, the shared library and the critical system elements all of which are operating in user mode.

[32] In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

[33] In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

[34] In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

[35] In some embodiments, the kernel module is adapted to export services for device interface.

[36] In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

[37] In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

[38] In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

[39] According to a second broad aspect, the invention provides an operating system comprising the above computing system.

[40] According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

[41] According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

[42] According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

[43] According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

[44] The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

[45] In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

Brief Description of the Drawings

[46] Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

[47] Figure 1 is an architectural view of the traditional monolithic prior art operating system;

[48] Figure 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

[49] Figure 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

[50] Figure 3 is an architectural view of an embodiment of the invention;

[51] Figure 4 is a functional view showing how critical system elements exist in the same context as an application;

[52] Figure 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

[53] Figure 6 shows how interrupt handling occurs in an embodiment of the invention.

Detailed Description of the Preferred Embodiments

[54] Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

[55] The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TPC/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

[56] In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires

that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

[57] By way of introduction, a number of terms will now be defined.

[58] Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

[59] A CSE is a dynamic object providing some function that is executing instructions used by applications.

[60] BY WAY OF EXAMPLE CSEs INCLUDE:

[61] Network services including TCP/IP, Bluetooth, ATM, or message passing protocols

[62] File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality
2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

[63] Implementation of file system optimizations for specific application behavior

[64] Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided
2. Modified protocol processing for custom hardware services

[65] Compute platform: The combination of computer hardware and a single instance of an operating system.

[66] User mode: The context in which applications execute.

[67] Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

[68] Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

[69] Application library: A collection of functions in an archive format that is combined with an application to export system elements.

[70] Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

[71] Static library: An application library whose code space is contained in a single application.

[72] Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

[73] Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

[74] Figure 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

[75] In order for an application of Figure 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b makes a call to the application libraries 14. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application 12a, 12b makes a call to a critical system element 17 through the application library 14, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware 18 enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval 12a, 12b in Figure 1 represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

[76] Figure 2a shows a system architecture where critical system elements 27a, 27b execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in figure 2a is the GNU Hurd operating system.

[77] The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The

libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of Figure 1, should be able to run on the system of Figure 2a as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

[78] Figure 2b is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in Figure 2a and Figure 2b is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in Figure 2b is Apple's MAC OS X™.

[79] This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in Figure 1; also they are not removed from the context of an application, as is the case with a multi-server architecture depicted in Figure 2a. Rather, they are replicated, and embodied in the context of an application.

[80] Figure 3 shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

[81] Software applications 32a, 32b, utilize shared libraries 34 as is done in United States Provisional Patent Application serial number 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries 35. Extended services are similar to those that appear in the context of the operating system kernel 36.

[82] Figure 4 illustrates the functionality of an application process as it exists above an operating system that was described in figure 3. Applications exist and run in user mode while the operating system kernel 46 itself runs in kernel mode. User mode functionality includes the user applications 42, the standard application libraries 44, and one or more critical system elements, 45 & 47. Figure 4 shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, 45, provides control operations while the CSE shared library, 47, provides an implementation of a critical system element.

[83] The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications 42 and as such can be run in the same context as the applications 42. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library 45,47 are also included in the operating system kernel 46.

[84] Furthermore, there might be different versions of a given critical system element 47 with different applications accessing these different versions within their respective context.

[85] In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

[86] Figure 5 represents the function of the kernel module 58, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

[87] Figure 6 represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module 58 is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

[88] Function Overlays

[89] A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

[90] The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

[91] The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

[92] Critical System Elements

[93] According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

[94] Figure 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

[95] Figure 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in Figure 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

[96] Kernel module

[97] In some embodiments, control code is placed in kernel mode as shown in Figure 4. Figure 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.

10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

[98] As shown in Figure 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

[99] The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been

implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

[100] It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

Claims

What is claimed is:

1. A computing system for executing a plurality of software applications comprising: an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,
a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode and wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and when accessed by one or more software applications form a part of the one or more software applications, and wherein an instance of a SLCSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing essentially the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim 1 wherein the one or more SLCSEs provided to an application having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between a SLCSE in the context of an application program and a device driver.

6. A computing system as defined in claim 1, wherein an SLCSE related to a predetermined function is provided to a first application for running an instance of the SLCSE, and wherein an SLCSE for performing essentially a same function is provided to a second application for running a second instance of the SLCSE simultaneously.

7. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

8. A computing system according to claim 7 wherein a handler is provided for notifying the SLCSE in the context of an application through the use of an up call mechanism.

9. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

10. A computing system according to claim 2, wherein a function overlay is used to provide a software application access to operating system services.

11. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications as the software applications are loaded such that software applications have a link that provides unique access to a unique instance of a CSE.

12. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

13. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

14. A computing system according to claim 11 wherein some SLCSEs are modified for a particular software application.

15. A computing system according to claim 14 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

16. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

17. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the software applications, by being linked thereto.

18. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

19. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSLCEs.

20. An operating system comprising the computing system of claim 2.

Abstract of the Disclosure

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is run in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

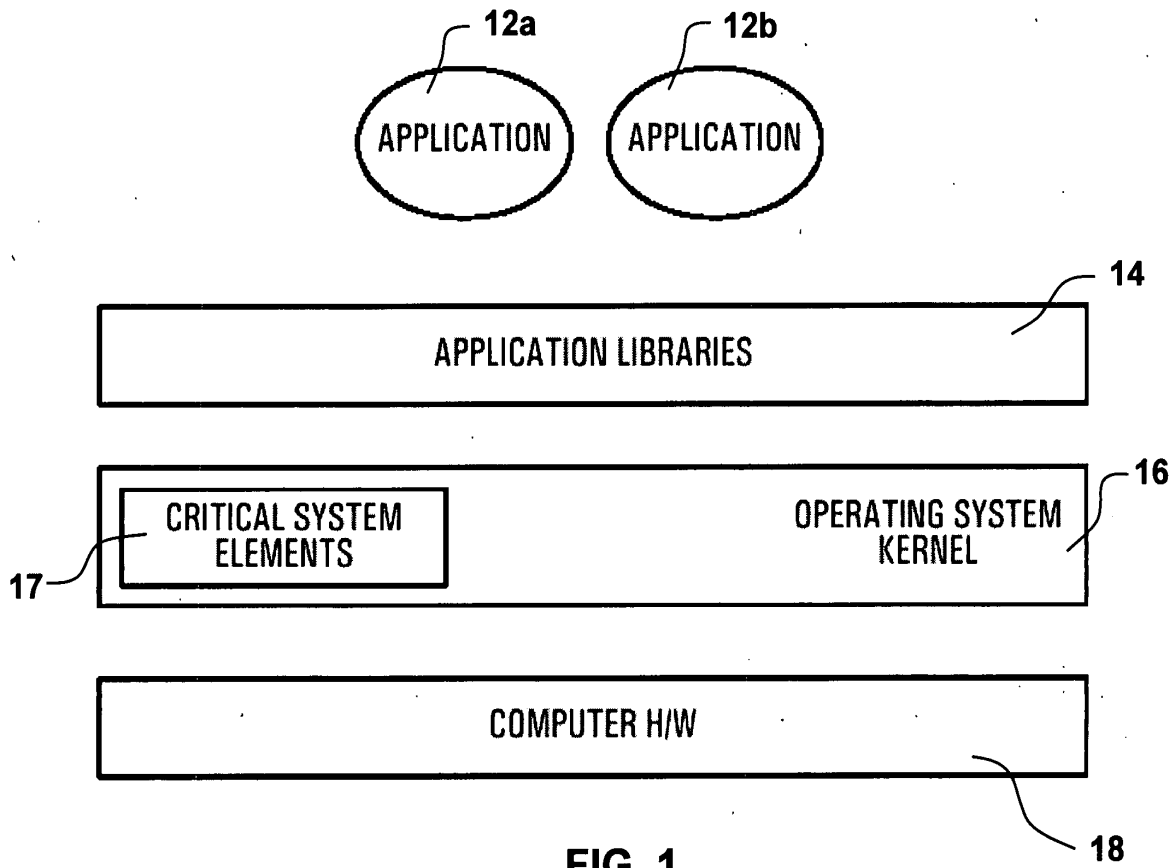


FIG. 1
Prior Art

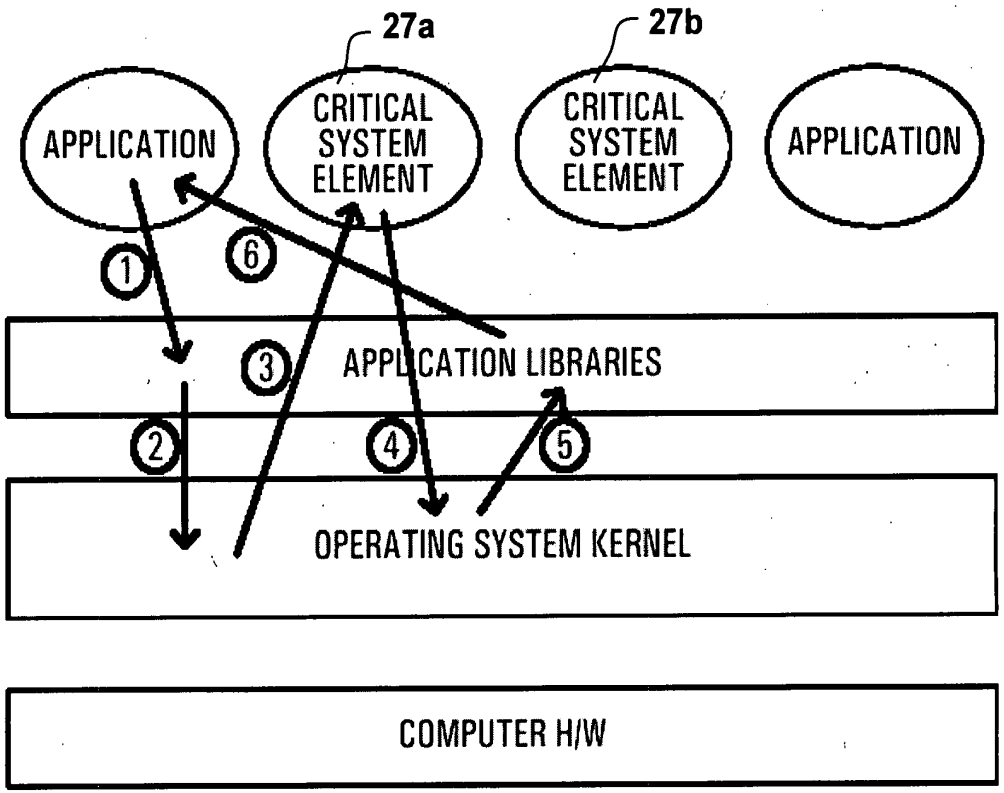


FIG. 2a
Prior Art

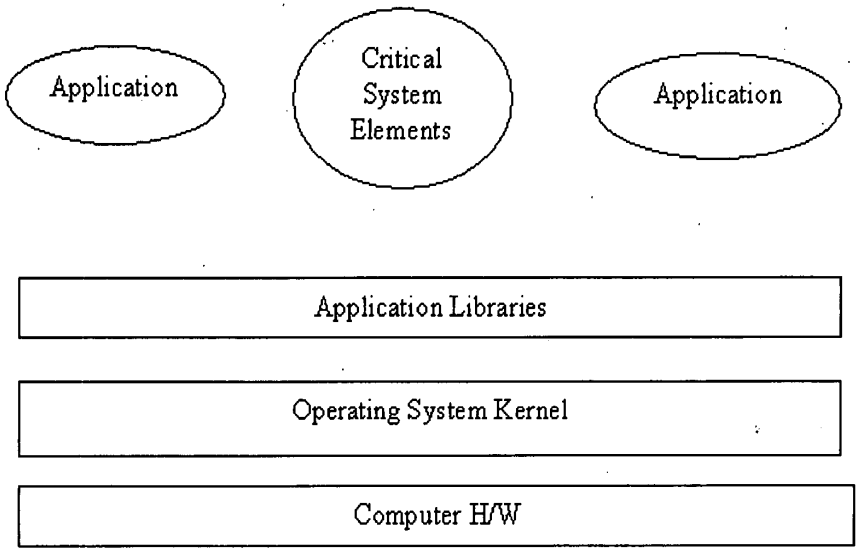


FIG. 2b
Prior Art

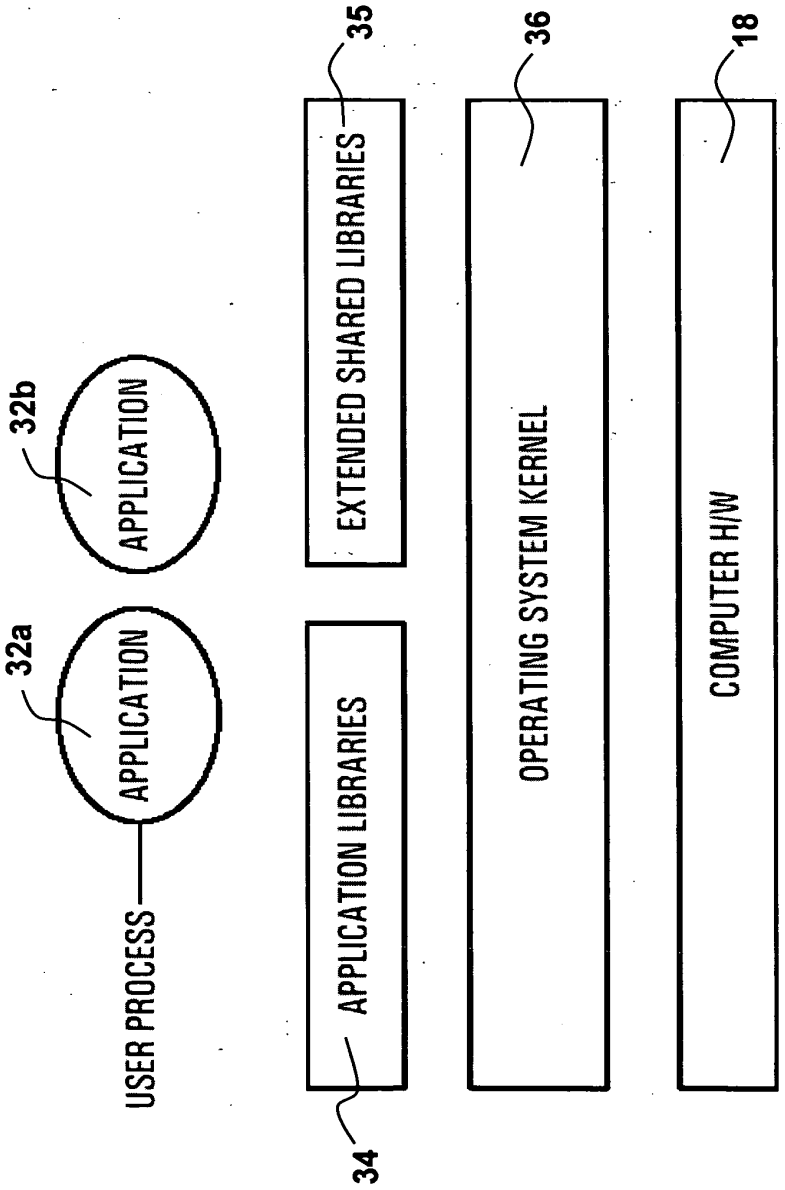

USER PROCESS
32a
APPLICATION
32b
APPLICATION
34
APPLICATION LIBRARIES
35
EXTENDED SHARED LIBRARIES
36
OPERATING SYSTEM KERNEL
18
COMPUTER H/W

FIG. 3

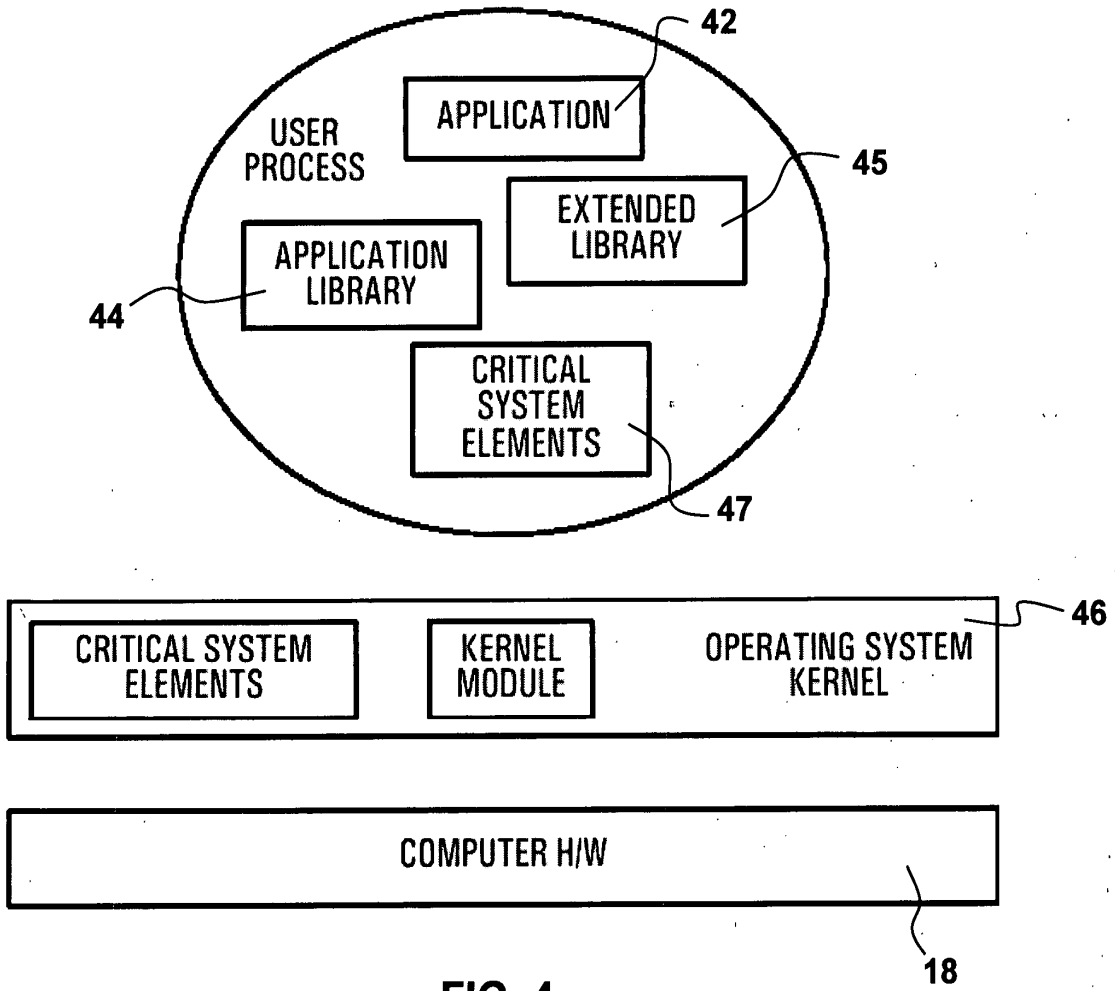

USER PROCESS
APPLICATION
42
EXTENDED LIBRARY
45
APPLICATION LIBRARY
44
CRITICAL SYSTEM ELEMENTS
47
CRITICAL SYSTEM ELEMENTS
KERNEL MODULE
OPERATING SYSTEM KERNEL
46
COMPUTER H/W
18

FIG. 4

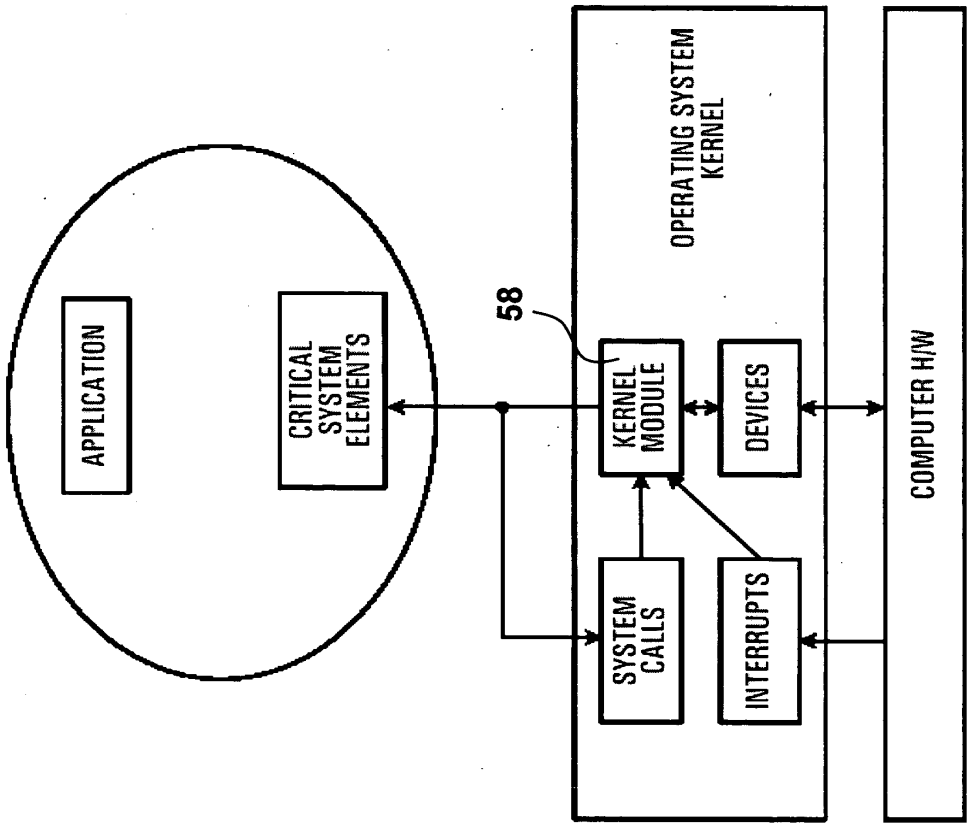

APPLICATION
CRITICAL SYSTEM ELEMENTS
58
OPERATING SYSTEM KERNEL
KERNEL MODULE
DEVICES
SYSTEM CALLS
INTERRUPTS
COMPUTER H/W

FIG. 5

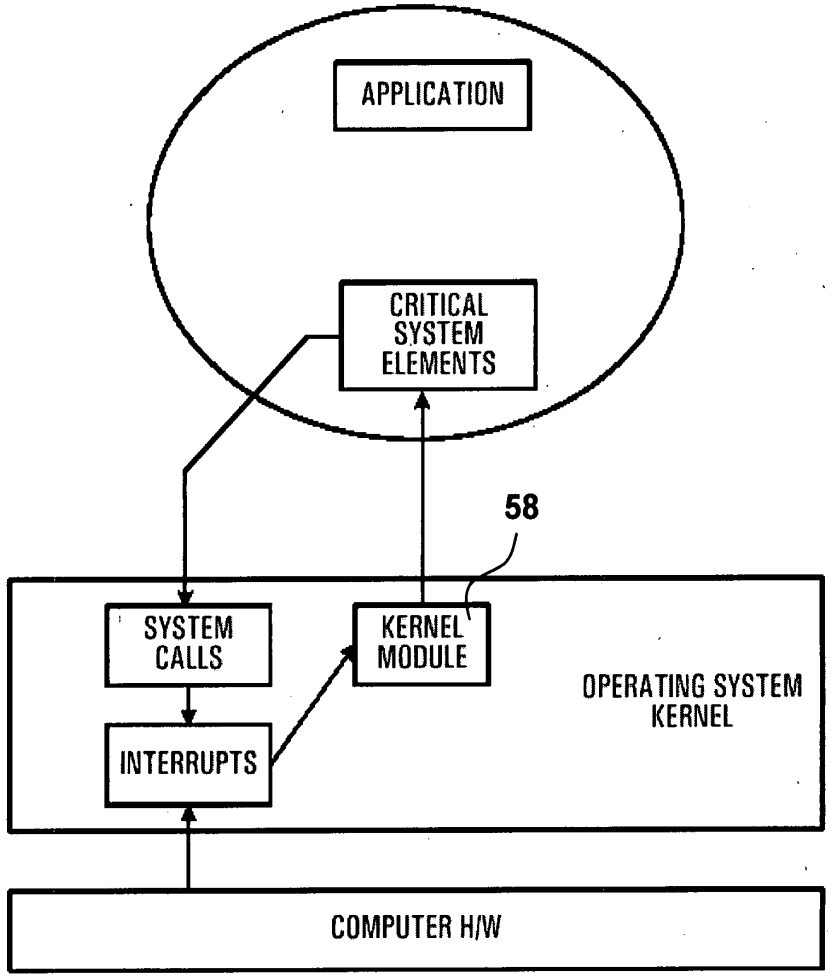

APPLICATION
CRITICAL SYSTEM ELEMENTS
58
SYSTEM CALLS
KERNEL MODULE
OPERATING SYSTEM KERNEL
INTERRUPTS
COMPUTER H/W

FIG. 6

120-2 US

DECLARATION FOR PATENT APPLICATION

As below named inventors, we hereby declare that:

Our residence, post office addresses and citizenship are as stated below next to our names, **Donn ROCHETTE; Paul O'LEARY, and Dean HUFFMAN.**

We believe we are the original, first and joint inventors of the subject matter which is claimed and for which a patent is sought on the invention entitled

A COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

the specification of which is attached hereto.

I/we hereby authorize my/our attorney to insert here in parentheses (Application Number _____, filed _____), the filing date and application number of said application, when known.

We hereby state that we have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

We acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of the Federal Regulations, S1.56(a).

We hereby claim the benefit of Title 35, U.S.C. 119(e), of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is now disclosed in the prior United States **provisional** patent application

(Number)	(Filing Date)	(Status) patented/pending/abandoned
60/504,213	**September 22, 2003**	**Pending**

We hereby appoint the following agents to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith:

Address all telephone calls to: **Charles E. Wands, Regn. No: 25,649**
at telephone number: **(321) 725-4760**

Address all correspondence to: **Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.**
1401 Citrus Center
255 South Orange Avenue
Box 3791
Orlando, Florida
USA 32802-3791

I hereby declare that all statements made herein of my own knowledge are true and that 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 U.S.C. and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of first inventor: **Donn ROCHETTE**

Inventor's signature: Date: 9-16-04

Residence: **3408 30th Avenue, Fenton, Indiana 50539, U.S.A.** Iowa DR
Post office address: **As above**

Citizenship: **U.S.** 9-16-04

Full name of second inventor: **Paul O'LEARY**

Inventor's signature: Date:

Residence: **22 Pentland Crescent, Kanata, Ontario, Canada K2K 1V5**
Post office address: **As above**

Citizenship: **Canadian**

Full name of third inventor: **Dean HUFFMAN**

Inventor's signature: Date:

Residence: **13 Pelee Street, Kanata, Ontario, Canada K2M 2R4**
Post office address: **As above**

120-2 US

Address all telephone calls to:	**Charles E. Wands, Regn. No: 25,649**
at telephone number:	**(321) 725-4760**
Address all correspondence to:	**Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.** **1401 Citrus Center** **255 South Orange Avenue** **Box 3791** **Orlando, Florida** **USA 32802-3791**

I hereby declare that all statements made herein of my own knowledge are true and that 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 U.S.C. and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of first inventor:	**Donn ROCHETTE**
Inventor's signature:	Date:
Residence: Post office address:	**3408 30th Avenue, Fenton, Indiana 50539, U.S.A.** **As above**
Citizenship:	**U.S.**

Full name of second inventor:	**Paul O'LEARY**
Inventor's signature:	Paul O'Leary Date: Sept. 15, 2004
Residence: Post office address:	**22 Pentland Crescent, Kanata, Ontario, Canada K2K 1V5** **As above**
Citizenship:	**Canadian**

Full name of third inventor:	**Dean HUFFMAN**
Inventor's signature:	Date: Sept. 15, 2004
Residence: Post office address:	**13 Pelee Street, Kanata, Ontario, Canada K2M 2R4** **As above**
Citizenship:	**Canadian**

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

09/23/2004 FFANAEIA 00000103 502810 10946536
01 FC:2001 385.00 DA

PTO-1556
(5/87)

*U.S. Government Printing Office: 2002 — 489-267/69033

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2003

Application or Docket Number
10/946536

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	20	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	20 minus 20=	*
INDEPENDENT CLAIMS	1 minus 3 =	*
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐		OR	OTHER THAN SMALL ENTITY	
RATE	FEE		RATE	FEE
BASIC FEE	385.00	OR	BASIC FEE	770.00
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL	385	OR	TOTAL	

CLAIMS AS AMENDED - PART II

AMENDMENT A

	(Column 1)		(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	(Column 1)		(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	(Column 1)		(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

RECEIVED
CENTRAL FAX CENTER
OCT 18 2005

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE PATENT APPLICATION OF: ROCHETTE ET AL

OUR FILE NO: 78803 (120-2 US) CONFIRMATION NO. 7612

SERIAL NUMBER: 10/946,536 GROUP: 2127

FILED: SEPTEMBER 21, 2004

TITLE: COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

Information Disclosure Statement

VIA FACSIMILE NO. 571-273-8300
Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Transmitted herewith is an Information Disclosure Statement (Form PTO-1449A) in the above-captioned application with references. In accordance with current USPTO procedures published 05 AUG 2003, in 1276 OG 55, copies of the U.S. patent documents cited in the form 1449A are not attached.

Certification

_____ This Information Disclosure Statement is submitted within three months of:

(i) the filing date of the above-identified U.S .National Patent application, or
(ii) the date of entry into the U.S. National Stage of the above-identified International Application, or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies.

__X__ This Information Disclosure Statement is submitted prior to the mailing date of the first Office Action on the merits received by Applicant in the above-identified application.

_____ This Information Disclosure Statement is submitted after three months from

(i) the filing date of the above-identified U.S. National Patent application, or
(ii) after three months from entry into the U.S. National Stage of the above-identified International Application; or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies; and after the mailing date of the first Office Action on the merits of the above-identified application, but prior to issuance of the earlier of any Final Action or Notice of Allowance sent in such application. The certification under 37 C.F.R. §1.97(e) is submitted separately or below, or the fee required under 37 C.F.R. §1.97(c) and § 1.17(p) is submitted herewith.

_____ This Information Disclosure Statement is submitted after the earlier of the mailing date of a final rejection or Notice of Allowance sent in this application but before payment of the Issue Fee. The certification required under 37 C.F.R. § 1.97(e) is submitted separately or below. A petition to the Commissioner and the appropriate fee pursuant to §1.17(i) (1) are submitted herewith.

PAGE 1/3 * RCVD AT 10/18/2005 12:16:47 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-6/24 * DNIS:2738300 * CSID:13219847078 * DURATION (mm-ss):00-56

_____ A certification under 37 C.F.R §1.97 is submitted herewith separately from this paper.

_____ It is hereby certified that each item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this statement.

_____ No item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application or, to the knowledge of the undersigned, after making reasonable inquiry, was known to any individual designated in 37 C.F.R. §1.56(c) more than three months prior to the filing of this statement.

Respectfully submitted,

CHARLES E. WANDS
Reg. No. 25,649

Telephone: (321) 725-4760

Customer No.: 27975

CERTIFICATE OF FACSIMILE TRANSMISSION

I HEREBY CERTIFY that the foregoing correspondence has been forwarded via facsimile number 571-273-8300 to MAIL STOP AMENDMENT, COMMISSIONER FOR PATENTS, this 18 day of October 2005.

J. Kallemeres

Form PTO 1449A U.S. Department of Commerce Patent and Trademark Office Information Disclosure Statement by Applicant	ATTY. DOCKET NUMBER: 78803 (120-2 US)	SERIAL NUMBER: 10/946,536
	APPLICANT: ROCHETTE ET AL	
	FILING DATE: SEPTEMBER 21, 2004	GROUP: 2127

U.S. Patent Documents

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING APPS
		60/512,103	Oct 20, 2003	Rochette et al			

Foreign Patent Documents

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO

Other Documents (Including Author, Title, Date Pertinent Pages, Etc.)

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if citation is considered, whether or not citation is in conformance with MPEP 609; draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant

PAGE 3/3 * RCVD AT 10/18/2005 12:16:47 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-6/24 * DNIS:2738300 * CSID:13219847078 * DURATION (mm-ss):00-56

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Patent Application of: Rochette et al.		**Our File:**	**78803 (120-2 US)**
Serial No:	**10/946,536**	**Group:**	**2127**
Filed:	**September 21, 2004**	**Confirmation No. 7612**	
Title:	**COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**		

Information Disclosure Statement

EFILED
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Transmitted herewith is an Information Disclosure Statement (Form PTO-1449A) in the above-captioned application with references. In accordance with current USPTO procedures published 05 AUG 2003, in 1276 OG 55, copies of the U.S. patent documents cited in the form 1449A are not attached.

Certification

_____ This Information Disclosure Statement is submitted within three months of:

(i) the filing date of the above-identified U.S .National Patent application, or
(ii) the date of entry into the U.S. National Stage of the above-identified International Application, or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies.

__X__ This Information Disclosure Statement is submitted prior to the mailing date of the first Office Action on the merits received by Applicant in the above-identified application.

_____ This Information Disclosure Statement is submitted after three months from

(i) the filing date of the above-identified U.S. National Patent application, or
(ii) after three months from entry into the U.S. National Stage of the above-identified International Application; or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies; and after the mailing date of the first Office Action on the merits of the above-identified application, but prior to issuance of the earlier of any Final Action or Notice of Allowance sent in such application. The certification under 37 C.F.R. §1.97(e) is submitted separately or below, or the fee required under 37 C.F.R. §1.97(c) and § 1.17(p) is submitted herewith.

_____ This Information Disclosure Statement is submitted after the earlier of the mailing date of a final rejection or Notice of Allowance sent in this application but before payment of the Issue Fee. The certification required under 37 C.F.R. § 1.97(e) is submitted separately or below. A petition to the Commissioner and the appropriate fee pursuant to §1.17(i) (1) are submitted herewith.

_____ A certification under 37 C.F.R §1.97 is submitted herewith separately from this paper.

_____ It is hereby certified that each item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this statement.

_____ No item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application or, to the knowledge of the undersigned, after making reasonable inquiry, was known to any individual designated in 37 C.F.R. §1.56(c) more than three months prior to the filing of this statement.

Respectfully submitted,

/charles edmund wands/
Reg. No. 25,649

NOV 2 8 2006
Date:

CUSTOMER NO. 27975

Form PTO 1449A U.S. Department of Commerce Patent and Trademark Office Information Disclosure Statement by Applicant	ATTY. DOCKET NUMBER: 78803 (120-2 US)	SERIAL NUMBER: 10/946,536
	APPLICANT: ROCHETTE ET AL.	
	FILING DATE: September 21, 2004	GROUP: 2127

U.S. Patent Documents

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING APPE
		2002/0004854 A1	Jan 10, 2002	Hartley			
		2003/0101292 A1	May 20, 2003	Fisher			
		2002/0174215 A1	Nov 21, 2002	Schaefer	709	224	

Foreign Patent Documents

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO
		WO 02/06941 A	Jan 24, 2002	Connectix Corporation				
		WO 06/039181 A	Apr 13, 2006	Citrix Systems, Inc.				

Other Documents (Including Author, Title, Date Pertinent Pages, Etc.)

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if citation is considered, whether or not citation is in conformance with MPEP 609; draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant

Electronic Acknowledgement Receipt

EFS ID:	1338012
Application Number:	10946536
International Application Number:	
Confirmation Number:	7612
Title of Invention:	Computing system having user mode critical system elements as shared libraries
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	Charles Edmund Wands.
Filer Authorized By:	
Attorney Docket Number:	78803 (120-2 US)
Receipt Date:	28-NOV-2006
Filing Date:	21-SEP-2004
Time Stamp:	15:50:16
Application Type:	Utility

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)	Multi Part /.zip	Pages (if appl.)
1	Foreign Reference	WO2006039181.pdf	6953363	no	146
Warnings:					

Information:					
2	Foreign Reference	WO0206941A1.pdf	1039779	no	21
Warnings:					
Information:					
3	Information Disclosure Statement (IDS) Filed	78803ids2.pdf	223504	no	3
Warnings:					
Information:					
This is not an USPTO supplied IDS fillable form					
			Total Files Size (in bytes):	8216646	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 April 2006 (13.04.2006)

PCT

(10) International Publication Number
WO 2006/039181 A1

(51) International Patent Classification[7]: G06F 9/46

(21) International Application Number: PCT/US2005/033994

(22) International Filing Date: 23 September 2005 (23.09.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

10/711,737	30 September 2004 (30.09.2004)	US
10/711,736	30 September 2004 (30.09.2004)	US
10/711,735	30 September 2004 (30.09.2004)	US
10/711,734	30 September 2004 (30.09.2004)	US
10/711,733	30 September 2004 (30.09.2004)	US
10/711,732	30 September 2004 (30.09.2004)	US
10/956,723	1 October 2004 (01.10.2004)	US
11/231,284	19 September 2005 (19.09.2005)	US
11/231,316	19 September 2005 (19.09.2005)	US
11/231,317	19 September 2005 (19.09.2005)	US
11/231,315	19 September 2005 (19.09.2005)	US
11/231,370	19 September 2005 (19.09.2005)	US

(71) Applicant *(for all designated States except US)*: **CITRIX SYSTEMS, INC.** [US/US]; 851 West Cypress Creek Road, Fort Lauderdale, FL 33309 (US).

(72) Inventors; and

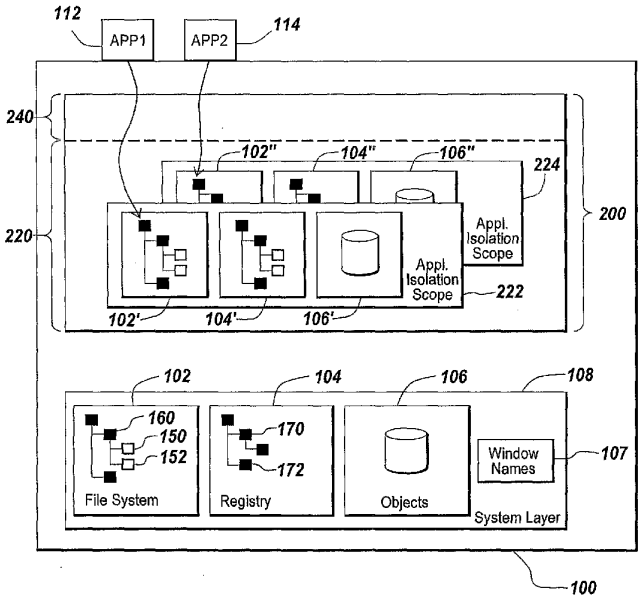
(75) Inventors/Applicants *(for US only)*: **LABORCZFALVI, Lee, George** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, Nort Ryde NSW 2113 (AU). **ROYCHOUDHRY, Anil** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU). **BORZYCKI, Andrew, Gerard** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU). **CHIN, Huai, Chiun** [MY/AU]; c/o Citrix systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde NSW 2113 (AU). **MAZZAFERRI, Richard, James** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU). **BISSETT, Nicholas, Alexander** [GB/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU). **MUIR, Jeffrey, Dale** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU).

(74) Agent: **LANZA, John, D.**; Choate, Hall & Stewart, Two International Place, Boston, MA 02110 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN,

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR MOVING PROCESSES BETWEEN ISOLATION ENVIRONMENTS



(57) Abstract: A method for moving an executing process from a source isolation scope to a target isolation scope includes the step of determining that the process is in a state suitable for moving. The association of the process changes from a source isolation scope to a target isolation scope. A rule loads in association with the target isolation scope.

WO 2006/039181 A1

CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR MOVING PROCESSES BETWEEN ISOLATION ENVIRONMENTS

Field of the Invention

The invention relates to managing execution of software applications by computers and, in particular, to methods and apparatus for moving processes between isolation enviroments.

Background of the Invention

Computer software application programs, during execution and installation, make use of a variety of native resources provided by the operating system of a computer. A traditional, single-user computer is depicted in FIG. 1A. As shown in FIG. 1A, native resources provided by the operating system 100 may include a file system 102, a registry database 104, and objects 106. The file system 102 provides a mechanism for an application program to open, create, read, copy, modify, and delete data files 150, 152. The data files 150, 152 may be grouped together in a logical hierarchy of directories 160, 162. The registry database 104 stores information regarding hardware physically attached to the computer, which system options have been selected, how computer memory is set up, various items of application-specific data, and what application programs should be present when the operating system 100 is started. As shown in FIG. 1A, the registry database 104 is commonly organized in a logical hierarchy of "keys" 170, 172 which are containers for registry values. The operating system 100 may also provide a number of communication and synchronization objects 106, including semaphores, sections, mutexes, timers, mutants, and pipes. Together, the file system 102, registry database 104, objects 106, and any other native resources made available by the operating system 100 will be referred to

throughout this document as the "system layer" 108. The resources provided by the system layer 108 are available to any application or system program 112, 114.

A problem arises, however, when execution or installation of two incompatible application programs 112, 114 is attempted. As shown in FIG. 1A, two application programs, APP1 112 and APP2 114, execute "on top of" the operating system 100, that is, the application programs make use of functions provided by the operating system to access native resources. The application programs are said to be incompatible with one another when they make use of native resources 102, 104, 106 in a mutually exclusive manner either during execution or during the installation process. APP1 112 may require, or attempt to install, a file located by the pathname c:\windows\system32\msvcrt.dll and APP2 114 may require, or attempt to install, a second, different file that is located by the same pathname. In this case, APP1 112 and APP2 114 cannot be executed on the same computer and are said to be incompatible with one another. A similar problem may be encountered for other native resources. This is, at best, inconvenient for a user of the computer who requires installation or execution of both APP1 112 and APP2 114 together in the same operating system 100 environment.

FIG. 1B depicts a multi-user computer system supporting concurrent execution of application programs 112, 114, 112', 114' on behalf of several users. As shown in FIG. 1B, a first instance of APP1 112 and a first instance of APP2 114 are executed in the context of a first user session 110, a second instance of APP1 112' is executed in the context of a second user session 120, and a second instance of APP2 114' is executed in the context of a third user session 130. In this environment, a problem arises if both instances of APP1 112, 112' and both instances of APP2 114, 114' make use of native resources 102, 104, 106 as if only a single user executes the application. For example, the APP1 112 may store application specific data in a registry key 170. When the

first instance of APP1 112 executing in the first user context 110 and the second instance of APP1 112' executing in the second user context 120 both attempt to store configuration data to the same registry key 170, incorrect configuration data will be stored for one of the users. A similar problem can occur for other native resources.

The present invention addresses these application program compatibility and sociability problems.

Summary of the Invention

The present invention allows installation and execution of application programs that are incompatible with each other, and incompatible versions of the same application program, on a single computer. In addition, it allows the installation and execution on a multi-user computer of programs that were created for a single-user computer or were created without consideration for issues that arise when executing on a multi-user computer. The methods and apparatus described are applicable to single-user computing environments, which includes environments in which multiple users may use a single computer one after the other, as well as multi-user computing environments, in which multiple users concurrently use a single computer. The present invention virtualizes user and application access to native resources, such as the file system, the registry database, system objects, window classes and window names, without modification of the application programs or the underlying operating system. In addition, virtualized native resources may be stored in native format (that is, virtualized files are stored in the file system, virtualized registry entries are stored in the registry database, etc.) allowing viewing and management of virtualized resources to be accomplished using standard tools and techniques. The provided isolation scopes may be taken advantage of by moving processes between isolation scopes, into isolation scopes, or out of isolation scopes while those processes are executing, allowing the view of native resources provided to those application to change.

In one aspect, the invention relates to a method for moving an executing process from a first isolation scope to a second isolation scope. A determination is made that the process is in a state suitable for moving. The association of the process changes from a first isolation scope to a second isolation scope. A rule loads in association with the second isolation scope.

In one embodiment, determining whether the process is in a state suitable for moving is accomplished by monitoring whether the process is processing a request. In some embodiments, the process is put in a state suitable for moving. In another embodiment, the association of the process from the first isolation scope to the second isolation scope changes in a file system filter driver. In yet another embodiment, the association of the process from the first isolation scope to the second isolation scope changes in one of a kernel hooking function and a user mode hooking function.

In another aspect, a method relates to moving an executing process into an isolation scope. A determination is made that the process is in a state suitable for moving. There is an association of the process with an isolation scope. A rule loads in association with the isolation scope.

In one embodiment, determining whether the process is in a state suitable for moving is accomplished by monitoring whether the process is processing a request. In some embodiments, the process is put in a state suitable for moving. In other embodiments, information associating the process with the isolation scope is written to a rules engine.

Brief Description of the Drawings

The invention is pointed out with particularity in the appended claims. The advantages of the invention described above, as well as further advantages of the invention, may be better understood by reference to the following description taken in conjunction with the accompanying drawings, in which:

FIG. 1A is a block diagram of a prior art operating system environment supporting execution of two application programs on behalf of a user;

FIG. 1B is a block diagram of a prior art operating system environment supporting concurrent execution of multiple applications on behalf of several users;

FIG. 2A is a block diagram of an embodiment of a computer system having reduced application program compatibility and sociability problems;

FIG. 2B is a diagram of an embodiment of a computer system having reduced application program compatibility and sociability problems;

FIG. 2C is a flowchart showing one embodiment of steps taken to associate a process with an isolation scope;

FIG 3A is a flowchart showing one embodiment of steps taken to virtualize access to native resources in a computer system;

FIG. 3B is a flowchart showing an embodiment of steps taken to identify a replacement instance in execute mode;

FIG. 3C is a flowchart depicting one embodiment of steps taken in install mode to identify a literal resource when a request to open a native resource is received that indicates the resource is being opened with the intention of modifying it;

FIG. 3D is a flowchart depicting one embodiment of steps taken in install mode to identify the literal resource when a request to create a virtual resource is received;

FIG. 4 is a flowchart depicting one embodiment of the steps taken to open an entry in a file system in the described virtualized environment;

FIG. 5 is a flowchart depicting one embodiment of the steps taken to delete an entry from a file system in the described virtualized environment;

FIG. 6 is a flowchart depicting one embodiment of the steps taken to enumerate entries in a file system in the described virtualized environment;

FIG. 7 is a flowchart depicting one embodiment of the steps taken to create an entry in a file system in the described virtualized environment;

FIG. 8 is a flowchart depicting one embodiment of the steps taken to open a registry key in the described virtualized environment;

FIG. 9 is a flowchart depicting one embodiment of the steps taken to delete a registry key in the described virtualized environment;

FIG. 10 is a flowchart depicting one embodiment of the steps taken to enumerate subkeys of a key in a registry database in the described virtualized environment;

FIG. 11 is a flowchart depicting one embodiment of the steps taken to create a registry key in the described virtualized environment;

FIG. 12 is a flowchart depicting one embodiment of the steps taken to virtualize access to named objects;

FIG. 13 is a flowchart depicting one embodiment of the steps taken to virtualize window names and window classes in the described environment;

FIGs. 13A is a flowchart depicting one embodiment of the steps taken to determine literal window names and window class names;

FIG. 14 is a flowchart depicting one embodiment of the steps taken to invoke an out-of-process COM server in the described virtualized environment;

FIG. 15 is a flowchart depicting one embodiment of the steps taken to virtualize application invocation using file-type association; and

FIG. 16 is a flowchart depicting one embodiment of the steps taken to move a process from a source isolation scope to a target isolation scope.

Index

The index is intended to help the reader follow the discussion of the invention:

1.0 Isolation Environment Conceptual Overview
- 1.1 Application Isolation
- 1.2 User Isolation
- 1.3 Aggregate view of native resources
- 1.4 Association of processes with isolation scopes
  - 1.4.1 Association of out-of-scope processes with isolation scopes

2.0 Virtualization Mechanism Overview

3.0 Installation into an isolation environment

4.0 Execution in an isolation environment
- 4.1 File system virtualization
  - 4.1.1 File System Open Operations
  - 4.1.2 File System Delete Operations
  - 4.1.3 File System Enumerate Operations
  - 4.1.4 File System Create Operations
  - 4.1.5 Short Filename Management
- 4.2 Registry virtualization
  - 4.2.1 Registry Open Operations
  - 4.2.2 Registry Delete Operations
  - 4.2.3 Registry Enumerate Operations
  - 4.2.4 Registry Create Operations
- 4.3 Named object virtualization
- 4.4 Window name virtualization
- 4.5 Out-of-process COM server virtualization
- 4.6 Virtualized file-type association
- 4.7 Dynamic movement of processes between isolation environments

Detailed Description of the Invention

1.0 Isolation Environment Conceptual Overview

1.1 Application Isolation

Referring now to FIG. 2A, one embodiment of a computer running under control of an operating system 100 that has reduced application compatibility and application sociability problems is shown. The operating system 100 makes available various native resources to application programs 112, 114 via its system layer 108. The view of resources embodied by the system layer 108 will be termed the "system scope". In order to avoid conflicting access to native resources 102, 104, 106, 107 by the application programs 112, 114, an isolation environment 200 is provided. As shown in FIG. 2A, the isolation environment 200 includes an application isolation layer 220 and a user isolation layer 240. Conceptually, the isolation environment 200 provides, via the application isolation layer 220, an application program 112, 114, with a unique view of native resources, such as the file system 102, the registry 104, objects 106, and window names 107. Each isolation layer modifies the view of native resources provided to an application. The modified view of native resources provided by a layer will be referred to as that layer's "isolation scope". As shown in FIG. 2A, the application isolation layer includes two application isolation scopes 222, 224. Scope 222 represents the view of native resources provided to application 112 and scope 224 represents the view of native resources provided to application 114. Thus, in the embodiment shown in FIG. 2A, APP1 112 is provided with a specific view of the file system 102', while APP2 114 is provided with another view of the file system 102" which is specific to it. In some embodiments, the application isolation layer 220 provides a specific view of native resources 102, 104, 106, 107 to each individual application program executing on top of the operating system 100. In other embodiments, application programs 112, 114 may be grouped into sets and, in these embodiments, the application isolation layer 220 provides a specific view of native resources for each set of application

programs. Conflicting application programs may be put into separate groups to enhance the compatibility and sociability of applications. In still further embodiments, the applications belonging to a set may be configured by an administrator. In some embodiments, a "passthrough" isolation scope can be defined which corresponds exactly to the system scope. In other words, applications executing within a passthrough isolation scope operate directly on the system scope.

In some embodiments, the application isolation scope is further divided into layered sub-scopes. The main sub-scope contains the base application isolation scope, and additional sub-scopes contain various modifications to this scope that may be visible to multiple executing instances of the application. For example, a sub-scope may contain modifications to the scope that embody a change in the patch level of the application or the installation or removal of additional features. In some embodiments, the set of additional sub-scopes that are made visible to an instance of the executing application is configurable. In some embodiments, that set of visible sub-scopes is the same for all instances of the executing application, regardless of the user on behalf of which the application is executing. In others, the set of visible sub-scopes may vary for different users executing the application. In still other embodiments, various sets of sub-scopes may be defined and the user may have a choice as to which set to use. In some embodiments, sub-scopes may be discarded when no longer needed. In some embodiments, the modifications contained in a set of sub-scopes may be merged together to form a single sub-scope.

1.2 User Isolation

Referring now to FIG. 2B, a multi-user computer having reduced application compatibility and application sociability problems is depicted. The multi-user computer includes native resources 102, 104, 106, 107 in the system layer 108, as well as the isolation environment 200 discussed immediately above. The application isolation layer 220 functions as discussed above,

providing an application or group of applications with a modified view of native resources. The user isolation layer 240, conceptually, provides an application program 112, 114, with a view of native resources that is further altered based on user identity of the user on whose behalf the application is executed. As shown in FIG. 2B, the user isolation layer 240 may be considered to comprise a number of user isolation scopes 242', 242", 242''', 242'''', 242''''', 242'''''' (generally 242). A user isolation scope 242 provides a user-specific view of application-specific views of native resources. For example, APP1 112 executing in user session 110 on behalf of user "a" is provided with a file system view 102'(a) that is altered or modified by both the user isolation scope 242' and the application isolation scope 222.

Put another way, the user isolation layer 240 alters the view of native resources for each individual user by "layering" a user-specific view modification provided by a user isolation scope 242' "on top of" an application-specific view modification provided by an application isolation scope 222, which is in turn "layered on top of" the system-wide view of native resources provided by the system layer. For example, when the first instance of APP1 112 accesses an entry in the registry database 104, the view of the registry database specific to the first user session and the application 104'(a) is consulted. If the requested registry key is found in the user-specific view of the registry 104'(a), that registry key is returned to APP1 112. If not, the view of the registry database specific to the application 104' is consulted. If the requested registry key is found in the application-specific view of the registry 104', that registry key is returned to APP1 112. If not, then the registry key stored in the registry database 104 in the system layer 108 (i.e. the native registry key) is returned to APP1 112.

In some embodiments, the user isolation layer 240 provides an isolation scope for each individual user. In other embodiments, the user isolation layer 240 provides an isolation scope for a group of users, which may be defined by roles within the organization or may be predetermined by an administrator. In

still other embodiments, no user isolation layer 240 is provided. In these embodiments, the view of native resources seen by an application program is that provided by the application isolation layer 220. The isolation environment 200, although described in relation to multi-user computers supporting concurrent execution of application programs by various users, may also be used on single-user computers to address application compatibility and sociability problems resulting from sequential execution of application programs on the same computer system by different users, and those problems resulting from installation and execution of incompatible programs by the same user.

In some embodiments, the user isolation scope is further divided into sub-scopes. The modifications by the user isolation scope to the view presented to an application executing in that scope is the aggregate of the modifications contained within each sub-scope in the scope. Sub-scopes are layered on top of each other, and in the aggregate view modifications to a resource in a higher sub-scope override modifications to the same resource in lower layers.

In some of these embodiments, one or more of these sub-scopes may contain modifications to the view that are specific to the user. In some of these embodiments, one or more sub-scopes may contain modifications to the view that are specific to sets of users, which may be defined by the system administrators or defined as a group of users in the operating system. In some of these embodiments, one of these sub-scopes may contain modifications to the view that are specific to the particular login session, and hence that are discarded when the session ends. In some of these embodiments, changes to native resources by application instances associated with the user isolation scope always affects one of these sub-scopes, and in other embodiments those changes may affect different sub-scopes depending on the particular resource changed.

1.3 Aggregate Views of Native Resources

The conceptual architecture described above allows an application executing on behalf of a user to be presented with an aggregate, or unified, virtualized view of native resources, specific to that combination of application and user. This aggregated view may be referred to as the "virtual scope". The application instance executing on behalf of a user is presented with a single view of native resources reflecting all operative virtualized instances of the native resources. Conceptually this aggregated view consists firstly of the set of native resources provided by the operating system in the system scope, overlaid with the modifications embodied in the application isolation scope applicable to the executing application, further overlaid with the modifications embodied in the user isolation scope applicable to the application executing on behalf of the user. The native resources in the system scope are characterized by being common to all users and applications on the system, except where operating system permissions deny access to specific users or applications. The modifications to the resource view embodied in an application isolation scope are characterized as being common to all instances of applications associated with that application isolation scope. The modifications to the resource view embodied in the user isolation scope are characterized as being common to all applications associated with the applicable application isolation scope that are executing on behalf of the user associated with the user isolation scope.

This concept can be extended to sub-scopes; the modifications to the resource view embodied in a user sub-scope are common to all applications associated with the applicable isolation sub-scope executing on behalf of a user, or group of users, associated with a user isolation sub-scope. Throughout this description it should be understood that whenever general reference is made to "scope," it is intended to also refer to sub-scopes, where those exist.

When an application requests enumeration of a native resource, such as a portion of the file system or registry database, a virtualized enumeration is constructed by first enumerating the "system-scoped" instance of the native

resource, that is, the instance found in the system layer, if any. Next, the "application-scoped" instance of the requested resource, that is the instance found in the appropriate application isolation scope, if any, is enumerated. Any enumerated resources encountered in the application isolation scope are added to the view. If the enumerated resource already exists in the view (because it was present in the system scope, as well), it is replaced with the instance of the resource encountered in the application isolation scope. Similarly, the "user-scoped" instance of the requested resource, that is the instance found in the appropriate user isolation scope, if any, is enumerated. Again, any enumerated resources encountered in the user isolation scope are added to the view. If the native resource already exists in the view (because it was present in the system scope or in the appropriate application isolation scope), it is replaced with the instance of the resource encountered in the user isolation scope. In this manner, any enumeration of native resources will properly reflect virtualization of the enumerated native resources. Conceptually the same approach applies to enumerating an isolation scope that comprises multiple sub-scopes. The individual sub-scopes are enumerated, with resources from higher sub-scopes replacing matching instances from lower sub-scopes in the aggregate view.

In other embodiments, enumeration may be performed from the user isolation scope layer down to the system layer, rather than the reverse. In these embodiments, the user isolation scope is enumerated. Then the application isolation scope is enumerated and any resource instances appearing in the application isolation scope that were not enumerated in the user isolation scope are added to the aggregate view that is under construction. A similar process can be repeated for resources appearing only in the system scope.

In still other embodiments, all isolation scopes may be simultaneously enumerated and the respective enumerations combined.

If an application attempts to open an existing instance of a native resource with no intent to modify that resource, the specific instance that is returned to the

application is the one that is found in the virtual scope, or equivalently the instance that would appear in the virtualized enumeration of the parent of the requested resource. From the point of view of the isolation environment, the application is said to be requesting to open a "virtual resource", and the particular instance of native resource used to satisfy that request is said to be the "literal resource" corresponding to the requested resource.

If an application executing on behalf of a user attempts to open a resource and indicates that it is doing so with the intent to modify that resource, that application instance is normally given a private copy of that resource to modify, as resources in the application isolation scope and system scope are common to applications executing on behalf of other users. Typically a user-scoped copy of the resource is made, unless the user-scoped instance already exists. The definition of the aggregate view provided by a virtual scope means that the act of copying an application-scoped or system-scoped resource to a user isolation scope does not change the aggregate view provided by the virtual scope for the user and application in question, nor for any other user, nor for any other application instance. Subsequent modifications to the copied resource by the application instance executing on behalf of the user do not affect the aggregate view of any other application instance that does not share the same user isolation scope. In other words, those modifications do not change the aggregate view of native resources for other users, or for application instances not associated with the same application isolation scope.

1.4 Association of processes with isolation scopes

Applications may be installed into a particular isolation scope (described below in more detail). Applications that are installed into an isolation scope are always associated with that scope. Alternatively, applications may be launched into a particular isolation scope, or into a number of isolation scopes. In effect, an application is launched and associated with one or more isolation scopes. The associated isolation scope, or scopes, provide the process with a particular

view of native resources. Applications may also be launched into the system scope, that is, they may be associated with no isolation scope. This allows for the selective execution of operating system applications such as Internet Explorer, as well as third party applications, within an isolation environment.

This ability to launch applications within an isolation scope regardless of where the application is installed mitigates application compatibility and sociability issues without requiring a separate installation of the application within the isolation scope. The ability to selectively launch installed applications in different isolation scopes provides the ability to have applications which need helper applications (such as Word, Notepad, etc.) to have those helper applications launched with the same rule sets.

Further, the ability to launch an application within multiple isolated environments allows for better integration between isolated applications and common applications.

Referring now to FIG. 2C, and in brief overview, a method for associating a process with an isolation scope includes the steps of launching the process in a suspended state (step 282). The rules associated with the desired isolation scope are retrieved (step 284) and an identifier for the process and the retrieved rules are stored in a memory element (step 286) and the suspended process is resumed (step 288). Subsequent calls to access native resources made by the process are intercepted or hooked (step 290) and the rules associated with the process identifier, if any, are used to virtualize access to the requested resource (step 292).

Still referring to FIG. 2C, and in more detail, a process is launched in a suspended state (step 282). In some embodiments, a custom launcher program is used to accomplish this task. In some of these embodiments, the launcher is specifically designed to launch a process into a selected isolation scope. In other embodiments, the launcher accepts as input a specification of the desired isolation scope, for example, by a command line option.

The rules associated with the desired isolation scope are retrieved (step 284). In some embodiments, the rules are retrieved from a persistent storage element, such as a hard disk drive or other solid state memory element. The rules may be stored as a relational database, flat file database, tree-structured database, binary tree structure, or other persistent data structure. In other embodiments, the rules may be stored in a data structure specifically configured to store them.

An identifier for the process, such as a process id (PID), and the retrieved rules are stored in a memory element (step 286). In some embodiments, a kernel mode driver is provided that receives operating system messages concerning new process creation. In these embodiments, the PID and the retrieved rules may be stored in the context of the driver. In other embodiments, a file system filter driver, or mini-filter, is provided that intercepts native resource requests. In these embodiments, the PID and the retrieved rules may be stored in the filter. In other embodiments still, all interception is performed by user-mode hooking and no PID is stored at all. The rules are loaded by the user-mode hooking apparatus during the process initialization, and no other component needs to know the rules that apply to the PID because rule association is performed entirely in-process.

The suspended process is resumed (step 288) and subsequent calls to access native resources made by the process are intercepted or hooked (step 290) and the rules associated with the process identifier, if any, are used to virtualize access to the requested resource (step 292). In some embodiments, a file system filter driver, or mini-filter, intercepts requests to access native resources and determines if the process identifier associated with the intercepted request has been associated with a set of rules. If so, the rules associated with the stored process identifier are used to virtualize the request to access native resources. If not, the request to access native resources is passed through unmodified. In other embodiments, a dynamically-linked library is loaded into the

newly-created process and the library loads the isolation rules. In still other embodiments, both kernel mode techniques (hooking, filter driver, mini-filter) and user-mode techniques are used to intercept calls to access native resources. For embodiments in which a file system filter driver stores the rules, the library may load the rules from the file system filter driver.

Processes that are "children" of processes associated with isolation scopes are associated with the isolation scopes of their "parent" process. In some embodiments, this is accomplished by a kernel mode driver notifying the file system filter driver when a child process is created. In these embodiments, the file system filter driver determines if the process identifier of the parent process is associated with an isolation scope. If so, file system filter driver stores an association between the process identifier for the newly-created child process and the isolation scope of the parent process. In other embodiments, the file system filter driver can be called directly from the system without use of a kernel mode driver. In other embodiments, in processes that are associated with isolation scopes, operating system functions that create new processes are hooked or intercepted. When request to create a new process are received from such a process, the association between the new child process and the isolation scope of the parent is stored.

In some embodiments, a scope or sub-scope may be associated with an individual thread instead of an entire process, allowing isolation to be performed on a per-thread basis. In some embodiments, per-thread isolation may be used for Services and COM+ servers.

1.4.1 Associating out-of-scope processes with isolation scopes

Another aspect of this invention is the ability to associate any application instance with any application isolation scope, regardless of whether the application was installed into that application isolation scope, another application isolation scope or no application isolation scope. Applications which were not installed into a particular application scope can nevertheless be executed on

behalf of a user in the context of an application isolation scope and the corresponding user isolation scope, because their native resources are available to them via the aggregated virtual scope formed by the user isolation scope, application isolation scope and system scope. Where it is desired to run an application in an isolation scope, this provides the ability for applications installed directly into the system scope to be run within the isolation scope without requiring a separate installation of the application within the isolation scope. This also provides the ability for applications installed directly into the system scope to be used as helper applications in the context of any isolation scope.

Each application instance, including all of the processes that make up the executing application, is associated with either zero or one application isolation scopes, and by extension exactly zero or one corresponding user isolation scopes. This association is used by the rules engine when determining which rule, if any, to apply to a resource request. The association does not have to be to the application isolation scope that the application was installed into, if any. Many applications that are installed into an isolation scope will not function correctly when running in a different isolation scope or no isolation scope because they cannot find necessary native resources. However, because an isolation scope is an aggregation of resource views including the system scope, an application installed in the system scope can generally function correctly inside any application isolation scope. This means that helper programs, as well as out-of-process COM servers, can be invoked and executed by an application executing on behalf of a user in a particular isolation scope.

In some embodiments, applications that are installed in the system scope are executed in an isolation scope for the purpose of identifying what changes are made to the computer's files and configuration settings as a result of this execution. As all affected files and configuration settings are isolated in the user isolation scope, these files and configuration settings are easily identifiable. In some of these embodiments, this is used in order to report on the changes made

to the files and configuration settings by the application. In some embodiments, the files and configuration settings are deleted at the end of the application execution, which effectively ensures that no changes to the computer's files and configuration setting are stored as a result of application execution. In still other embodiments, the files and configuration settings are selectively deleted or not deleted at the end of the application execution, which effectively ensures that only some changes to the computer's files and configuration setting are stored as a result of application execution.

2.0 Virtualization Mechanism Overview

Referring now to FIG. 3A, one embodiment of the steps to be taken to virtualize access to native resources in execute mode, which will be distinguished from install mode below, is shown. In brief overview, a request to access a native resource is intercepted or received (step 302). The request identifies the native resource to which access is sought. The applicable rule regarding how to treat the received access request is determined (step 304). If the rule indicates the request should be ignored, the access request is passed without modification to the system layer (step 306) and the result returned to the requestor (step 310). If the rule indicates that the access request should be either redirected or isolated, the literal instance of the resource that satisfies the request is identified (step 308), a modified or replacement request for the literal resource is passed to the system layer (step 306) and the result is returned to the requestor (step 310).

Still referring to FIG. 3, and in more detail, a request identifying a native resource is intercepted or received (step 302). In some embodiments, requests for native resources are intercepted by "hooking" functions provided by the operating system for applications to make native resource requests. In specific embodiments, this is implemented as a dynamically-linked library that is loaded into the address space of every new process created by the operating system, and which performs hooking during its initialization routine. Loading a DLL into every process may be achieved via a facility provided by the operating system, or

alternatively by modifying the executable image's list of DLLs to import, either in the disk file, or in memory as the executable image for the process is loaded from disk. In other embodiments, function hooking is performed by a service, driver or daemon. In other embodiments, executable images, including shared libraries and executable files, provided by the operating system may be modified or patched in order to provide function hooks or to directly embody the logic of this invention. For specific embodiments in which the operating system is a member of the Microsoft WINDOWS family of operating systems, interception may be performed by a kernel mode driver hooking the System Service Dispatch Table. In still other embodiments, the operating system may provide a facility allowing third parties to hook functions that request access to native resources. In some of these embodiments, the operating system may provide this facility via an application programming interface (API) or a debug facility.

In other embodiments, the native resource request is intercepted by a filter in the driver stack or handler stack associated with the native resource. For example, some members of the family of Microsoft WINDOWS operating systems provide the capability to plug a third-party filter driver or mini-filter into the file system driver stack and a file system filter driver or mini-filter may be used to provide the isolation functions described below. In still other embodiments the invention comprises a file system implementation that directly incorporates the logic of this invention. Alternatively, the operating system may be rewritten to directly provide the functions described below. In some embodiments, a combination of some or all of the methods listed above to intercept or receive requests for resources may be simultaneously employed.

In many embodiments, only requests to open an existing native resource or create a new native resource are hooked or intercepted. In these embodiments, the initial access to a native resource is the access that causes the resource to be virtualized. After the initial access, the requesting application program is able to communicate with the operating system concerning the

virtualized resource using a handle or pointer or other identifier provided by the operating system that directly identifies the literal resource. In other embodiments, other types of requests to operate on a virtualized native resource are also hooked or intercepted. In some of these embodiments, requests by the application to open or create virtual resources return virtual handles that do not directly identify the literal resource, and the isolation environment is responsible for translating subsequent requests against virtual handles to the corresponding literal resource. In some of those embodiments, additional virtualization operations can be deferred until proven necessary. For example, the operation of providing a private modifiable copy of a resource to an isolation scope can be deferred until a request to change the resource is made, rather than when the resource is opened in a mode that allows subsequent modification.

Once the native resource request is intercepted or received, the applicable rule determining how to treat the particular request is determined (step 304). The most applicable rule may be determined by reference to a rules engine, a database of rules, or a flat file containing rules organized using an appropriate data structure such as a list or a tree structure. In some embodiments, rules are accorded a priority that determines which rule will be regarded as most applicable when two or more rules apply. In some of these embodiments, rule priority is included in the rules themselves or, alternatively, rule priority may be embedded in the data structure used to store the rules, for example, rule priority may be indicated by a rule's position in a tree structure. The determined rule may include additional information regarding how to process the virtualized resource request such as, for example, to which literal resource to redirect the request. In a specific embodiment a rule is a triple comprising a filter field, an action field, and data field. In this embodiment, the filter field includes the filter used to match received native resource requests to determine if the rule is valid for the requested resource name. The action field can be "ignore," "redirect," or "isolate". The data field may be any additional information concerning the action

to be taken when the rule is valid, including the function to be used when the rule is valid.

A rule action of "ignore" means the request directly operates on the requested native resource in the system scope. That is, the request is passed unaltered to the system layer 108 (step 306) and the request is fulfilled as if no isolation environment 200 exists. In this case, the isolation environment is said to have a "hole", or the request may be referred to as a "passthrough" request.

If the rule action indicates that the native resource request should be redirected or isolated, then the literal resource that satisfies the request is identified (step 308).

A rule action of "redirect" means that the request directly operates on a system-scoped native resource, albeit a different resource than specified in the request. The literal resource is identified by applying a mapping function specified in, or implied by, the data field of the determined rule to the name of the requested native resource. In the most general case the literal native resource may be located anywhere in the system scope. As a simple example, the rule {prefix_match ("c:\temp\", resource name), REDIRECT, replace_prefix ("c:\temp\", "d:\wutemp\", resource name)} will redirect a requested access to the file c:\temp\examples\d1.txt to the literal file d:\wutemp\examples\d1.txt. The mapping function included in the data field of the rule and the matching function may be further generalized to support complex behaviors by, for example, using regular expressions. Some embodiments may provide the ability to specify mapping functions that locate the literal resource within the user isolation scope or a sub-scope applicable to the application executing on behalf of the user, or the application isolation scope or a sub-scope applicable to the application. Further embodiments may provide the ability to specify mapping functions that locate the literal resource within the application isolation scope applicable to a different application in order to provide a controlled form of interaction between isolated applications. In some particular embodiments, the "redirect" action can

be configured to provide behavior equivalent to the "ignore" rule action. In these embodiments, the literal resource is exactly the requested native resource. When this condition is configured, the isolation environment may be said to have a "hole," or the request may be referred to as a "passthrough" request.

A rule action of "isolate" means that the request operates on a literal resource that is identified using the appropriate user isolation scope and application isolation scope. That is, the identifier for the literal resource is determined by modifying the identifier for the requested native resource using the user isolation scope, the application isolation scope, both scopes, or neither scope. The particular literal resource identified depends on the type of access requested and whether instances of the requested native resource already exist in the applicable user isolation scope, the applicable application isolation scope and the system scope.

FIG. 3B depicts one embodiment of steps taken to identify the literal resource (step 306 in FIG. 3A) when a request to open a native resource is received that indicates the resource is being opened with the intention of modifying it. Briefly, a determination is made whether the user-scoped instance, that is, an instance that exists in an applicable user scope or user sub-scope, of the requested native resource exists (step 354). If so, the user-scoped instance is identified as the literal resource for the request (step 372), and that instance is opened and returned to the requestor. If the user-scoped instance does not exist, a determination whether the application-scoped instance of the requested native resource exists is made (step 356). If the application-scoped instance exists, it is identified as the "candidate" resource instance (step 359), and permission data associated with the candidate instance is checked to determine if modification of that instance is allowed (step 362). If no application-scoped instance exists, then a determination is made whether the system-scoped instance of the requested native resource exists (step 358). If it does not, an error condition is returned to the requestor indicating that the requested

virtualized resource does not exist in the virtual scope (step 360). However, if the system-scoped resource exists, it is identified as the candidate resource instance (step 361), and permission data associated with the candidate instance is checked to determine if modification of that instance is allowed (step 362). If not, an error condition is returned to the requestor (step 364) indicating that modification of the virtualized resource is not allowed. If the permission data indicates that the candidate resource may be modified, a user-scoped copy of the candidate instance of the native resource is made (step 370), the user-scoped instance is identified as the literal instance for the request (step 372), and is opened and returned to the requestor.

Still referring to FIG. 3B, and in more detail, a determination is made whether the user-scoped resource exists, or in other words, whether the requested resource exists in the applicable user scope or sub-scope (step 354). The applicable user scope or sub-scope is the scope associated with the user that is layered on the application isolation scope associated with the application making the request. The user isolation scope or a sub-scope, in the file system case, may be a directory under which all files that exist in the user isolation scope are stored. In some of these embodiments, the directory tree structure under the user isolation directory reflects the path of the requested resource. For example, if the requested file is c:\temp\test.txt and the user isolation scope directory is d:\user1\app1\, then the path to the user-scoped literal file may be d:\user1\app1\c\temp\test.txt. In other embodiments, the path to the user-scoped literal may be defined in a native naming convention. For example, the path to the user-scoped literal file may be d:\user1\app1\device\harddisk1\temp\test.txt. In still other embodiments, the user-scoped files may all be stored in a single directory with names chosen to be unique and a database may be used to store the mapping between the requested file name and the name of the corresponding literal file stored in the directory. In still other embodiments, the contents of the literal files may be

stored in a database. In still other embodiments, the native file system provides the facility for a single file to contain multiple independent named "streams", and the contents of the user-scoped files are stored as additional streams of the associated files in the system scope. Alternatively, the literal files may be stored in a custom file system that may be designed to optimize disk usage or other criteria of interest.

If the user-scoped resource instance does not exist, then a determination is made whether the application-scoped resource exists, or in other words whether the requested resource exists in the application isolation scope (step 356). The methods described above are used to make this determination. For example, if the requested file is c:\temp\test.txt and the application isolation scope directory is e:\app1\, then the path to the application-scoped file may be e:\app1\c\temp\test.txt. As above, the path to the application-scoped file may be stored in a native naming convention. The embodiments described above may also apply to the application isolation scope.

If the application-scoped resource does not exist, then a determination is made whether the system-scoped resource exists, or in other words, whether the requested resource exists in the system scope (step 358). For example, if the requested file is c:\temp\test.txt then the path to the system-scoped file is c:\temp\test.txt. If the requested resource does not exist in the system scope, an indication that the requested resource does not exist in the virtual scope is returned to the requestor (step 360).

Whether the candidate resource instance for the requested resource is located in the application isolation scope or in the system scope, a determination is made whether modification of the candidate resource instance is allowed (step 362). For example, the candidate native resource instance may have associated native permission data indicating that modification of the candidate instance is not allowed by that user. Further, the rules engine may include configuration settings instructing the isolation environment to obey or override the native

permission data for virtualized copies of resources. In some embodiments, the rules may specify for some virtual resources the scope in which modifications are to occur, for example the system scope or the application isolation scope or a sub-scope, or the user isolation scope or a sub-scope. In some embodiments, the rules engine may specify configuration settings that apply to subsets of the virtualized native resources based on hierarchy or on type of resource accessed. In some of these embodiments, the configuration settings may be specific to each atomic native resource. In another example, the rules engine may include configuration data that prohibits or allows modification of certain classes of files, such as executable code or MIME types or file types as defined by the operating system.

If the determination in step 362 is that modification of the candidate resource instance is not allowed, then an error condition is returned to the requestor indicating that write access to the virtual resource is not allowed (step 364). If the determination in step 362 is that modification of the candidate resource instance is allowed, then the candidate instance is copied to the appropriate user isolation scope or sub-scope (step 370). For embodiments in which the logical hierarchy structure of the requested native resource is maintained in the isolation scopes, copying the candidate instance of the resource to the user isolation scope may require the creation in the user isolation scope of hierarchy placeholders. A hierarchy placeholder is a node that is placed in the hierarchy to correctly locate the copied resource in the isolation scope. A hierarchy placeholder stores no data, is identified as a placeholder node, and "does not exist" in the sense that it cannot be the literal resource returned to a requestor. In some embodiments, the identification of a node as a placeholder node is made by recording the fact in metadata attached to the node, or to the parent of the node, or to some other related entity in the system layer. In other embodiments, a separate repository of placeholder node names is maintained.

In some embodiments, the rules may specify that modifications to particular resources may be made at a particular scope, such as the application isolation scope. In those cases the copy operation in step 370 is expanded to determine whether modification to the candidate resource instance is allowed at the scope or sub-scope in which it is found. If not, the candidate resource instance is copied to the scope or sub-scope in which modification is allowed, which may not always be the user isolation scope, and the new copy is identified as the literal resource instance (step 372). If so, the candidate resource instance is identified as the literal instance (step 372), and is opened and the result returned to the requestor (step 306).

Referring back to FIG. 3A, the literal resource instance, whether located in step 354 or created in step 370, is opened (step 306) and returned to the requestor (step 310). In some embodiments, this is accomplished by issuing an "open" command to the operating system and returning to the requestor the response from the operating system to the "open" command.

If an application executing on behalf of a user deletes a native resource, the aggregated view of native resources presented to that application as the virtual scope must reflect the deletion. A request to delete a resource is a request for a special type of modification, albeit one that modifies the resource by removing its existence entirely. Conceptually a request to delete a resource proceeds in a similar manner to that outlined in Figure 3A, including the determination of the literal resource as outlined in Figure 3B. However, step 306 operates differently for isolated resources and for redirected or ignored resources. For redirect and ignore, the literal resource is deleted from the system scope. For isolate, the literal resource is "virtually" deleted, or in other words the fact that it has been deleted is recorded in the user isolation scope. A deleted node contains no data, is identified as deleted, and it and all of its descendants "do not exist". In other words, if it is the resource or the ancestor of a resource that would otherwise satisfy a resource request a "resource not found"

error is returned to the requestor. Further details will be outlined in Section 4. In some embodiments, the identification of a node as a deleted node is made by recording the fact in metadata attached to the node, or to the parent of the node, or to some other related entity in the system layer. In other embodiments, a separate repository of deleted node names is maintained, for example, in a separate sub-scope.

3.0 Installation into an isolation environment

The application isolation scope described above can be considered as the scope in which associated application instances share resources independently of any user, or equivalently on behalf of all possible users, including the resources that those application instances create. The main class of such resources is the set created when an application is installed onto the operating system. As shown in Fig. 1A, two incompatible applications cannot both be installed into the same system scope, but this problem can be resolved by installing at least one of those applications into an isolation environment.

An isolation scope, or an application instance associated with an isolation scope, can be operated in an "install mode" to support installation of an application. This is in contrast to "execute mode" described below in connection with FIGs. 4-16. In install mode, the application installation program is associated with an application isolation scope and is presumed to be executing on behalf of all users. The application isolation scope acts, for that application instance, as if it were the user isolation scope for "all users", and no user isolation scope is active for that application instance.

FIG. 3C depicts one embodiment of steps taken in install mode to identify a literal resource when a request to open a native resource is received that indicates the resource is being opened with the intention of modifying it. Briefly, as no user-isolation scope is active, a determination is first made whether the application-scoped instance of the requested native resource exists (step 374). If the application-scoped instance exists, it is identified as the literal resource

instance (step 384). If no application-scoped instance exists, a determination is made whether the system-scoped instance of the requested native resource exists (step 376). If it does not, an error condition is returned to the requestor indicating that the requested virtualized resource does not exist in the virtual scope (step 377). However, if the system-scoped resource exists, it is identified as the candidate resource instance (step 378), and permission data associated with the candidate instance is checked to determine if modification of that instance is allowed (step 380). If not, an error condition is returned to the requestor (step 381) indicating that modification of the virtualized resource is not allowed. If the permission data indicates that the candidate resource may be modified, as no user-isolation scope is active, an application-scoped copy of the candidate instance of the native resource is made (step 382), and the application-scoped instance is identified as the literal instance for the request (step 384). In some embodiments, the candidate file is copied to a location defined by the rules engine. For example, a rule may specify that the file is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file that do not appear in the isolation scope to which the file is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

FIG. 3D shows one embodiment of steps taken in install mode to identify the literal resource when a request to create a native resource is received. Briefly, as no user-isolation scope is active, a determination is first made whether the application-scoped instance of the requested native resource exists (step 390). If the application–scoped instance exists, an error condition may be returned to the requestor indicating that the resource cannot be created because it already exists (step 392). If no application-scoped instance exists, a determination may be made whether the system-scoped instance of the

requested native resource exists (step 394). If the system-scoped instance exists, an error condition may be returned to the requestor indicating that the resource cannot be created because it already exists (step 392). In some embodiments, the request used to open the resource may specify that any extant system-scoped instance of the resource may be overwritten. If the system-scoped resource instance does not exist, the application-scoped resource instance may be identified as the literal instance which will be created to fulfill the request (step 396).

By comparing FIGs. 3B with FIGs. 3C and 3D, it can be seen that install mode operates in a similar manner to execute mode, with the application isolation scope taking the place of the user isolation scope. In other words, modifications to persistent resources, including creation of new resources, take place in the appropriate application isolation scope instead of the appropriate user isolation scope. Furthermore, virtualization of access to existing isolated resources also ignores the appropriate user isolation scope and begins searching for a candidate literal resource in the application isolation scope.

There are two other cases where the application isolation scope operates in this manner to contain modifications to existing resources and creation of new resources. Firstly, there may be an isolation environment configured to operate without a user isolation layer, or a virtual scope configured to operate without a user isolation scope. In this case, the application isolation scope is the only isolation scope that can isolate modified and newly created resources. Secondly, the rules governing a particular set of virtual resources may specify that they are to be isolated into the appropriate application isolation scope rather than into the appropriate user isolation scope. Again, this means modifications to and creations of resources subject to that rule will be isolated into the appropriate application isolation scope where they are visible to all application instances sharing that scope, rather than in the user isolation scope where they are only visible to the user executing those application instances.

In still other embodiments, an isolation environment may be configured to allow certain resources to be shared in the system scope, that is, the isolation environment may act, for one or more system resources, as if no user isolation scope and no application isolation scope exists. System resources shared in the system scope are never copied when accessed with the intent to modify, because they are shared by all applications and all users, i.e., they are global objects.

4.0 Detailed Virtualization Examples

The methods and apparatus described above may be used to virtualize a wide variety of native resources 108. A number of these are described in detail below.

4.1 File System Virtualization

The methods and apparatus described above may be used to virtualize access to a file system. As described above, a file system is commonly organized in a logical hierarchy of directories, which are themselves files and which may contain other directories and data files.

4.1.1 File System Open Operations

In brief overview, FIG. 4 depicts one embodiment of the steps taken to open a file in the virtualized environment described above. A request to open a file is received or intercepted (step 402). The request contains a file name, which is treated as a virtual file name by the isolation environment. The processing rule applicable to the target of the file system open request is determined (step 404). If the rule action is "redirect" (step 406), the virtual file name provided in the request is mapped to a literal file name according to the applicable rule (step 408). A request to open the literal file using the literal file name is passed to the operating system and the result from the operating system is returned to the requestor (step 410). If instead the rule action is "ignore" (step 406), then the literal file name is determined to be exactly the virtual file name (step 412), and the request to open the literal file is passed to the operating system and the

result from the operating system is returned to the requestor (step 410). If in step 406 the rule action is "isolate", then the file name corresponding to the virtual file name in the user isolation scope is identified as the candidate file name (step 414). In other words, the candidate file name is formed by mapping the virtual file name to the corresponding native file name specific to the applicable user isolation scope. The category of existence of the candidate file is determined by examining the user isolation scope and any metadata associated with the candidate file (step 416). If the candidate file is determined to have "negative existence", because either the candidate file or one of its ancestor directories in the user isolation scope is marked as deleted, this means the requested virtual file is known to not exist. In this case, an error condition indicating the requested file is not found is returned to the requestor (step 422). If instead in step 416 the candidate file is determined to have "positive existence", because the candidate file exists in the user isolation scope and is not marked as a placeholder node, then the requested virtual file is known to exist. The candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420). If, however, in step 416, the candidate file has "neutral existence" because the candidate file does not exist, or the candidate file exists but is marked as a placeholder node, it is not yet known whether the virtual file exists or not. In this case the application-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 424). In other words, the candidate file name is formed by mapping the virtual file name to the corresponding native file name specific to the applicable application isolation scope. The category of existence of the candidate file is determined by examining the application isolation scope and any metadata associated with the candidate file (step 426). If the candidate file is determined to have "negative existence", because either the candidate file or one of its ancestor directories in the application isolation scope is marked as deleted, this means the requested virtual file is known to not exist. In this case, an error

condition indicating the requested file is not found is returned to the requestor (step 422). If instead in step 426 the candidate file is determined to have "positive existence", because the candidate file exists in the application isolation scope and is not marked as a placeholder node, then the requested virtual file is known to exist. The request is checked to determine if the open request indicates an intention to modify the file (step 428). If not, the candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420). If, however, in step 428, it is determined that the open request indicates intention to modify the file, permission data associated with the file is checked to determine if modification of the file is allowed (step 436). If not, an error condition is returned to the requestor (step 438) indicating that modification of the file is not allowed. If the permission data indicates that the file may be modified, the candidate file is copied to the user isolation scope (step 440). In some embodiments, the candidate file is copied to a location defined by the rules engine. For example, a rule may specify that the file is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file that do not appear in the isolation scope to which the file is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy. The scoped instance is identified as the literal file (step 442) and a request issued to open the literal file and the result returned to the requestor (step 420). Returning to step 426, if the candidate file has neutral existence because the candidate file does not exist, or because the candidate file is found but marked as a placeholder node, it is not yet known whether the virtual file exists or not. In this case, the system-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 430). In other words, the candidate file name is exactly the virtual file name. If the candidate file does not exist (step 432), an error condition indicating

the virtual file was not found is returned to the requestor (step 434). If on the other hand the candidate file exists (step 432), the request is checked to determine if the open request indicates an intention to modify the file (step 428). If not, the candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420). If, however, in step 428, it is determined that the open request indicates intention to modify the file, permission data associated with the file is checked to determine if modification of the file is allowed (step 436). If not, an error condition is returned to the requestor (step 438) indicating that modification of the file is not allowed. If the permission data indicates that the file may be modified, the candidate file is copied to the user isolation scope (step 440). In some embodiments, the candidate file is copied to a location defined by the rules engine. For example, a rule may specify that the file is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file that do not appear in the isolation scope are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy. The scoped instance is identified as the literal file (step 442) and a request issued to open the literal file and the result returned to the requestor (step 420).

This embodiment can be trivially modified to perform a check for existence of a file rather than opening a file. The attempt to open the literal file in step 420 is replaced with a check for the existence of that literal file and that status returned to the requestor.

Still referring to FIG. 4 and now in more detail, a request to open a virtual file is received or intercepted (step 402). The corresponding literal file may be of user isolation scope, application isolation scope or system scope, or it may be scoped to an application isolation sub-scope or a user isolation sub-scope. In some embodiments, the request is hooked by a function that replaces the

operating system function or functions for opening a file. In another embodiment a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native files. For embodiments in which a separate operating system function is provided for each type of file operation, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of file operations.

The request contains a file name, which is treated as a virtual file name by the isolation environment. The processing rule applicable to the file system open request is determined (step 404) by consulting the rules engine. In some embodiments the processing rule applicable to the open request is determined using the virtual name included in the open request. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual file name provided for the requested file is used as an index into the rule engine to locate one or more rules that apply to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular file and, in these embodiments, the rule having the longest prefix match with the virtual file name is the rule applied to the request. In other embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 4 as a single database transaction or single lookup into a file, the rule lookup may be performed as a series of rule lookups.

If the rule action is "redirect" (step 406), the virtual file name provided in the request is mapped to a literal file name according to the applicable rule (step 408). A request to open the literal file identified by the literal file name is passed to the operating system and the result from the operating system is returned to the requestor (step 410). For example, a request to open a file named "file_1" may result in the opening of a literal file named "Different_ file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to open the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the determined literal name. The I/O Manager then reissues the file open request with the determined literal name include in the STATUS_REPARSE response.

If instead the rule action is "ignore" (step 406), then the literal file name is determined to be exactly the virtual file name (step 412), and the request to open the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 410). For example, a request to open a file named "file_1" will result in the opening of an actual file named "file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument.

If in step 406 the rule action is "isolate", then the user-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 414). In other words, the candidate file name is formed by mapping the virtual file name to the corresponding native file name specific to the applicable user isolation scope. For example, a request to open a file named "file_1" may result in the opening of an actual file named "Isolated_file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For

embodiments using a file system filter driver, the first request to open the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the determined literal name. The I/O Manager then reissues the file open request with the determined literal name include in the REPARSE response.

In some embodiments, the literal name formed in order to isolate a requested system file may be based on the virtual file name received and a scope-specific identifier. The scope-specific identifier may be an identifier associated with an application isolation scope, a user isolation scope, a session isolation scope, an application isolation sub-scope, a user isolation sub-scope, or some combination of the above. The scope-specific identifier is used to "mangle" the virtual name received in the request.

In other embodiments, the user isolation scope or a sub-scope may be a directory under which all files that exist in the user isolation scope are stored. In some of these embodiments, the directory tree structure under the user isolation directory reflects the path of the requested resource. In other words, the literal file path is formed by mapping the virtual file path to the user isolation scope. For example, if the requested file is c:\temp\test.txt and the user isolation scope directory is d:\user1\app1\, then the path to the user-scoped literal file may be d:\user1\app1\c\temp\test.txt. In other embodiments, the path to the user-scoped literal may be defined in a native naming convention. For example, the path to the user-scoped literal file may be d:\user1\app1\device\harddisk1\temp\test.txt. In still other embodiments, the user-scoped files may all be stored in a single directory with names chosen to be unique and a database may be used to store the mapping between the requested file name and the name of the corresponding literal file stored in the directory. In still other embodiments, the contents of the literal files may be stored in a database. In still other embodiments, the native file system provides the facility for a single file to contain multiple independent named "streams", and

the contents of the user-scoped files are stored as additional streams of the associated files in the system scope. Alternatively, the literal files may be stored in a custom file system that may be designed to optimize disk usage or other criteria of interest.

The category of existence of the candidate file is determined by examining the user isolation scope and any metadata associated with the candidate file (step 416). If the candidate file is determined to have "negative existence", because either the candidate file or one of its ancestor directories in the user isolation scope is marked as deleted, this means the requested virtual file is known to not exist. In this case, an error condition indicating the requested file is not found is returned to the requestor (step 422).

In some embodiments, small amounts of metadata about a file may be stored directly in the literal filename, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the file by virtual filename check for possible variations of the literal filename due to the presence of a metadata indicator, and requests to retrieve the name of the file itself are hooked or intercepted in order to respond with the literal name. In other embodiments, one or more alternate names for the file may be formed from the virtual file name and a metadata indicator, and may be created using hard link or soft link facilities provided by the file system. The existence of these links may be hidden from applications by the isolation environment by indicating that the file is not found if a request is given to access a file using the name of a link. A particular link's presence or absence may indicate one bit of metadata for each metadata indicator, or there may be a link with a metadata indicator that can take on multiple states to indicate several bits of metadata. In still other embodiments, where the file system supports alternate file streams, an alternate file stream may be created to embody metadata, with the size of the stream indicating several

bits of metadata. In still other embodiments, a file system may directly provide the ability to store some 3rd party metadata for each file in the file system.

In specific ones of these embodiments, a list of deleted files or file system elements may be maintained and consulted to optimize this check for deleted files. In these embodiments, if a deleted file is recreated then the file name may be removed from the list of deleted files. In others of these embodiments, a file name may be removed from the list if the list grows beyond a certain size.

If instead in step 416 the candidate file is determined to have "positive existence", because the candidate file exists in the user isolation scope and is not marked as a placeholder node, then the requested virtual file is known to exist. The candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420).

If, however, in step 416, the candidate file has "neutral existence" because the candidate file does not exist, or the candidate file exists but is marked as a placeholder node, it is not yet known whether the virtual file exists or not. In this case the application-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 424). In other words, the candidate file name is formed by mapping the virtual file name to the corresponding native file name specific to the applicable application isolation scope. The category of existence of the candidate file is determined by examining the application isolation scope and any metadata associated with the candidate file (step 426).

If the application-scoped candidate file is determined to have "negative existence", because either the candidate file or one of its ancestor directories in the application isolation scope is marked as deleted, this means the requested virtual file is known to not exist. In this case, an error condition indicating the requested file is not found is returned to the requestor (step 422).

If in step 426 the candidate file is determined to have "positive existence", because the candidate file exists in the application isolation scope and is not

marked as a placeholder node, then the requested virtual file is known to exist. The request is checked to determine if the open request indicates an intention to modify the file (step 428). If not, the candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420).

If, however, in step 428, it is determined that the open request indicates intention to modify the file, permission data associated with the file is checked to determine if modification of the file is allowed (step 436). In some embodiments, the permission data is associated with the application-scoped candidate file. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate file. In other embodiments, the permission data associated with the candidate file is provided by the operating system. Further, the rules engine may include configuration settings instructing the isolation environment to obey or override the native permission data for virtualized copies of resources. In some embodiments, the rules may specify for some virtual resources the scope in which modifications are to occur, for example the system scope or the application isolation scope or a sub-scope, or the user isolation scope or a sub-scope. In some embodiments, the rules engine may specify configuration settings that apply to subsets of the virtualized native resources based on hierarchy or on type of resource accessed. In some of these embodiments, the configuration settings may be specific to each atomic native resource. In another example, the rules engine may include configuration data that prohibits or allows modification of certain classes of files, such as executable code or MIME types or file types as defined by the operating system.

If the permission data associated with the candidate file indicates that it may not be modified, an error condition is returned to the requestor (step 438) indicating that modification of the file is not allowed. If the permission data indicates that the file may be modified, the candidate file is copied to the user isolation scope (step 440). In some embodiments, the candidate file is copied to

a location defined by the rules engine. For example, a rule may specify that the file is copied to another application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file that do not appear in the isolation scope to which the file is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

In some embodiments, metadata is associated with files copied to the isolation scope that identifies the date and time at which the files were copied. This information may be used to compare the time stamp associated with the copied instance of the file to the time stamp of the last modification of the original instance of the file or of another instance of the file located in a lower isolation scope. In these embodiments, if the original instance of the file, or an instance of the file located in a lower isolation scope, is associated with a time stamp that is later than the time stamp of the copied file, that file may be copied to the isolation scope to update the candidate file. In other embodiments, the copy of the file in the isolation scope may be associated with metadata identifying the scope containing the original file that was copied.

In further embodiments, files that are copied to isolation scopes because they have been opened with intent to modify them may be monitored to determine if they are, in fact, modified. In one embodiment a copied file may be associated with a flag that is set when the file is actually modified. In these embodiments, if a copied file is not actually modified, it may be removed from the scope to which it was copied after it is closed, as well as any placeholder nodes associated with the copied file.

The scoped instance is identified as the literal file (step 442) and a request issued to open the literal file and the result returned to the requestor (step 420).

Returning to step 426, if the candidate file has neutral existence because the candidate file does not exist, or if the candidate file is found but marked as a

placeholder node, it is not yet known whether the virtual file exists or not. In this case, the system-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 430). In other words, the candidate file name is exactly the virtual file name.

If the candidate file does not exist (step 432), an error condition indicating the virtual file was not found is returned to the requestor (step 434). If on the other hand the candidate file exists (step 432), the request is checked to determine if the open request indicates an intention to modify the file (step 428).

As above, if the candidate file is being opened without the intent to modify it, the system-scoped candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420). If, however, in step 428, it is determined that the open request indicates intention to modify the file, permission data associated with the file is checked to determine if modification of the file is allowed (step 436). In some embodiments, the permission data is associated with the system-scoped candidate file. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate file. In other embodiments, the permission data associated with the candidate file is provided by the operating system.

If the permission data associated with the system-scoped candidate file indicates that the file may not be modified, an error condition is returned to the requestor (step 438) indicating that modification of the file is not allowed. If, however, the permission data indicates that the file may be modified, the candidate file is copied to the user isolation scope (step 440). In some embodiments, the candidate file is copied to a location defined by the rules engine. For example, a rule may specify that the file is copied to an application isolation scope or that it may be left in the system scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file

that do not appear in the isolation scope are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

In some embodiments, metadata is associated with files copied to the isolation scope that identifies the date and time at which the files were copied. This information may be used to compare the time stamp associated with the copied instance of the file to the time stamp of the last modification of the original instance of the file. In these embodiments, if the original instance of the file is associated with a time stamp that is later than the time stamp of the copied file, the original file may be copied to the isolation scope to update the candidate file. In other embodiments, the candidate file copied to the isolation scope may be associated with metadata identifying the scope from which the original file was copied.

In further embodiments, files that are copied to isolation scopes because they have been opened with intent to modify them may be monitored to determine if they are, in fact, modified. In one embodiment a copied file may be associated with a flag that is set when the file is actually modified. In these embodiments, if a copied file is not actually modified, when it is closed it may be removed from the scope to which it was copied, as well as any placeholder nodes associated with the copied file. In still further embodiments, the file is only copied to the appropriate isolation scope when the file is actually modified.

The scoped instance is identified as the literal file (step 442) and a request issued to open the literal file and the result returned to the requestor (step 420).

4.1.2 File System Delete Operations

Referring now to FIG. 5, and in brief overview, one embodiment of the steps taken to delete a file is depicted. A request to delete a file is received or intercepted (step 502). The request contains a file name, which is treated as a virtual file name by the isolation environment. A rule determines how the file operation is processed (step 504). If the rule action is "redirect" (step 506), the virtual file name is mapped directly to a literal file name according to the rule

(step 508). A request to delete the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 510). If the rule action is "ignore" (step 506), then the literal file name is identified as exactly the virtual file name (step 513), and a request to delete the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 510). If the rule action is "isolate" (step 506), then the existence of the virtual file is determined (step 514). If the virtual file does not exist, an error condition is returned to the requestor indicating that the virtual file does not exist (step 516). If the virtual file exists, and if the virtualized file specifies a directory rather than an ordinary file, the virtual directory is consulted to determine if it contains any virtual files or virtual subdirectories (step 518). If the requested virtualized file is a virtual directory that contains any virtual files or virtual subdirectories, the virtual directory cannot be deleted and an error message is returned (step 520). If the requested virtualized file is an ordinary file or is a virtual directory that contains no virtual files and no virtual subdirectories, then the literal file corresponding to the virtual file is identified (step 522). Permission data associated with the file is checked to determine if deletion is allowed (step 524). If not, a permission error message is returned (step 526). If, however, deletion of the file is allowed, and if the literal file is in the appropriate user isolation scope (step 528), the literal file is deleted (step 534) and a "deleted" node representing the deleted virtual file is created in the appropriate user isolation scope (step 536). If, however, in step 528 it is determined that the literal file is not in the user isolation scope but is in the appropriate application isolation scope or the system scope, then an instance of every user-scoped ancestor of the user-scoped instance of the requested file that does not already exist is created and marked as a placeholder (step 532). This is done to maintain the logical hierarchy of the directory structure in the user isolation scope. A user-scoped "deleted" node representing the deleted virtual file is then created in the appropriate user isolation scope (step 536).

Still referring to FIG. 5, and in more detail, a request to delete a file is received or intercepted (step 502). The file may be of user isolation scope, application isolation scope, system scope, or some applicable isolation sub-scope. In some embodiments, the request is hooked by a function that replaces the operating system function or functions for deleting the file. In another embodiment a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native files. For embodiments in which a separate operating system function is provided for each type of file, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of files.

The request contains a file name, which is treated as a virtual file name by the isolation environment. A processing rule applicable to the delete operation is determined (step 504) by consulting the rules engine. In some embodiments, the virtual file name provided for the requested file is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular file and, in these embodiments, the rule having the longest prefix match with the virtual file name is the rule applied to the request. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual file name provided in the request is used as an index into a rules engine to locate one or more rules that apply to the request. In other embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be

to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 5 as a series of decisions, the rule lookup may occur as a single database transaction.

If the rule action is "redirect" (step 506), the virtual file name is mapped directly to a literal file name according to the applicable rule (step 508). A request to delete the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 510). For example, a request to delete a file named "file_1" may result in the deletion of an actual file named "Different_ file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to delete the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the determined literal name. The I/O Manager then reissues the file delete request with the determined literal name include in the STATUS_REPARSE response.

In some embodiments, operating system permissions associated with the literal file "Different_file_1" may prevent deletion of the literal file. In these embodiments, an error message is returned that the file could not be deleted.

If the rule action is "ignore" (step 506), then the literal file name is identified as exactly the virtual file name (step 513), and a request to delete the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 510). For example, a request to delete a file named "file_1" will result in the deletion of an actual file named "file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to delete the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the

literal name. The I/O Manager then reissues the file delete request with the determined literal name include in the STATUS_REPARSE response.

In some embodiments, operating system permissions associated with the literal file "file_1" may prevent deletion of the literal file. In these embodiments, an error message is returned that the file could not be deleted.

If the rule action is "isolate" (step 506), then the existence of the virtual file is determined (step 514). If the file does not exist, an error is returned indicating that the file is not found (step 516).

If, however, in step 518 it is determined that the file exists but that it is not an ordinary file and is not an empty virtual directory, i.e., it contains virtual files or virtual subdirectories, an error message is returned indicating that the file may not be deleted (step 520).

If, however, the file is determined to exist and the requested virtualized file is an ordinary file or is an empty virtual directory, i.e., it contains no virtual files and no virtual subdirectories (step 518), then the literal file corresponding to the virtual file is identified (step 522). The literal file name is determined from the virtual file name as specified by the isolation rule. For example, a request to delete a file named "file_1" may result in the deletion of an actual file named "Isolated_file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to delete the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the literal name. The I/O Manager then reissues the file delete request with the determined literal name include in the STATUS_REPARSE response.

Once the literal file corresponding the virtual file is identified, it is determined whether the literal file may be deleted (step 524). If the file may not be deleted, an error is returned indicating that the file could not be deleted (step 524). In some embodiments, the permission data is associated with the system-

scoped candidate file. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate file. In other embodiments, the permission data associated with the candidate file is provided by the operating system.

If, however, deletion of the file is allowed, and if the literal file is in the appropriate user isolation scope (step 528), the literal file is deleted (step 534) and a "deleted" node representing the deleted virtual file is created in the appropriate user isolation scope (step 536).

If, however, in step 528 it is determined that the literal file is not in the user isolation scope but is in the appropriate application isolation scope or the system scope, then an instance of every user-scoped ancestor of the user-scoped instance of the requested file that does not already exist is created and marked as a placeholder (step 532). This is done to maintain the logical hierarchy of the directory structure in the user isolation scope. A user-scoped "deleted" node representing the deleted virtual file is then created in the appropriate user isolation scope (step 536). In some embodiments, the identity of the deleted file is stored in a file or other cache memory to optimize checks for deleted files.

In some embodiments, the located virtualized file may be associated with metadata indicating that the virtualized file has already been deleted. In some other embodiments, an ancestor of the virtualized file (e.g., a higher directory containing the file) is associated with metadata indicating that it is deleted. In these embodiments, an error message may be returned indicating that the virtualized file does not exist. In specific ones of these embodiments, a list of deleted files or file system elements may be maintained and consulted to optimize this check for deleted files.

4.1.3 File System Enumeration Operations

Referring now to FIG. 6, and in brief overview, one embodiment of the steps taken to enumerate a directory in the described virtualized environment is shown. A request to enumerate is received or intercepted (step 602). The

request contains a directory name that is treated as a virtual directory name by the isolation environment. Conceptually, the virtual directory's existence is determined as described in section 4.1.1 (step 603). If the virtual directory does not exist, a result indicating that the virtual directory is not found is returned to the requestor (step 620). If instead the virtual directory exists, the rules engine is consulted to determine the rule for the directory specified in the enumerate request (step 604). If the rule specifies an action of "redirect" (step 606), the literal directory name corresponding to the virtual directory name is determined as specified by the rule (step 608) and the literal directory identified by the literal name is enumerated, and the enumeration results stored in a working data store (step 612), followed by step 630 as described later. If the rule action specified is not "redirect" and is "ignore," (step 610) the literal directory name is exactly the virtual directory name (step 613) and the literal directory is enumerated, and the enumeration results stored in a working data store (step 612), followed by step 630 as described later. If, however, the rule action specifies "isolate," firstly the system scope is enumerated; that is, the candidate directory name is exactly the virtual directory name, and if the candidate directory exists it is enumerated. The enumeration results are stored in a working data store. If the candidate directory does not exist, the working data store remains empty at this stage (step 614). Next, the candidate directory is identified as the application-scoped instance of the virtual directory, and the category of existence of the candidate directory is determined (step 615). If the candidate directory has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 642). If instead the candidate directory does not have negative existence, the candidate directory is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each file system element in the enumeration, its category of existence is determined. Elements with negative existence are removed from the working data store, and elements with positive

existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding element if one is already present in the working data store (step 616).

In either case, the candidate directory is identified as the user-scoped instance of the virtual directory, and the category of existence of the candidate directory is determined (step 617). If the candidate directory has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 644). If instead the candidate directory does not have negative existence, the candidate directory is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each file system element in the enumeration, its category of existence is determined. Elements with negative existence are removed from the working data store, and elements with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding element if one is already present in the working data store (step 618), followed by step 630 as described below.

Then, for all three types of rules, step 630 is executed. The rules engine is queried to find the set of rules whose filters match immediate children of the requested directory, but do not match the requested directory itself (step 630). For each rule in the set, the existence of the virtual child whose name matches the name in the rule is queried using the logic outlined in section 4.1.1. If the child has positive existence, it is added to the working data store, replacing any child of the same name already there. If the child has negative existence, the entry in the working data store corresponding to the child, if any, is removed. (Step 632). Finally, the constructed enumeration is then returned from the working data store to the requestor (step 620).

Still referring to FIG. 6, and in more detail, a request to enumerate a directory is received or intercepted (step 602). In some embodiments, the request is hooked by a function that replaces the operating system function or functions for enumerating a directory. In another embodiment, a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for file operations. For embodiments in which a separate operating system function is provided for each type of file operation, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of file operations.

The existence of the virtual directory is determined (step 603). This is achieved as described in section 4.1.1. If the virtual directory does not exist, it cannot be enumerated, and a result indicating that the virtual directory does not exist is returned to the requestor (step 620).

The request contains a directory name, which is treated as a virtual directory name by the isolation environment. If the virtual directory exists, then a rule determining how the enumeration operation is to be processed is located (step 604) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual directory name provided for the requested directory is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular directory and, in these embodiments, the rule having the longest prefix match with the virtual directory name is the rule applied to the

request. In other embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 6 as a single database transaction or single lookup into a file, the rule lookup may be performed as a series of rule lookups.

If the rule action is "redirect" (step 606), the virtual directory name is mapped directly to a literal directory name according to the rule (step 608). A request to enumerate the literal directory is passed to the operating system (step 612) and step 630 is executed as described later. For example, a request to enumerate a directory named "directory _1" may result in the enumeration of a literal directory named "Different_ directory_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to open the directory for enumeration using the virtual name results in a "STATUS_REPARSE" request response indicating the determined literal name. The I/O Manager then reissues the directory open request for enumeration with the determined literal name include in the STATUS_REPARSE response.

If the rule action is not "redirect" (step 606), but is "ignore" (step 610), then the literal directory name is identified as exactly the virtual directory name (step 613), and a request to enumerate the literal directory is passed to the operating system (step 612) and step 630 is executed as described later. For example, a request to enumerate a directory named "directory _1" will result in the enumeration of an actual directory named "directory _1." In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to enumerate the directory using the virtual name is passed on unmodified by the filter driver.

If the rule action determined in step 610 is not "ignore" but is "isolate", then the system scope is enumerated, that is, the virtual name provided in the request is used to identify the enumerated directory (step 614). The results of the enumeration are stored in a working data store. In some embodiments, the working data store is comprised of a memory element. In other embodiments, the working data store comprises a database or a file or a solid-state memory element or a persistent data store.

Next, the candidate directory is identified as the application-scoped instance of the virtual directory, and the category of existence of the candidate directory is determined (step 615). If the candidate directory has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 642).

In some embodiments, small amounts of metadata about a file may be stored directly in the literal filename, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the file by virtual filename check for possible variations of the literal filename due to the presence of a metadata indicator, and requests to retrieve the name of the file itself are hooked or intercepted in order to respond with the literal name. In other embodiments, one or more alternate names for the file may be formed from the virtual file name and a metadata indicator, and may be created using hard link or soft link facilities provided by the file system. The existence of these links may be hidden from applications by the isolation environment by indicating that the file is not found if a request is given to access a file using the name of a link. A particular link's presence or absence may indicate one bit of metadata for each metadata indicator, or there may be a link with a metadata indicator that can take on multiple states to indicate several bits of metadata. In still other embodiments,

where the file system supports alternate file streams, an alternate file stream may be created to embody metadata, with the size of the stream indicating several bits of metadata. In still other embodiments, a file system may directly provide the ability to store some 3rd party metadata for each file in the file system. In yet other embodiment, a separate sub-scope may be used to record deleted files, and existence of a file (not marked as a placeholder) in that sub-scope is taken to mean that the file is deleted.

If instead the candidate directory does not have negative existence, the candidate directory is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each file system element in the enumeration, its category of existence is determined. Elements with negative existence are removed from the working data store, and elements with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding element if one is already present in the working data store (step 616).

In either case, the candidate directory is identified as the user-scoped instance of the virtual directory, and the category of existence of the candidate directory is determined (step 617). If the candidate directory has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 644). If instead the candidate directory does not have negative existence, the candidate directory is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each file system element in the enumeration, its category of existence is determined. Elements with negative existence are removed from the working data store, and elements with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data

store, replacing the corresponding element if one is already present in the working data store (step 618), followed by step 630 as described below.

Then, for all three types of rules, step 630 is executed. The rules engine is queried to find the set of rules whose filters match immediate children of the requested directory, but do not match the requested directory itself (step 630). For each rule in the set, the existence of the virtual child whose name matches the name in the rule is queried using the logic outlined in section 4.1.1. If the child has positive existence, it is added to the working data store, replacing any child of the same name already there. If the child has negative existence, the entry in the working data store corresponding to the child, if any, is removed. (Step 632). Finally, the constructed enumeration is then returned from the working data store to the requestor (step 620).

A practitioner of ordinary skill in the art will realize that the layered enumeration process described above can be applied with minor modification to the operation of enumerating a single isolation scope which comprises a plurality of isolation sub-scopes. A working data store is created, successive sub-scopes are enumerated and the results are merged into the working data store to form the aggregated enumeration of the isolation scope.

4.1.4. File System Creation Operations

Referring now to FIG. 7, and in brief overview, one embodiment of the steps taken to create a file in the isolation environment is shown. A request to create a file is received or intercepted (step 702). The request contains a file name, which is treated as a virtual file name by the isolation environment. An attempt is made to open the requested file using full virtualization using applicable rules, i.e. using appropriate user and application isolation scope, as described in section 4.1.1 (step 704). If access is denied (step 706), an access denied error is returned to the requestor (step 709). If access is granted (step 706), and the requested file is successfully opened (step 710), the requested file is returned to the requestor (step 712). However, if access is granted (step 706),

but the requested file is not opened successfully (step 710) then if the parent of the requested file also does not exist (step 714), an error appropriate to the request semantics is issued to the requestor (step 716). If on the other hand, the parent of the requested file is found in full virtualized view using the appropriate user and application scope (step 714), a rule then determines how the file operation is processed (step 718). If the rule action is "redirect" or "ignore" (step 720), the virtual file name is mapped directly to a literal file name according to the rule. Specifically, if the rule action is "ignore", the literal file name is identified as exactly the virtual file name. If, instead, the rule action is "redirect", the literal file name is determined from the virtual file name as specified by the rule. Then a request to create the literal file is passed to the operating system, and the result is returned to the requestor (step 724). If on the other hand, the rule action determined in step 720 is "isolate", then the literal file name is identified as the instance of the virtual file name in the user isolation scope. If the literal file already exists, but is associated with metadata indicating that it is a placeholder or that it is deleted, then the associated metadata is modified to remove those indications, and it is ensured that the file is empty. In either case, a request to open the literal file is passed to the operating system (step 726). If the literal file was opened successfully (step 728), the literal file is returned to the requestor (step 730). If on the other hand, in step 728, the requested file fails to open, placeholders for each ancestor of the literal file that does not currently exist in the user-isolation scope (step 732) and a request to create the literal file using the literal name is passed to the operating system and the result is returned to the requestor (step 734).

Still referring to FIG. 7, and in more detail, a request to create a file is received or intercepted (step 702). In some embodiments, the request is hooked by a function that replaces the operating system function or functions for creating the file. In another embodiment, a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in

kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for files. For embodiments in which a separate operating system function is provided for each type of file operation, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of file operations.

The request contains a file name, which is treated as a virtual file name by the isolation environment. The requestor attempts to open the requested file using full virtualization using applicable rules, i.e. using appropriate user and application isolation scope, as described in section 4.1.1 (step 704). If access is denied during the full virtualized open operation (step 706), an access denied error is returned to the requestor (step 709). If access is granted (step 706), and the requested virtual file is successfully opened (step 710), the corresponding literal file is returned to the requestor (step 712). However, if access is granted (step 706), but the requested file is not opened successfully (step 710) then the virtual file has been determined not to exist. If the virtual parent of the requested virtual file also does not exist, as determined by the procedures in section 4.1.1 (step 714), an error appropriate to the request semantics is issued to the requestor (step 716). If on the other hand, the virtual parent of the requested virtual file is found in full virtualized view using the appropriate user and application scope (step 714), then a rule that determines how the create operation is processed is located (step 718) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual file name provided for the requested file is used to locate in the rule engine a rule that

applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular file and, in some of these embodiments, the rule having the longest prefix match with the virtual file name is the rule applied to the request. In some embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 7 as a single database transaction or single lookup into a file, the rule lookup may be performed as a series of rule lookups..

If the rule action is "redirect" or "ignore" (step 720), the virtual file name is mapped directly to a literal file name according to the rule (step 724). If the rule action is "redirect" (step 720), the literal file name is determined from the virtual file name as specified by the rule (step 724). If the rule action is "ignore" (step 720), the literal file name is determined to be exactly the virtual file name (step 724). If the rule action is "ignore" or the rule action is "redirect", a request to create the literal file using the determined literal file name is passed to the operating system and the result from the operating system is returned to the requestor (step 724). For example, a request to create a virtual file named "file_1" may result in the creation of a literal file named "Different_ file_1." In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. (step 724). For embodiments using a file system filter driver, the first request to open the file using the virtual name results in a "STATUS_REPARSE" request response that indicates the determined literal name. The I/O Manager then reissues the file open request with the determined literal name include in the STATUS_REPARSE response.

If the rule action determined in step 720 is not "ignore" or "redirect" but is "isolate," then the literal file name is identified as the instance of the virtual file name in the user isolation scope. If the literal file already exists, but is

associated with metadata indicating that it is a placeholder or that it is deleted, then the associated metadata is modified to remove those indications, and it is ensured that the file is empty.

In some embodiments, small amounts of metadata about a file may be stored directly in the literal filename, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the file by virtual filename check for possible variations of the literal filename due to the presence of a metadata indicator, and requests to retrieve the name of the file itself are hooked or intercepted in order to respond with the literal name. In other embodiments, one or more alternate names for the file may be formed from the virtual file name and a metadata indicator, and may be created using hard link or soft link facilities provided by the file system. The existence of these links may be hidden from applications by the isolation environment by indicating that the file is not found if a request is given to access a file using the name of a link. A particular link's presence or absence may indicate one bit of metadata for each metadata indicator, or there may be a link with a metadata indicator that can take on multiple states to indicate several bits of metadata. In still other embodiments, where the file system supports alternate file streams, an alternate file stream may be created to embody metadata, with the size of the stream indicating several bits of metadata. In still other embodiments, a file system may directly provide the ability to store some 3rd party metadata for each file in the file system.

In specific ones of these embodiments, a list of deleted files or file system elements may be maintained and consulted to optimize this check for deleted files. In these embodiments, if a deleted file is recreated then the file name may be removed from the list of deleted files. In others of these embodiments, a file name may be removed from the list if the list grows beyond a certain size.

In either case, a request to open the user-scoped literal file is passed to the operating system (step 726). In some embodiments, rules may specify that the literal file corresponding to the virtual file should be created in a scope other than the user isolation scope, such as the application isolation scope, the system scope, a user isolation sub-scope or an application isolation sub-scope.

If the literal file was opened successfully (step 728), the literal file is returned to the requestor (step 730). If on the other hand, in step 728, the requested file fails to open, placeholders are created for each ancestor of the literal file that does not currently exist in the user-isolation scope (step 732) and a request to create the literal file using the literal name is passed to the operating system and the result is returned to the requestor (step 734).

This embodiment is for operating systems with APIs or facilities that only support creation of one level per call/invocation. Extension to multi-levels per call/invocation should be obvious to one skilled in the art.

4.1.5 Short filename management

In some file systems, both short and long filenames may be given to each file. Either name may be used to access the file in any of the file operations described above. For each file that possesses both a short and long filename, this implicitly creates an association between the short and long filename assigned to that file. In some of these file systems, short names are automatically assigned by the file system to files that are created using long file names. If the association between short and long filename is not maintained by the isolation environment, files with different long names in the same directory but in different scope levels may have the same short file name, leading to ambiguity if the short name is used to access a virtual file. Alternately, the short file name may change when a file is copied to a user isolation scope for modification meaning the virtual file can no longer be accessed using the original short name.

In order to prevent these issues, firstly file system operations that copy file instances opened with intention to modify to a "higher" scope preserve the association between the short and long filenames associated with the copied instance. Secondly, unique short names are created for newly-created isolated files in lieu of the filenames assigned by the operating system. The generated short filenames should satisfy the condition that the generated filenames do not match any existing short filenames in the same directory in the same isolation scope or in the same directory in a "lower" isolation scope. For example, a short filename generated for an instance of a file located in a user isolation scope should not match existing short filenames in application-scoped instance of the directory or in the system-scoped instance of the directory.

Referring now to FIG. 7A, one embodiment of the steps taken to assign unique short filenames after creating a new file is shown. In brief overview, a check is made to determine if short filenames should be generated (step 752). If not, a status is returned indicating that no short filename will be generated (step 754). Otherwise, the filename is checked to determine if it is already a legal short filename according to the file system (step 756). If it is already a legal short filename, a status is returned indicating that no short name will be generated (step 754). Otherwise, a suitable short filename is constructed (step 758).

Still referring to FIG. 7A, and in greater detail, a check is made to determine if short filenames should be generated (step 752). In some embodiments, this decision may be made based on the device storing the file to which the filename refers. In other embodiments, generation of short filenames may be enabled for certain scopes or sub-scopes, or for the isolation environment as a whole. In some of these embodiments, a registry setting may specify whether a short filename will be generated for a particular filename. If no short filename should be generated, a status that no short filename will be generated is returned (step 754).

Otherwise, the filename is checked to determine if it is already a legal short filename (step 756). In some embodiments, legal short filenames contain up to eight characters in the filename and up to three characters in an optional extension. In some embodiments, legal short names contain only legal characters, such as A-Z, a-z, 0-9, `, ~, !, @, #, $, %, ^, &, *, (,), -, _, ', {, and }. In some embodiments a leading space or "." or more than one embedded "." is illegal. If the provided filename is already a legal short filename, a status is returned that no short filename will be generated (step 754).

Otherwise, if it is determined in step 756 that the filename is an illegal short filename, a suitable short filename is constructed (step 758). In some embodiments this is achieved by using some of the parts of the long filename that are legal to use in a short filename, combined with an encoded iteration count to form a candidate short filename. The iteration count is increased until the associated candidate short filename is suitable, that is it is a legal short filename that is not used by any other file in the same directory in the same scope, or in the same directory in a lower scope. In other embodiments, the long filename is mangled or hashed and encoded, and is combined with an encoded iteration count to form a candidate short filename. The iteration count is increased until the associated candidate short filename is suitable, that is it is a legal short filename that is not used by any other file in the same directory in the same scope, or in the same directory in a lower scope. In all of these embodiments a scope-specific string may be incorporated into the candidate short filename to increase the likelihood that a suitable candidate short filename will be found with a low iteration count.

4.2 Registry Virtualization

The methods and apparatus described above may be used to virtualize access to a registry database. As described above a registry database stores information regarding hardware physically attached to the computer, which system options have been selected, how computer memory is set up, various

items of application-specific data, and what application programs should be present when the operating system is started. A registry database is commonly organized in a logical hierarchy of "keys" 170, 172, which are containers for registry values.

4.2.1 Registry Key Open Operations

In brief overview, FIG. 8 depicts one embodiment of the steps taken to open a registry key in the isolation environment described above. A request to open a registry key is received or intercepted, the request containing a registry key name which is treated as a virtual key name by the isolation environment (step 802). A processing rule applicable to the virtual name in the request determines how the registry key operation is processed (step 804). If the rule action is "redirect" (step 806), the virtual key name provided in the request is mapped to a literal key name as specified by the applicable rule (step 808). A request to open the literal registry key using the literal key name is passed to the operating system and the result from the operating system is returned to the requestor (step 810). If the rule action is not "redirect", but is "ignore" (step 806), then the virtual key name is identified as the literal key name (step 812), and a request to open the literal registry key is passed to the operating system and the result from the operating system is returned to the requestor (step 810). If the rule action determined in step 806 is not "redirect" and is not "ignore," but is "isolate", the virtual key name provided in the request is mapped to a user-scoped candidate key name, that is a key name corresponding to the virtual key name that is specific to the applicable user isolation scope (step 814). The category of existence of the user-scoped candidate key is determined by examining the user isolation scope and any metadata associated with the candidate key (step 816). If the candidate key is determined to have "negative existence", because either the candidate key or one of its ancestor keys in the user isolation scope is marked as deleted, this means the requested virtual key is known to not exist. In this case, an error condition indicating the requested file is

not found is returned to the requestor (step 822). If instead in step 816 the candidate key is determined to have "positive existence", because the candidate key exists in the user isolation scope and is not marked as a placeholder node, then the requested virtual key is known to exist. The candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820). If, however, in step 816, the candidate key has "neutral existence" because the candidate key does not exist, or the candidate key exists but is marked as a placeholder node, it is not yet known whether the virtual key exists or not. In this case the application-scoped key name corresponding to the virtual key name is identified as the candidate key name (step 824). In other words, the candidate key name is formed by mapping the virtual key name to the corresponding native key name specific to the applicable application isolation scope. The category of existence of the candidate key is determined by examining the application isolation scope and any metadata associated with the candidate key (step 826). If the candidate key is determined to have "negative existence", because either the candidate key or one of its ancestor keys in the application isolation scope is marked as deleted, this means the requested virtual key is known to not exist. In this case, an error condition indicating the requested key is not found is returned to the requestor (step 822). If instead in step 826 the candidate key is determined to have "positive existence", because the candidate key exists in the application isolation scope and is not marked as a placeholder node, then the requested virtual key is known to exist. The request is checked to determine if the open request indicates an intention to modify the key (step 828). If not, the candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820). If, however, in step 828, it is determined that the open request indicates an intention to modify the key, permission data associated with the key is checked to determine if modification of the key is allowed (step 836). If not, an error

condition is returned to the requestor (step 838) indicating that modification of the key is not allowed. If the permission data indicates that the key may be modified, the candidate key is copied to the user isolation scope (step 840). In some embodiments, the candidate key is copied to a location defined by the rules engine. For example, a rule may specify that the key is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the key should be copied. Any ancestors of the requested key that do not appear in the isolation scope to which the key is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy. The newly copied scoped instance is identified as the literal key (step 842) and a request issued to open the literal key and the result returned to the requestor (step 820). Returning to step 826, if the candidate key has neutral existence because the candidate key does not exist, or because the candidate key is found but marked as a placeholder node, it is not yet known whether the virtual key exists or not. In this case, the system-scoped key name corresponding to the virtual key name is identified as the candidate key name (step 830). In other words, the candidate key name is exactly the virtual key name. If the candidate key does not exist (step 832), an error condition indicating the virtual key was not found is returned to the requestor (step 834). If on the other hand the candidate key exists (step 832), the request is checked to determine if the open request indicates an intention to modify the key (step 828). If not, the candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820). If, however, in step 828, it is determined that the open request indicates intention to modify the key, permission data associated with the key is checked to determine if modification of the key is allowed (step 836). If not, an error condition is returned to the requestor (step 838) indicating that modification of the key is not allowed. If the permission data indicates that the key may be modified, the

candidate key is copied to the user isolation scope (step 840). In some embodiments, the candidate key is copied to a location defined by the rules engine. For example, a rule may specify that the key is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the key should be copied. Any ancestors of the requested key that do not appear in the isolation scope are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy. The newly copied scoped instance is identified as the literal key (step 842) and a request issued to open the literal key and the result returned to the requestor (step 820).

Still referring to FIG. 8 and now in more detail, a request to open a virtual registry key is received or intercepted (step 802). The corresponding literal registry key may be of user isolation scope, application isolation scope or system scope, or it may be scoped to an application isolation sub-scope or a user isolation sub-scope. In some embodiments, the request is hooked by a function that replaces the operating system function or functions for opening a registry key. In another embodiment a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native registry keys. For embodiments in which a separate operating system function is provided for each type of registry key operation, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of registry key operations.

The request contains a registry key name, which is treated as a virtual registry key name by the isolation environment. The processing rule applicable

to the registry key open request is determined (step 804) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual registry key name provided for the requested registry key is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular registry key and, in these embodiments, the rule having the longest prefix match with the virtual registry key name is the rule applied to the request. In other embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 8 as a single database transaction or single lookup into a file, the rule lookup may be performed as a series of rule lookups .

If the rule action is "redirect" (step 806), the virtual registry key name provided in the request is mapped to the literal registry key name according to the applicable rule (step 808). A request to open the literal registry key using the literal registry key name is passed to the operating system and the result from the operating system is returned to the requestor (step 810). For example, a request to open a registry key named "registry_key_1" may result in the opening of a literal registry key named "Different_ registry_key _1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. In these embodiments, opening the literal registry key may be achieved by responding to the original request to open the virtual key by signaling to the registry filter manager to reparse the request using the determined literal key name.. If instead the rule action is "ignore" (step 806), then the literal registry key name is

determined to be exactly the virtual registry key name (step 812), and the request to open the literal registry key is passed to the operating system and the result from the operating system is returned to the requestor (step 810). For example, a request to open a registry key named "registry_key_1" will result in the opening of a literal registry key named "registry_key _1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. In another embodiment, this is accomplished by signaling to the registry filter manager to continue processing the original unmodified request in the normal fashion.

If in step 806 the rule action is "isolate", then the user-scoped registry key name corresponding to the virtual registry key name is identified as the candidate registry key name (step 814). In other words, the candidate registry key name is formed by mapping the virtual registry key name to the corresponding native registry key name specific to the applicable user isolation scope. For example, a request to open a registry key named "registry_key_1" may result in the opening of a literal registry key named "Isolated_UserScope_UserA_registry_key_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. In other embodiments, opening the literal registry key may be achieved by responding to the original request to open the virtual key by signaling to the registry filter manager to reparse the request using the determined literal key name.

In some embodiments, the literal name formed in order to isolate a requested virtual registry key may be based on the virtual registry key name received and a scope-specific identifier. The scope-specific identifier may be an identifier associated with an application isolation scope, a user isolation scope, a session isolation scope, an application isolation sub-scope, a user isolation sub-scope, or some combination of the above. The scope-specific identifier is used to "mangle" the virtual name received in the request.

In other embodiments, the user isolation scope or a sub-scope may be a registry key under which all keys that exist in the user isolation scope are stored. In some of these embodiments, the key hierarchy under the user isolation key reflects the path of the requested resource. In other words, the literal key path is formed by mapping the virtual key path to the user isolation scope. For example, if the requested key is HKLM\Software\Citrix\MyKey and the user isolation scope key is HKCU\Software\UserScope\, then the path to the user-scoped literal key may be HKCU\Software\UserScope\HKLM\Software\Citrix\MyKey. In other embodiments, the path to the user-scoped literal may be defined in a native naming convention. For example, the path to the user-scoped literal key may be HKCU\Software\UserScope\Registry\Machine\Software\Citrix\MyKey. In still other embodiments, the user-scoped keys may all be stored under a single key with names chosen to be unique and a database may be used to store the mapping between the requested key name and the name of the corresponding literal key stored in the user isolation key. In still other embodiments, the contents of the literal keys may be stored in a database or a file store.

The category of existence of the candidate key is determined by examining the user isolation scope and any metadata associated with the candidate key (step 816). If the candidate key is determined to have "negative existence", because either the candidate key or one of its ancestor keys in the user isolation scope is marked as deleted, this means the requested virtual key is known to not exist. In this case, an error condition indicating the requested key is not found is returned to the requestor (step 822).

In some embodiments, the literal registry key may be associated with metadata indicating that the virtualized registry key has already been deleted. In some embodiments, metadata about a registry key may be stored in a distinguished value held by that key, with the existence of that value hidden from ordinary application usage of registry APIs. In some embodiments, small amounts of metadata about a registry key may be stored directly in the literal key

name, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the key by virtual name check for possible variations of the literal key name due to the presence of a metadata indicator, and requests to retrieve the name of the key itself are hooked or intercepted in order to respond with the literal name. In other embodiments, the metadata indicator may be encoded in a subkey name or a registry value name instead of the key name itself. In still other embodiments, a registry key system may directly provide the ability to store some 3rd party metadata for each key. In some embodiments, metadata is stored in a database or other repository separate from the registry database. In some embodiments, a separate sub-scope may be used to store keys that are marked as deleted. The existence of a key in the sub-scope indicates that the key is marked as deleted.

In specific ones of these embodiments, a list of deleted keys or key system elements may be maintained and consulted to optimize this check for deleted keys. In these embodiments, if a deleted key is recreated then the key name may be removed from the list of deleted keys. In others of these embodiments, a key name may be removed from the list if the list grows beyond a certain size.

If instead in step 816 the candidate key is determined to have "positive existence", because the candidate key exists in the user isolation scope and is not marked as a placeholder node, then the requested virtual key is known to exist. The candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820).

If, however, in step 816, the candidate key has "neutral existence" because the candidate key does not exist, or the candidate key exists but is marked as a placeholder node, it is not yet known whether the virtual key exists

or not. In this case the application-scoped key name corresponding to the virtual key name is identified as the candidate key name (step 824). In other words, the candidate key name is formed by mapping the virtual key name to the corresponding native key name specific to the applicable application isolation scope. The category of existence of the candidate key is determined by examining the application isolation scope and any metadata associated with the candidate key (step 826).

If the application-scoped candidate key is determined to have "negative existence", because either the candidate key or one of its ancestor keys in the application isolation scope is marked as deleted, this means the requested virtual key is known to not exist. In this case, an error condition indicating the requested key is not found is returned to the requestor (step 822).

If, however, in step 826 the candidate key is determined to have "positive existence", because the candidate key exists in the application isolation scope and is not marked as a placeholder node, then the requested virtual key is known to exist. The request is checked to determine if the open request indicates an intention to modify the key (step 828). If not, the candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820).

If, however, in step 828, it is determined that the open request indicates intention to modify the key, permission data associated with the key is checked to determine if modification of the key is allowed (step 836). In some embodiments, the permission data is associated with the application-scoped candidate key. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate key. In other embodiments, the permission data associated with the candidate key is provided by the operating system. Further, the rules engine may include configuration settings instructing the isolation environment to obey or override the native permission data for virtualized copies of resources. In some embodiments, the rules may specify for

some virtual resources the scope in which modifications are to occur, for example the system scope or the application isolation scope or a sub-scope, or the user isolation scope or a sub-scope. In some embodiments, the rules engine may specify configuration settings that apply to subsets of the virtualized native resources based on hierarchy. In some of these embodiments, the configuration settings may be specific to each atomic native resource.

If the permission data associated with the candidate key indicates that it may not be modified, an error condition is returned to the requestor (step 838) indicating that modification of the key is not allowed. If the permission data indicates that the key may be modified, the candidate key is copied to the user isolation scope (step 840). In some embodiments, the candidate key is copied to a location defined by the rules engine. For example, a rule may specify that the key is copied to another application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the key should be copied. Any ancestors of the requested key that do not appear in the isolation scope to which the key is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

In some embodiments, metadata is associated with keys copied to the isolation scope that identifies the date and time at which the keys were copied. This information may be used to compare the time stamp associated with the copied instance of the key to the time stamp of the last modification of the original instance of the key or of another instance of the key located in a lower isolation scope. In these embodiments, if the original instance of the key, or an instance of the key located in a lower isolation scope, is associated with a time stamp that is later than the time stamp of the copied key, that key may be copied to the isolation scope to update the candidate key. In other embodiments, the copy of the key in the isolation scope may be associated with metadata identifying the scope containing the original key that was copied.

In further embodiments, keys that are copied to isolation scopes because they have been opened with intent to modify them may be monitored to determine if they are, in fact, modified. In one embodiment a copied key may be associated with a flag that is set when the key is actually modified. In these embodiments, if a copied key is not actually modified, it may be removed from the scope to which it was copied after it is closed, as well as any placeholder nodes associated with the copied key.

The scoped instance is identified as the literal key (step 842) and a request issued to open the literal key and the result returned to the requestor (step 820).

Returning to step 826, if the candidate key has neutral existence because the candidate key does not exist, or if the candidate key is found but marked as a placeholder node, it is not yet known whether the virtual key exists or not. In this case, the system-scoped key name corresponding to the virtual key name is identified as the candidate key name (step 830). In other words, the candidate key name is exactly the virtual key name.

If the candidate key does not exist (step 832), an error condition indicating the virtual key was not found is returned to the requestor (step 834). If on the other hand the candidate key exists (step 832), the request is checked to determine if the open request indicates an intention to modify the key (step 828).

As above, if the candidate key is being opened without the intent to modify it, the system-scoped candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820). If, however, in step 828, it is determined that the open request indicates intention to modify the key, permission data associated with the key is checked to determine if modification of the key is allowed (step 836). In some embodiments, the permission data is associated with the application-scoped candidate key. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate key. In

other embodiments, the permission data associated with the candidate key is provided by the operating system. Further, the rules engine may include configuration settings instructing the isolation environment to obey or override the native permission data for virtualized copies of resources. In some embodiments, the rules may specify for some virtual resources the scope in which modifications are to occur, for example the system scope or the application isolation scope or a sub-scope, or the user isolation scope or a sub-scope. In some embodiments, the rules engine may specify configuration settings that apply to subsets of the virtualized native resources based on hierarchy. In some of these embodiments, the configuration settings may be specific to each atomic native resource.

If the permission data associated with the system-scoped candidate key indicates that the key may not be modified, an error condition is returned to the requestor (step 838) indicating that modification of the key is not allowed. If, however, the permission data indicates that the key may be modified, the candidate key is copied to the user isolation scope (step 840). In some embodiments, the candidate key is copied to a location defined by the rules engine. For example, a rule may specify that the key is copied to an application isolation scope or that it may be left in the system scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the key should be copied. Any ancestors of the requested key that do not appear in the isolation scope are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

In some embodiments, metadata is associated with keys copied to the isolation scope that identifies the date and time at which the keys were copied. This information may be used to compare the time stamp associated with the copied instance of the key to the time stamp of the last modification of the original instance of the key. In these embodiments, if the original instance of the key is associated with a time stamp that is later than the time stamp of the copied

key, the original key may be copied to the isolation scope to update the candidate key. In other embodiments, the candidate key copied to the isolation scope may be associated with metadata identifying the scope from which the original key was copied.

In further embodiments, keys that are copied to isolation scopes because they have been opened with intent to modify them may be monitored to determine if they are, in fact, modified. In one embodiment a copied key may be associated with a flag that is set when the key is actually modified. In these embodiments, if a copied key is not actually modified, when it is closed it may be removed from the scope to which it was copied, as well as any placeholder nodes associated with the copied key. In still further embodiments, the key is only copied to the appropriate isolation scope when the key is actually modified.

The scoped instance is identified as the literal key (step 842) and a request issued to open the literal key and the result returned to the requestor (step 820).

4.2.2 Registry Key Delete Operations

Referring now to FIG. 9, and in brief overview, one embodiment of the steps taken to delete a registry key is depicted. Before a key can be deleted, the key must first be opened successfully with delete access (step 901). If the key is not opened successfully, an error is returned (step 916). If the virtual key is opened successfully, a request to delete a virtualized registry key is received or intercepted, the request including the handle to the literal key corresponding to the virtual key (step 902). A rule determines how the registry key operation is processed (step 904). In addition to the rule applicable to the key to be deleted, any other rules applicable to immediate subkeys are examined (step 905). For each rule applicable to an immediate subkey found, an attempt is made to open a virtual subkey, with the virtual subkey's name being specified by the name given in the rule found in step 905. If a subkey with a name corresponding to one of the rules found in step 905 is opened successfully (step 906), then the virtual

key is considered to have subkeys, which means it cannot be deleted, and an error returned (step 907).

If, after all the virtual key names extracted in step 905 have been attempted to be opened (step 906), no virtual keys were found to exist, further examination is required. If the rule action is not "isolate", but is "redirect", or is "ignore" (step 908),, a request to delete the literal registry key is passed to the operating system and the result from the operating system is returned to the requestor (step 911). If however the rule action determined in step 908 is "isolate" the aggregated virtualized registry key is consulted to determine if it contains any virtual subkeys (step 914). If the virtualized key has virtual subkeys, then the deletion cannot continue, and an error is returned indicating the key has not been deleted (step 920). If the virtualized key does not have virtual subkeys, then the literal key corresponding to the virtual key is examined to determine if it masks a scoped key with the same virtual name in another scope level (step 922). If the literal key corresponding to the virtual key does not mask a differently scoped key with the same virtual name, then the literal key which corresponds to the virtual key is deleted, and the result returned (step 926). If the literal key corresponding to the virtual key masks a differently scoped key with the same virtual name, then the literal key corresponding to the virtual key is marked with a value indicating that it is deleted, and a successful result returned to the caller (step 924).

Still referring to FIG. 9, and in more detail, in order to delete a key, it must first be opened with delete access (step 901). The request to open the key with delete access includes the name of the key which is treated as a virtual name by the isolation environment. A full virtualized key open is performed as described in section 4.2.1. If the virtualized open operation fails, an error is returned to the requestor (step 916). If the virtualized open operation succeeds, the handle of the literal key corresponding to the virtual key is returned to the requestor. Subsequently a request to delete the registry key which was opened in step 901

is received or intercepted (step 902). The opened literal registry key may be of user isolation scope, application isolation scope, system scope, or some applicable isolation sub-scope. In some embodiments, the delete request is hooked by a function that replaces the operating system function or functions for deleting the registry key. In another embodiment a hooking dynamically-linked library is used to intercept the delete request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native registry keys. In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. A practitioner skilled in the art may create a registry filter driver to which the operating system passes requests to perform registry operations, thus providing a mechanism to intercept registry operation requests. For embodiments in which a separate operating system function is provided for each type of registry key function, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of registry key functions.

The delete request contains a literal key handle. The virtual key name associated with the handle is determined by querying the operating system for the literal name associated with the handle. The rules engine is consulted to determine the virtual name associated with the literal name, if any. A rule determining how the registry key operation is processed (step 904) is obtained by consulting the rules engine. In some embodiments, the virtual key name of the virtual registry key to be deleted is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular virtual registry key and, in some of

these embodiments, the rule having the longest prefix match with the virtual key name is the rule applied to the request. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat registry key database. In some embodiments, the virtual key name corresponding to the virtual key handle in the request is used as an index into a rules engine to locate one or more rules that apply to the request. In some embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. The rule lookup may occur as a series of decisions, or the rule lookup may occur as a single database transaction.

The virtual name of the key to be deleted is used to consult the rules engine to locate the set of rules applicable to any immediate child keys of the virtual key to delete, but not applicable to the virtual key to be deleted. This set of rules is located whether those child keys exist or not (step 905). If this set of rules applicable to immediate child keys is not empty, then the virtual name of each of these rules is extracted. An attempt is made to do a full virtualized open of each of the virtual child key names extracted, in turn (step 906). If any of the virtual keys corresponding to any of these virtual names can be opened successfully, then this means that a virtual subkey exists. This means that the virtual key cannot be deleted, as it has a virtual child that exists, and an error is returned (step 907). If after examining all of the set of rules applicable to immediate children of the virtual key (step 905), no virtual subkeys are found to exist, the deletion can continue. For example, a key with virtual name “key_1” may have child rules applicable to “key1\subkey_1” and “key1\subkey_2”. In this step, an attempt is made to do a virtualized open of “key1\subkey_1” and “key1\subkey_2”. If either of these virtual subkeys can be opened successfully,

then the deletion will fail, and an error is returned (step 907). Only if neither of these virtual subkeys exist can the deletion continue.

If the rule action is not "isolate", but is "redirect", or is "ignore" (step 908), a request to delete the literal registry key using the literal key handle is passed to the operating system and the result from the operating system is returned to the requestor (step 911). This request will fail if the literal key contains literal subkeys. In one embodiment, the request to delete the literal registry key is accomplished by calling the original version of the hooked function and passing the literal key handle to the function as an argument. In embodiments that make use of a registry filter driver, this is accomplished by responding to the request with a completion status that signals the operating system to perform normal processing on the request. In some embodiments, operating system permissions associated with the literal registry key may prevent its deletion. In these embodiments, an error message is returned that the virtual registry key could not be deleted.

If the rule action determined in step 908 is "isolate", then the aggregated virtualized registry key is consulted to determine if it contains any virtual subkeys (step 914). If the requested virtual registry key contains virtual subkeys, then the virtual key cannot be deleted, and an error is returned to the caller (step 920).

If the requested virtual registry key does not contain virtual subkeys, then the virtual key can be deleted. The action taken next depends on the scope that contains the literal key to be deleted. For example, a request to delete a virtual registry key may result in the deletion of an application-scoped literal key. The scope containing the literal key can be determined by consulting the rules engine with the full path to the literal key.

If the literal key to be deleted is found in a particular scope, and that literal key masks another key of the same virtual name in another scope, then the literal key to be deleted is marked as deleted, and a result returned to the requestor (step 924). For example, a virtual key that corresponds to a user-scoped literal

key is considered to mask a differently-scoped key if a corresponding application-scoped key with the same virtual name or a corresponding system-scoped key with the same virtual name has "positive existence", that is, exists in the scope, and is not marked as a placeholder, and is not considered to be deleted. Similarly, an application-scoped key is considered to mask a system-scoped key corresponding to the same virtual name if that system-scoped key exists and is not considered to be deleted.

If the literal key to be deleted is found not to mask another key of the same virtual name in another scope, then the literal key to be deleted is actually deleted and a result returned (step 926).

In some embodiments, operating system permissions associated with the literal registry key may prevent deletion of the literal registry key. In these embodiments, an error message is returned that the virtual registry key could not be deleted.

In some embodiments, the literal registry key may be associated with metadata indicating that the virtualized registry key has already been deleted. In some embodiments, metadata about a registry key may be stored in a distinguished value held by that key, with the existence of that value hidden from ordinary application usage of registry APIs. In some embodiments, small amounts of metadata about a registry key may be stored directly in the literal key name, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the key by virtual name check for possible variations of the literal key name due to the presence of a metadata indicator, and requests to retrieve the name of the key itself are hooked or intercepted in order to respond with the literal name. In other embodiments, the metadata indicator may be encoded in a subkey name or a registry value name instead of the key name itself. In still other embodiments, a registry key system may

directly provide the ability to store some 3rd party metadata for each key. In some embodiments, metadata could be stored in a database or other repository separate from the registry database. In some embodiments, a separate sub-scope may be used to store keys that are marked as deleted. The existence of a key in the sub-scope indicates that the key is marked as deleted.

In specific ones of these embodiments, a list of deleted keys or key system elements may be maintained and consulted to optimize this check for deleted keys. In these embodiments, if a deleted key is recreated then the key name may be removed from the list of deleted keys. In others of these embodiments, a key name may be removed from the list if the list grows beyond a certain size.

In some embodiments, an ancestor of the literal registry key in the same scope is associated with metadata indicating that it is deleted, or is otherwise indicated to be deleted. In these embodiments, an error message may be returned indicating that the virtualized registry key does not exist. In specific ones of these embodiments, a list of deleted registry keys or registry key system elements may be maintained and consulted to optimize this check for deleted registry keys.

4.2.3 Registry Key Enumeration Operations

Referring now to FIG. 10, and in brief overview, one embodiment of the steps taken to enumerate a key in the described virtualized environment is shown. Before a key can be enumerated, the key must first be opened successfully with enumerate access (step 1001). If the key is not opened successfully, an error is returned (step 1040). If the virtual key is opened successfully, a request to enumerate is received or intercepted, the request including the handle to the literal key corresponding to the virtual key (step 1002). The virtual key name corresponding to the handle is determined, and the rules engine is consulted to determine the rule for the key specified in the enumerate request (step 1004). If the rule doesn't specify an action of "isolate", but instead

specifies "ignore" or specifies "redirect" (step 1006), the literal key identified by the literal key handle is enumerated, and the enumeration results stored in a working data store (step 1012), followed by step 1030 as described later.

If, however, the rule action specifies "isolate," firstly the system scope is enumerated; that is, the candidate key name is exactly the virtual key name, and if the candidate key exists it is enumerated. The enumeration results are stored in a working data store. If the candidate key does not exist, the working data store remains empty at this stage (step 1014). Next, the candidate key is identified as the application-scoped instance of the virtual key, and the category of existence of the candidate key is determined (step 1015). If the candidate key has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 1042). If instead the candidate key does not have negative existence, the candidate key is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each subkey in the enumeration, its category of existence is determined. Subkeys with negative existence are removed from the working data store, and subkeys with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding subkey if one is already present in the working data store (step 1016).

In either case, the candidate key is identified as the user-scoped instance of the virtual key, and the category of existence of the candidate key is determined (step 1017). If the candidate key has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 1044). If instead the candidate key does not have negative existence, the candidate key is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each subkey in the enumeration, its

category of existence is determined. Subkeys with negative existence are removed from the working data store, and subkeys with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding subkey if one is already present in the working data store (step 1018), followed by step 1030 as described below.

Then, for all three types of rules, step 1030 is executed. The rules engine is queried to find the set of rules whose filters match immediate children of the requested virtual key name, but do not match the requested virtual key name itself (step 1030). For each rule in the set, the existence of the virtual child whose name matches the name in the rule is determined. If the child has positive existence, it is added to the working data store, replacing any child of the same name already there. If the child has negative existence, the entry in the working data store corresponding to the child, if any, is removed. (Step 1032). Finally, the constructed enumeration is then returned from the working data store to the requestor (step 1020).

Still referring to FIG. 10, and in more detail, in order to enumerate a key, it must first be opened with enumerate access (step 1001). The request to open the key with enumerate access includes the name of the key which is treated as a virtual name by the isolation environment. A full virtualized key open is performed as described in section 4.2.1. If the virtualized open operation fails, an error is returned to the requestor (step 1040). If the virtualized open operation succeeds, the handle of the literal key corresponding to the virtual key is returned to the requestor. Subsequently a request to enumerate the registry key which was opened in step 1001 is received or intercepted (step 1002). The opened literal registry key may be of user isolation scope, application isolation scope, system scope, or some applicable isolation sub-scope. In some embodiments, the enumerate request is hooked by a function that replaces the operating system function or functions for enumerating a registry key. In another

embodiment a hooking dynamically-linked library is used to intercept the enumerate request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native registry keys. In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. A practitioner skilled in the art may create a registry filter driver to which the operating system passes requests to perform registry operations, thus providing a mechanism to intercept registry operation requests. For embodiments in which a separate operating system function is provided for each type of registry key function, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of registry key functions.

The enumerate request contains a literal key handle. The virtual key name associated with the handle is determined by querying the operating system for the literal name associated with the handle. The rules engine is consulted to determine the virtual name associated with the literal name, if any.

A rule determining how the registry key operation is processed (step 1004) is obtained by consulting the rules engine. In some embodiments, the virtual key name of the virtual registry key to be enumerated is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular virtual registry key and, in some of these embodiments, the rule having the longest prefix match with the virtual key name is the rule applied to the request. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash

table, or a flat registry key database. In some embodiments, the virtual key name corresponding to the virtual key handle in the request is used as an index into a rules engine to locate one or more rules that apply to the request. In some embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. The rule lookup may occur as a series of decisions, or the rule lookup may occur as a single database transaction.

If the rule action is not "isolate" (step 1006), but is "ignore" or is "redirect", then a request to enumerate the literal key is passed to the operating system using the literal key handle, and the enumeration results, if any, are stored in the working data store (step 1012), and step 1030 is executed as described later. In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. In these embodiments, enumerating the literal registry key may be achieved by responding to the original request to enumerate the key by signaling to the registry filter manager to process the unmodified request in the normal fashion.

If the rule action determined in step 1010 is "isolate", then the system scope is enumerated. To achieve this, the candidate key is identified as the system-scoped key corresponding to the virtual key to be enumerated. The candidate key is enumerated, and the results of the enumeration are stored in a working data store (step 1014). In some embodiments, the working data store is comprised of a memory element. In other embodiments, the working data store comprises a database or a key or a solid-state memory element or a persistent data store.

Next, the candidate key is identified as the application-scoped instance of the virtual key, and the category of existence of the candidate key is determined (step 1015). If the candidate key has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 1042).

In some embodiments, the candidate registry key may be associated with metadata indicating that the candidate registry key has been deleted. In some embodiments, metadata about a registry key may be stored in a distinguished value held by that key, with the existence of that value hidden from ordinary application usage of registry APIs. In some embodiments, small amounts of metadata about a registry key may be stored directly in the literal key name, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the key by virtual name check for possible variations of the literal key name due to the presence of a metadata indicator, and requests to retrieve the name of the key itself are hooked or intercepted in order to respond with the literal name. In other embodiments, the metadata indicator may be encoded in a subkey name or a registry value name instead of the key name itself. In still other embodiments, a registry key system may directly provide the ability to store some 3rd party metadata for each key. In some embodiments, metadata is stored in a database or other repository separate from the registry database. In some embodiments, a separate sub-scope may be used to store keys that are marked as deleted. The existence of a key in the sub-scope indicates that the key is marked as deleted.

If instead, in step 1015, the candidate key does not have negative existence, the candidate key is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each subkey in the enumeration, its category of existence is determined. Subkeys with negative

existence are removed from the working data store, and subkeys with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding subkey if one is already present in the working data store (step 1016).

In either case, the candidate key is identified as the user-scoped instance of the virtual key, and the category of existence of the candidate key is determined (step 1017). If the candidate key has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 1044). If instead the candidate key does not have negative existence, the candidate key is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each subkey in the enumeration, its category of existence is determined. Subkeys with negative existence are removed from the working data store, and subkeys with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding subkey if one is already present in the working data store (step 1018), followed by step 1030 as described below.

Then, for all three types of rules, step 1030 is executed. The rules engine is queried to find the set of rules whose filters match immediate children of the requested key, but do not match the requested key itself (step 1030). For each rule in the set, the existence of the virtual child whose name matches the name in the rule is determined. In some embodiments, this is determined by examining the appropriate isolation scope and the metadata associated with the virtual child. In other embodiments, this is determined by attempting to open the key. If the open request succeeds, the virtual child has positive existence. If the open request fails with an indication that the virtual child does not exist, the virtual child has negative existence.

If the child has positive existence, it is added to the working data store, replacing any child of the same name already there. If the child has negative existence, the child in the working data store corresponding to the virtual child, if any, is removed. (Step 1032). Finally, the constructed enumeration is then returned from the working data store to the requestor (step 1020).

A practitioner of ordinary skill in the art will realize that the layered enumeration process described above can be applied with minor modification to the operation of enumerating a single isolation scope which comprises a plurality of isolation sub-scopes. A working data store is created, successive sub-scopes are enumerated and the results are merged into the working data store to form the aggregated enumeration of the isolation scope.

4.2.4. Registry Creation Operations

Referring now to FIG. 11, and in brief overview, one embodiment of the steps taken to create a key in the isolation environment is shown. A request to create a key is received or intercepted (step 1102). The request contains a key name, which is treated as a virtual key name by the isolation environment. An attempt is made to open the requested key using full virtualization using applicable rules, i.e. using appropriate user and application isolation scope, as described in section 4.2.1 (step 1104). If access is denied (step 1106), an access denied error is returned to the requestor (step 1109). If access is granted (step 1106), and the requested key is successfully opened (step 1110), the requested key is returned to the requestor (step 1112). However, if access is granted (step 1106), but the requested key is not opened successfully (step 1110) then if the parent of the requested key also does not exist (step 1114), an error appropriate to the request semantics is issued to the requestor (step 1116). If on the other hand, the parent of the requested key is found in full virtualized view using the appropriate user and application scope (step 1114), a rule then determines how the key operation is processed (step 1118). If the rule action is "redirect" or "ignore" (step 1120), the virtual key name is mapped directly to a literal key name

according to the rule. Specifically, if the rule action is "ignore", the literal key name is identified as exactly the virtual key name. If, instead, the rule action is "redirect", the literal key name is determined from the virtual key name as specified by the rule. Then a request to create the literal key is passed to the operating system, and the result is returned to the requestor (step 1124). If on the other hand, the rule action determined in step 1120 is "isolate", then the literal key name is identified as the instance of the virtual key name in the user isolation scope. If the literal key already exists, but is associated with metadata indicating that it is a placeholder or that it is deleted, then the associated metadata is modified to remove those indications, and it is ensured that the key is empty. In either case, a request to open the literal key is passed to the operating system (step 1 126). If the literal key was opened successfully (step 1128), the literal key is returned to the requestor (step 1130). If on the other hand, in step 1128, the requested key fails to open, placeholders for each ancestor of the literal key that does not currently exist in the user-isolation scope (step 1132) and a request to create the literal key using the literal name is passed to the operating system and the result is returned to the requestor (step 1134).

Still referring to FIG. 11, and in more detail, a request to create a key is received or intercepted (step 1102). In some embodiments, the request is hooked by a function that replaces the operating system function or functions for creating the key. In another embodiment, a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for key operations. For embodiments in which a separate operating system function is provided for each type of key operation, each function may be hooked

separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of key operations.

The request contains a key name, which is treated as a virtual key name by the isolation environment. In some embodiments, the virtual key name may be expressed as a combination of a handle to a parent key, and the relative path name to the descendant key. The parent key handle is associated with a literal key name, which is itself associated with a virtual key name. The requestor attempts to open the virtual key using full virtualization using applicable rules, i.e. using appropriate user and application isolation scope, as described in section 4.2.1 (step 1104). If access is denied during the full virtualized open operation (step 1106), an access denied error is returned to the requestor (step 1109). If access is granted (step 1106), and the requested virtual key is successfully opened (step 1110), the corresponding literal key is returned to the requestor (step 1112). However, if access is granted (step 1106), but the virtual key is not opened successfully (step 1110) then the virtual key has been determined not to exist. If the virtual parent of the requested virtual key also does not exist, as determined by the procedures in section 4.2.1 (step 1114), an error appropriate to the request semantics is issued to the requestor (step 1116). If on the other hand, the virtual parent of the requested virtual key is found in full virtualized view using the appropriate user and application scope (step 1114), then a rule that determines how the create operation is processed is located (step 1118) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat key database. In some embodiments, the virtual key name provided for the requested key is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular key and, in some of these embodiments, the rule having the longest prefix match with the virtual key name is the rule applied to the request. In some

embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 11 as a single database transaction or single lookup into a key, the rule lookup may be performed as a series of rule lookups..

If the rule action is "redirect" or "ignore" (step 1120), the virtual key name is mapped directly to a literal key name according to the rule (step 1124). If the rule action is "redirect" (step 1120), the literal key name is determined from the virtual key name as specified by the rule (step 1124). If the rule action is "ignore" (step 1120), the literal key name is determined to be exactly the virtual key name (step 1124). If the rule action is "ignore" or the rule action is "redirect", a request to create the literal key using the determined literal key name is passed to the operating system and the result from the operating system is returned to the requestor (step 1124). For example, a request to create a virtual key named "key_1" may result in the creation of a literal key named "Different_ key_1." In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. (step 1124). In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. In these embodiments, creating the literal registry key may be achieved by responding to the original request to create the virtual key by signaling to the registry filter manager to reparse the request using the determined literal key name.

If the rule action determined in step 1120 is not "ignore" or "redirect" but is "isolate," then the literal key name is identified as the instance of the virtual key name in the user isolation scope. If the literal key already exists, but is associated with metadata indicating that it is a placeholder or that it is deleted, then the associated metadata is modified to remove those indications, and it is ensured that the key is empty.

In some embodiments, metadata about a registry key may be stored in a distinguished value held by that key, with the existence of that value hidden from ordinary application usage of registry APIs. In some embodiments, small amounts of metadata about a registry key may be stored directly in the literal key name, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the key by virtual name check for possible variations of the literal key name due to the presence of a metadata indicator, and requests to retrieve the name of the key itself are hooked or intercepted in order to respond with the literal name. In other embodiments, the metadata indicator may be encoded in a subkey name or a registry value name instead of the key name itself. In still other embodiments, a registry key system may directly provide the ability to store some 3rd party metadata for each key. In some embodiments, metadata could be stored in a database or other repository separate from the registry database. In some embodiments, a separate sub-scope may be used to store keys that are marked as deleted. The existence of a key in the sub-scope indicates that the key is marked as deleted.

In specific ones of these embodiments, a list of deleted keys or key system elements may be maintained and consulted to optimize this check for deleted keys. In these embodiments, if a deleted key is recreated then the key name may be removed from the list of deleted keys. In others of these embodiments, a key name may be removed from the list if the list grows beyond a certain size.

In either case, a request to open the user-scoped literal key is passed to the operating system (step 1126). In some embodiments, rules may specify that the literal key corresponding to the virtual key should be created in a scope other than the user isolation scope, such as the application isolation scope, the system scope, a user isolation sub-scope or an application isolation sub-scope.

If the literal key was opened successfully (step 1128), the literal key is returned to the requestor (step 1130). If on the other hand, in step 1128, the requested key fails to open, placeholders are created for each ancestor of the literal key that does not currently exist in the user-isolation scope (step 1132) and a request to create the literal key using the literal name is passed to the operating system and the result is returned to the requestor (step 1134).

This embodiment is for operating systems with APIs or facilities that only support creation of one level per call/invocation. Extension to multi-levels per call/invocation should be obvious to one skilled in the art.

4.3 Named Object Virtualization

Another class of system-scoped resources that may be virtualized using the techniques described above are named objects, which include semaphores, mutexes, mutants, waitable timers, events, job objects, sections, named pipes, and mailslots. These objects are characterized in that they typically exist only for the duration of the process which creates them. The name space for these objects may be valid over an entire computer (global in scope) or only in an individual user session (session scoped).

Referring now to FIG. 12, and in brief overview, a request to create or open a named object is received or intercepted (step 1202). That request contains an object name which is treated as a virtual name by the isolation environment. A rule determining how to treat the request is determined (step 1204). If the rule indicates that the request should be ignored (step 1206), the literal object name is determined to be the virtual name (step 1207), and a request to create or open the literal object is issued to the operating system (step 1214). If the determined rule is not to ignore the request, but indicates instead that the request should be redirected (step 1208), the literal object name is determined from the virtual name as specified by the redirection rule (step 1210) and a create or open request for the literal object is issued to the operating system (step 1214). If the rule does not indicate that the request should be

redirected (step 1208), but instead indicates that the request should be isolated, then the literal object name is determined from the virtual name as specified by the isolation rule (step 1212) and a create or open command for the literal object is issued to the operating system (step 1214). The handle of the literal object returned by the operating system in response to the issued create or open command is returned to the program requesting creation or opening of the virtual object (step 1216).

Still referring to FIG. 12, and in more detail, a request from a process to create or open a named object is intercepted (step 1202). The named object may be of session scope or it may be of global scope. In some embodiments, the request is hooked by a function that replaces the operating system function or functions for creating or opening the named object. In another embodiment a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for system objects. The request to create or open the named object may refer to any one of a wide variety of system-scoped resources that are used for interprocess communication and synchronization and that are identified by a unique identifier including semaphores, mutexes, mutants, waitable timers, file-mapping objects, events, job objects, sections, named pipes, and mailslots. For embodiments in which a separate operating system function is provided for each type of object, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of objects.

The intercepted request contains an object name which is treated as a virtual name by the isolation environment. A rule determining how to treat the

request for the object is determined (step 1204) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual name provided for the requested object is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular object and, in these embodiments, the rule having the longest prefix match with the virtual name is the rule applied to the request. In some embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 12 as a series of decisions, the rule lookup may occur as a single database transaction.

If the rule indicates that the request should be ignored (step 1206), the literal object name is determined to be the virtual name, and a request to create or open the literal object is issued to the operating system (step 1214). For example, a request to create or open a named object named "Object_1" will result in the creation of an actual object named "Object_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument.

If the rule determined by accessing the rules engine is not to ignore the request, but indicates instead that the request should be redirected (step 1208), the literal object name is determined from the virtual name as specified by the redirection rule (step 1210) and a create or open request for the literal object is issued to the operating system (step 1214). For example, a request to create or open a named object named "Object_1" may result in the creation of an actual object named "Different_Object_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument.

If the rule does not indicate that the request should be redirected (step 1208), but instead indicates that the request should be isolated, then the literal object name is determined from the virtual name as specified by the isolation rule (step 1212) and a create or open command for the literal object is issued to the operating system (step 1214). For example, a request to create or open a named object named "Object_1" may result in the creation of an actual object named "Isolated_Object_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument.

The literal name formed in order to isolate a requested system object may be based on the virtual name received and a scope-specific identifier. The scope-specific identifier may be an identifier associated with an application isolation scope, a user isolation scope, a session isolation scope, or some combination of the three. The scope-specific identifier is used to "mangle" the virtual name received in the request. For example, if the request for the named object "Object_1" is isolated for the application-isolation scope whose associated identifier is "SA1", the literal name may be "Isolated_AppScope_SA1_Object_1". The following table identifies the effect of mangling the name of an object with session isolation scope, or user isolation scope, and an application isolation scope. Mangling with combinations of scopes combines the restrictions listed in the table.

	Session-specific identifier	User-specific identifier	Application-specific identifier
Global object	Object available to all isolated applications executing in the context of the user session	Object available to all isolated applications executing on behalf of the user	Object available to all isolated applications executing in application isolation scope

Session object	Object available to all isolated applications executing in the context of the user session	Object available to all isolated applications executing in the session on behalf of the user	Object available to all isolated applications executing in the application isolation scope within the session

For embodiments in which the operating system is one of the WINDOWS family of operating systems, object scope may be modified by toggling the global/local name prefix associated with the object, which, for isolated applications, has the same effect as mangling the object name with a session-specific identifier. However, toggling the global/local name prefix also affects the object scope for non-isolated applications.

The handle of the literal object returned by the operating system in response to the command issued in step 1214 to create or open the named object is returned to the program requesting creation or opening of the virtual object (step 1216).

4.4 Window Name Virtualization

Other classes of system-scoped resources that may be virtualized using the techniques described above are window names and window class names. Graphical software applications use the name of a window or its window class as a way of identifying if an application program is already running and for other forms of synchronization. Referring now to FIG. 13, and in brief overview, a request concerning a window name or window class is received or intercepted (step 1302). A request can be in the form of a Win32 API call or in the form of a Window message. Both types of requests are handled. Those requests contain, or request retrieval of, window names and/or window class names that are treated as virtual names by the isolation environment. If the request is to retrieve

a window name or window class for a window identified by a handle (step 1304), a window mapping table is consulted to determine if the handle and the requested information concerning the window is known (step 1306). If so, the requested information from the window mapping table is returned to the requestor (step 1308). If not, the request is passed to the operating system (step 1310), and the result returned to the requestor (step 1314). If, in step 1304, the request provides a window name or window class, the request is checked to determine if it specifies one of a class of windows defined by the operating system (step 1320). If it does, the request is issued to the operating system and the result returned from the operating system is returned to the requestor (step 1322). If the request does not specify one of a class of windows defined by the operating system, the literal class name is determined based on the virtual class name and the rules (step 1324) and the literal window name is determined based on the virtual window name and the rules (step 1326). The request is then passed to the operating system using the literal window and literal class names (step 1328). If either the literal window name or literal window class name determined in steps 1324 and 1326 differ from the corresponding virtual name, then the window mapping table entry for the window handle is updated to record the virtual window name or virtual class name provided in the request (step 1330). If the response from the operating system includes native window names or native identifications of classes, those are replaced with the virtual window name or virtual class name provided in the request (step 1312) and the result is returned to the requestor (step 1314).

Still referring to FIG. 13, and in more detail a request concerning a window name or window class is received or intercepted (step 1302). Those requests contain or request retrieval of window names and/or window class names that are treated as virtual names by the isolation environment.

If the request is to retrieve a window name or window class for a window identified by a handle (step 1304), a window mapping table is consulted to

determine if the handle and the requested information concerning the window is known (step 1306). In some embodiments, instead of a mapping table, additional data is stored for each window and window class using facilities provided by the operating system.

If so, the requested information from the window mapping table is returned to the requestor (step 1308). If not, the request is passed to the operating system (step 1310), and the result returned to the requestor (step 1314).

If, in step 1304, the request provides a window name or window class, the request is checked to determine if it specifies one of a class of windows defined by the operating system (step 1320). If it does, the request is passed to the operating system and the result returned from the operating system is returned to the requestor (step 1322).

If the request does not specify one of a class of windows defined by the operating system, the literal class name is determined based on the virtual class name and the rules (step 1324) and the literal window name is determined based on the virtual window name and the rules (step 1326). The request is then passed to the operating system using the literal window and literal class names (step 1328). In some embodiments the window names and window class names may be atoms, rather than character string literals. Typically an application places a string in an atom table and receives a 16-bit integer, called an atom, that can be used to access the string.

If either the literal window name or literal window class name determined in steps 1324 and 1326 differ from the corresponding virtual name, then the window mapping table entry for the window handle is updated to record the virtual window name or virtual class name provided in the request (step 1330).

If the response from the operating system includes native window names or native identifications of classes, those are replaced with the virtual window name or virtual class name provided in the request (step 1312) and the result is returned to the requestor (step 1314).

Referring now to FIG 13A, the literal window name or window class name is determined as shown there. The rules engine is consulted to determine the rule that applies to the request (step 1352). If the rule action is "ignore" (step 1354), then the literal name equals the virtual name (step 1356). If, however, the rule action is not "ignore" but is "redirect" (step 1358), then the literal name is determined from the virtual name as specified by the redirect rule (step 1360). If, however, the rule action is not "redirect" but is "isolate", then the literal name is determined from the virtual name using a scope-specific identifier (step 1362).

In some embodiments, the particular scope-specific identifier is specified in the rule. In other embodiments, the scope-specific identifier used is the one associated with the application isolation scope with which the requesting process is associated. This allows the window or window class to be used by any other applications associated with the same application isolation scope. In operating systems such as many of the Microsoft WINDOWS family of operating systems where window names and classes are already isolated within a session, this means that only applications executing in the same session that are associated with the same application isolation scope can use the window name or class.

In some of the family of Microsoft WINDOWS operating systems, the window name is used as the title of the window in the title bar. It is desirable to handle non-client area paint window message to ensure that the window title displayed in the window title bar reflects the virtual names and not the literal name for a particular window. When a non-client area paint message is intercepted, the virtual name associated with the window, if any, is retrieved from the mapping table. If a virtual name is retrieved, the non-client area is painted using the virtual name as the window title and it is indicated that the request message has been handled. If no virtual name is retrieved, the request is indicated as not handled, which passes the request on to the original function that paints the title bar, using the literal name of the window.

4.5 Out-of-process COM Server Virtualization

Software component technologies such as COM, CORBA, .NET and others allow software components to be developed, deployed, registered, discovered, activated or instantiated and utilized as discrete units. In most component models, components may execute in either the process of the caller or in a separate process on the same computer or on a separate computer entirely, although some components may only support a subset of these cases.

One or more unique identifiers identify these components. Typically the component infrastructure provides a service or daemon that brokers activation requests. A software process that wishes to begin using a component passes a request to the broker to activate the component specified by the component identifier. The broker activates the requested component, if possible, and returns a reference to the activated instance. In some of these component infrastructures, multiple versions of the same component may not co-exist because the component identifier remains the same from version to version.

Some of the members of the WINDOWS family of operating systems provide a component infrastructure called COM. COM components ("COM servers") are identified by a GUID called a Class Identifier (CLSID), and each component provides one or more interfaces each of which has its own unique interface identifier (UIID). The COM Service Control Manager (CSCM) is the broker for out-of-process activation requests and it provides interfaces that allow the caller to request activation of a COM server via CLSID. Although the following description will be phrased in terms of COM servers and COM clients, it will be understood by one of ordinary skill in the art that it applies to CORBA, .NET, and other software architectures that provide for dynamic activation of software components.

When COM components are installed onto a computer, they register their CLSIDs in a well-known portion of the registry database, along with the information needed by the CSCM to launch a new instance of the COM server.

For out of process COM servers, this may include the path and command line parameters to the executable to run. Multiple versions of the same COM server share the same CLSID, hence only one version can be installed onto a computer at a time.

In certain embodiments, an application (acting as a COM client) instantiates a COM server by calling a COM API (for example, CoCreateInstance() or CoCreateInstanceEx()). A parameter to this call specifies the desired activation context: in-process; out-of-process on the same computer; out-of-process on a remote computer; or allow the COM subsystem to determine which of these three cases to use. If it is determined that an out-of-process activation is required, the request including the CLSID is passed to the CSCM. The CSCM uses the registry database to locate the path and parameters needed to launch the executable that hosts the COM server. When that executable is launched, it registers all of the CLSIDs of all of the COM servers that it supports with the CSCM using the COM API CoRegisterClassObject(). If the requested CLSID is registered, the CSCM returns a reference to that COM server to the caller. All subsequent interaction between the COM client and the COM server takes place independently of the CSCM.

The isolation environment 200 previously described allows multiple instances of COM servers with the same CLSID to be installed on a computer, each in a different isolation scope (no more than one of which may be the system scope). However, this alone will not make those COM servers available to COM clients.

Fig. 14 depicts one embodiment of the steps to be taken to virtualize access to COM servers. In brief overview, a new CLSID, hereinafter called the Isolated CLSID (or ICLSID) is created for each out-of-process COM server that is launched into an isolation scope (step 1402). By definition this is a CLSID, and thus must be unique amongst all other CLSIDs, in other words it must have the properties of a GUID. A mapping table is created that maps the pair (CLSID,

application isolation scope) to ICLSID. A COM server registry entry is created for the ICLSID which describes how to launch the COM server, with launch parameters that start the COM server executable in the appropriate application isolation scope (step 1404). Calls by COM clients to COM APIs such as CoCreateInstance() and CoCreateInstanceEx() are hooked or intercepted (step 1406). If it is determined that (a) the request can be satisfied by an in-process COM server or (b) both the COM client and COM server are not associated with any isolation scope, then the request is passed unmodified to the original COM API and the result returned to the caller (step 1408). The appropriate instance of the COM server to use is identified (step 1410). If the selected COM server instance is in an application isolation environment its ICLSID is determined using the data structures outlined above. Otherwise, the CLSID in the request is used (step 1412). The original CoCreateInstance() or CoCreateInstanceEx() function is called, with the ICLSID if one was identified in step 1412. This passes the request to the CSCM (step 1414). The CSCM finds and launches the COM server executable in the normal fashion, by looking up the requested CLSID in the registry to determine launch parameters. If an ICLSID is requested, the ICLSID system scope registry entry as described in step 1404 is found and the COM server is launched in the appropriate application isolation scope (step 1416). The launched COM executable calls the hooked CoRegisterClassObject() API with the CLSIDs of the COM servers it supports and these are translated to the appropriate ICLSIDs which are passed to the original CoRegisterClassObject() API (step 1418). When the CSCM receives a response from a CoRegisterClassObject() call with the expected ICLSID, it returns a reference to that COM server instance to the caller (step 1420).

Still referring to FIG. 14, and in more detail, an ICLSID is created for each out-of-process COM server that is launched into an isolation scope (step 1402). In some embodiments, the ICLSID is created during installation of the COM server. In other embodiments, the ICLSID is created immediately after

installation. In still other embodiments, the ICLSID is created before the COM server is launched into the isolation scope. In all of these embodiments, the ICLSID may be created by hooking or intercepting the system calls that create or query the CLSID entry in the registry database. Alternatively, the ICLSID may be created by hooking or intercepting the COM API calls such as CoCreateInstance() and CoCreateInstanceEx() that create COM server instances. Alternatively, changes to the CLSID-specific portion of the registry database may be observed after an installation has taken place.

A mapping table is created that maps the pair (CLSID, application isolation scope) to ICLSID, along with the appropriate registry entries for a COM server with that ICLSID that describe how to launch the COM server, with launch parameters that start the COM server executable in the appropriate application isolation scope (step 1404). In many embodiments, this table is stored in a persistent memory element, such as a hard disk drive or a solid-state memory element. In other embodiments, the table may be stored in the registry, in a flat file, in a database or in a volatile memory element. In still other embodiments, the table may be distributed throughout the COM-specific portions of the registry database, for example by adding a new subkey specific to this purpose to each appropriate COM server entry identified by CLSID. Entries in this table may be created during or immediately after installation, by hooking or intercepting the calls that create the CLSID entry in the registry database, or by observing changes to the CLSID-specific portion of the registry database after an installation has taken place, or by hooking or intercepting the COM API calls such as CoCreateInstance() and CoCreateInstanceEx() that create COM server instances. Installation of a COM server into a specific isolation scope may be persistently recorded. Alternatively, the mapping of a particular COM server and isolation scope to ICLSID may be dynamically created and stored as an entry in a non-persistent database, or in the registry database.

Calls by COM clients to COM APIs, such as CoCreateInstance() and CoCreateInstanceEx(), are hooked or intercepted (step 1406). If it is determined that (a) the request can be satisfied by an in-process COM server or (b) both the COM client and COM server reside in the system scope (step 1407), then the request is passed unmodified to the original COM API and the result returned to the caller (step 1408).

If the request cannot be satisfied by an in-process COM server and either the COM client or the COM server do not reside in the system scope (step 1407), then the appropriate instance of the COM server to use is identified (step 1410). For embodiments in which COM clients execute in a particular isolation scope, preference may be given to COM servers installed into the same application isolation scope, followed by those installed into the system scope (possibly executing in the client's application isolation scope), followed by COM servers installed into other application isolation scopes. In some of these embodiments, COM servers installed into the system scope may execute in the same application isolation scope as the COM client. This may be controlled by the rules engine and administrative settings to allow this to happen for COM servers that execute correctly in this mode, and prevent it for COM servers that do not. For embodiments in which the COM client executes in the system scope, preference may be given to system scope COM servers followed by COM servers in isolation scopes. The COM client may specify a COM server to use in the call creating an instance of the COM server. Alternatively, a configuration store may store information identifying the COM server to be instantiated. In some embodiments, the specified COM server is hosted by another computer, which may be a separate, physical machine or a virtual machine. The mapping table described above in connection with step 1404 may be used to find the set of applicable COM servers and (if necessary) compute preference based on rules.

For embodiments in which the applicable COM server exists on another computer, a service or daemon that executes on the remote computer can be queried for the ICLSID to use. The COM client hook, if it determines that a remote COM server is required, first queries the service or daemon to determine the CLSID/ICLSID to use. The service or daemon determines an ICLSID corresponding to the CLSID given in the request. In some embodiments, the ICLSID returned by the service or daemon may be selected or created based on administrator-defined configuration data, rules contained in a rules engine, or built-in hard-coded logic. In other embodiments, the request may specify the isolation scope on the server to be used. In still other embodiments, the requested COM server may be associated with the server's system scope, in which case the CLSID associated with the COM server is returned. In still other embodiments, the requested COM server may be associated with one of the server's isolation scopes, in which case it returns the ICLSID associated with the instance of the COM server and the isolation scope. In some embodiments, a service or daemon as described above may be used to support launching local out-of-process COM servers.

If the selected COM server instance is in an application isolation environment on the local computer, its ICLSID is determined using the data structures described in connection with step 1404. If, instead, the selected COM server instance is in the system scope on the local computer, the CLSID in the request is used (step 1412). In some of these embodiments, an entry for the COM server using the ICLSID may be dynamically created.

If an ICLSID is returned, it is passed to the original COM API in place of the original CLSID. For example, the determined ICLSID may be passed to the original CoCreateInstance() or CoCreateInstanceEx() function, which passes the request to the CSCM (step 1414). For embodiments in which the COM server is hosted by another computer, the CSCM passes the ICLSID to the computer

hosting the COM server, where that computer's CSCM handles the COM server launch.

The CSCM finds and launches the COM server executable in the normal fashion, by looking up the requested CLSID or ICLSID in the registry to determine launch parameters. If an ICLSID is requested, the ICLSID system scope registry entry as described in step 1404 is found and the COM server launched in the appropriate application isolation scope (step 1416).

If the launched COM server instance executes in an application isolation scope (whether installed into that scope or installed into the system scope), the COM API function CoRegisterClassObject()of the COM server instance is hooked or intercepted. Each CLSID that is passed to CoRegisterClassObject() is mapped to the corresponding ICLSID, using the mapping table as defined in step 1404. The original CoRegisterClassObject() API is called with the ICLSID (step 1418).

When the CSCM receives a response from a CoRegisterClassObject() call with the expected ICLSID, it returns a reference to that COM server instance to the caller (step 1420).

This technique supports COM server execution when the COM client and COM server execute in any combination of application isolation scopes (including different scopes) and the system scope. The ICLSID is specific to the combination of server (identified by CLSID) and the desired appropriate isolation scope. The client need only determine the correct ICLSID (or the original CLSID if the server is in installed into and executing in the system scope).

4.6 Virtualized File Type Association (FTA)

File type association is a well-known graphical user interface technique for invoking execution of application programs. A user is presented with a graphical icon representing a data file. The user selects the data file using keyboard commands or using a pointing device, such as a mouse, and clicks, or double-clicks, on the icon to indicate that the user would like to open the file. Alternately,

in some computing environments, the user enters the path to the file at a command line prompt in place of a command. The file typically has an associated file type indication which is used to determine an application program to use when opening the file. This is generally done using a table that maps the file type indication to a specific application. In many members of the family of Microsoft WINDOWS operating systems, the mapping is typically stored in the registry database in a tuple including the file type indicator and the full pathname identifying the application to be executed, and only one application program may be associated with any particular file type.

In the described isolation environment, multiple versions of an application may be installed and executed on a single computer. Thus, in these environments, the relationship between file type and associated application program is no longer a one-to-one relationship but is, instead, a one-to-many relationship. A similar problem exists for MIME attachment types. In these environments, this problem is solved by replacing the pathname identifying the application program to be launched when a given file type is selected. The pathname is replaced with that of a chooser tool that gives to the user a choice of application programs to launch.

Referring now to FIG. 15, and in brief overview, a request to write file type association data to a configuration store is intercepted (step 1502). A determination is made whether the request is updating the file type association information in the configuration store (step 1504). If not, i.e., if the entry already exists, no update occurs (step 1506). Otherwise, a new entry is created using the virtualization techniques described above in sections 4.1.4 or 4.2.4, or the existing entry is updated (step 1508). The new or updated entry, which is virtualized for the appropriate isolation scope, maps the file type to a chooser tool which allows the user to select which of multiple application programs to use when viewing or editing the file.

Still referring to FIG. 15, and in more detail, a request to write file-type association data to a configuration store is intercepted (step 1502). In some embodiments, the configuration store is the WINDOWS registry database. The request to write data to the configuration store may be intercepted by a user mode hooking function, a kernel mode hooking function, a file system filter driver, or a mini-driver.

A determination is made whether the request seeks to update file-type-association information in the configuration store (step 1504). In one embodiment this is accomplished by detecting if the intercepted request indicates that it intends to modify the configuration store. In another embodiment, the target of the request is compared to the information included in the request to determine if the request is attempting to modify the configuration store. For embodiments in which the configuration store is a registry database, the request to modify the registry is intercepted, as described above in Section 4.2.

If it is determined that the request is not attempting to update the configuration store, no update occurs (step 1506). In some embodiments, it is determined that no attempt to update the configuration store is made because the intercepted request is a read request. In other embodiments, this determination may made when the target entry in the configuration store and the information included in the intercepted request are identical, or substantially identical.

If, however, it is determined in step 1504 that the request intends to update the configuration store, then a new entry is created in the configuration store, or the existing entry is updated (step 1508). In some embodiments, rules determine which isolation scope the entry is created or updated in. In some embodiments, the new entry is created or the existing entry is updated in the system scope or the application isolation scope. In many embodiments, the new entry is created or the existing entry is updated in the appropriate user isolation scope. If a new entry is created, then it, rather than identifying the application

program identified in the intercepted request, lists a chooser application as the application to be used when a file of a particular type is accessed. In some embodiments, the chooser tool is updated automatically when a new version of an application program is installed, or when another application that handles the same file type is installed, or when an application registers or deregisters itself to handle files of that particular type. In some embodiments, the chooser tool can incorporate into its list of suitable applications any applications registered to handle the same file type in the portion of the configuration store maintained in other scopes, such as the system scope, and the application scope if the chooser tool executes in the user scope. If an existing entry is updated, and the existing entry already lists the chooser application as the application to be used when a file of that particular file type is used, then the list of applications presented by the chooser for that file type may be updated to include the updating application. If the existing entry is updated, but it does not list the chooser application, then the updated entry is made to list the chooser application as the application to be used when a file of that particular file type is used. In these embodiments, the information relating to the associated applications may be stored in an associated configuration file or, in some embodiments, as an entry in the registry database.

The chooser application may present to the user a list of applications associated with the selected file type. The application may also allow the user to choose the application program that the user would like to use to process the file. The chooser then launches the application program in the appropriate scope: system scope; application isolation scope; or user isolation scope. In some embodiments, the chooser tool maintains the identity of the default application program associated with a file type. In these embodiments, the default application may be used by processes that do not have access to the desktop or are configured to use the default handler without presenting the user with a choice.

4.7 Dynamic movement of processes between isolation environments

An additional aspect of the invention is the facility to move a running process between different virtual scopes. In other words, the aggregated view of native resources presented to the application instance by the isolation environment 200 may be changed to a different aggregated view while the application is executing. This allows processes that have been isolated within a particular isolation scope to be "moved" to another isolation scope while the process is running. This is particularly useful for system services or processes of which only one instance may execute at a time, such as the MSI service in WINDOWS operating systems. This aspect of the invention may also be used to allow a user to work in several isolation scopes sequentially.

Referring to FIG. 16, and in brief overview, one embodiment of a process for moving processes between one isolation scope and a second isolation scope, or between the system scope and an isolation scope, is shown. As used in this description, the term "target isolation scope" will be used to refer to the isolation scope, including the system scope, to which the processes is being moved and the term "source isolation scope" will be used to refer to the isolation scope, including the system scope, from which the process is being moved. As shown in FIG. 16, and in brief overview, a method for moving a process to a target isolation scope includes the steps of: ensuring that the process is in a safe state (step 1602); changing the association of the process from its source isolation scope to the target isolation scope in the rules engine (step 1604); changing the association of the process from the source isolation scope to the target isolation scope for any filter driver or hooks (step 1606); and allowing the process to resume execution (step 1608).

Still referring to FIG. 16, and in more detail, the process should be in a "safe" state while being moved to a different isolation scope (step 1602). In some embodiments, the process is monitored to determine when it is not processing requests. In these embodiments, the process is considered to be in a

"safe" state for moving when no requests are processed by the process. In some of these embodiments, once the process is considered to be in a "safe" state, new requests to the process are delayed until the process is moved. In other embodiments, such as in connection with diagnostic applications, a user interface may be provided to trigger the change in isolation scope. In these embodiments, the user interface may run code that puts the process to be moved into a "safe" state. In still other embodiments, an administration program may force the process into a "safe" state by delaying all incoming requests to the process and waiting for the process to complete execution of any active requests.

The rules associated with the target isolation scope are loaded into the rules engine if they do not already exist in the rules engine (step 1603).

The association of the process with a source isolation scope is changed in the rules engine (step 1604). As described above, a process can be associated with any isolation scope. That association is used by the rules engine on every request for a virtual native resource to determine the rule to apply to the request. The application instance can be associated with a target isolation scope by changing the appropriate data structures in the rules engine. In some embodiments, a new database entry is written associating the process with a new isolation scope. In other embodiments, a tree node storing an identifier for the isolation scope with which the process is associated is overwritten to identify the new isolation scope. In still other embodiments, an operating system request can made to allocate additional storage for a process to store the rules associated with the target isolation scope or, in some embodiments, an identifier of the rules.

The association of the process with the source isolation scope is changed wherever the association or the rules are stored outside of the rules engine, such as filter drivers, kernel mode hooks, or user mode hooks (step 1606). For embodiments in which the association between a process and isolation scope

rules is maintained based on PID, the association between the processes PID and the rule set is changed. For embodiments in which a PID is not used to maintain the association between a process and the applicable set of isolation rules, the user mode hooking function may be altered to access the rule set associated with the target isolation scope. For embodiments in which process associations with rule sets for isolation scopes are maintained in a rule engine, it is sufficient to change the association stored in the rule engine in step 1604 above.

The process is allowed to resume execution in the new isolation scope (step 1610). For embodiments in which new requests were delayed or prohibited from being made, those requests are issued to the process and new requests are allowed.

In one particularly useful aspect, the method described above may be used to virtualize MSI, an installation packaging and installation technology produced by Microsoft and available in some of the Microsoft WINDOWS family of operating systems. An application packaged by this technology for installation is called an MSI package. Operating systems which support this technology have a WINDOWS service called the MSI service which assists in installing MSI packages. There is a single instance of this service on the system. Processes that wish to install MSI packages run an MSI process in their session which makes COM calls to the MSI service.

MSI installations can be virtualized to install MSI packages into an application isolation environment. Conceptually, this can be achieved by hooking or intercepting the calls made to the MSI API in the installation session to the MSI service. A mutex can be used to ensure that only one installation takes place at a time. When a call to the MSI API requesting to start a new installation is received or intercepted, and the calling process is associated with a particular application isolation scope, the MSI service is placed into the context of that isolation scope before the call is allowed to proceed. The installation proceeds

as the MSI service performs its normal installation actions, although native resource requests by the MSI service are virtualized according to the applicable isolation scope. When the end of the installation process is detected, the association between the MSI service and the isolation scope is removed. Although described above with respect to MSI, the technique described is applicable to other installation technologies.

EQUIVALENTS

The present invention may be provided as one or more computer-readable programs embodied on or in one or more articles of manufacture. The article of manufacture may be a floppy disk, a hard disk, a CD-ROM, a flash memory card, a PROM, a RAM, a ROM, or a magnetic tape. In general, the computer-readable programs may be implemented in any programming language, LISP, PERL, C, C++, PROLOG, or any byte code language such as JAVA. The software programs may be stored on or in one or more articles of manufacture as object code.

Having described certain embodiments of the invention, it will now become apparent to one of skill in the art that other embodiments incorporating the concepts of the invention may be used. Therefore, the invention should not be limited to certain embodiments, but rather should be limited only by the spirit and scope of the following claims.

CLAIMS

What is claimed is:

1. A method for moving an executing process from a first isolation scope to a second isolation scope, the method comprising the steps of:

(a) determining that a process is in a state suitable for moving;

(b) changing an association of the process from a first isolation scope to a second isolation scope; and

(c) loading at least one rule associated with the second isolation scope.

2. The method of claim 1 wherein step (a) comprises determining that a process is in a state suitable for moving by monitoring whether the process is processing a request.

3. The method of claim 2 wherein step (a) comprises putting a process in a state suitable for moving.

4. The method of claim 3 further comprising putting the process in the state suitable for moving via one of a user interface and an administration program.

5. The method of claim 3 further comprising the step of prohibiting new requests to the process.

6. The method of claim 3 further comprising the step of queuing requests to the process.

7. The method of claim 6 further comprising the step of processing the queued requests after associating the process with the second isolation scope.

8. The method of claim 1 wherein step (b) comprises writing to a rules engine information associating the process with a second isolation scope.

9. The method of claim 1 further comprising the step of changing an association of the process from the first isolation scope to the second isolation scope in a file system filter driver.

10. The method of claim 1 further comprising the step of changing an association of the process from the first isolation scope to the second isolation scope in one of a kernel hooking function and a user mode hooking function.

11. The method of claim 1 further comprising the step of changing an association of the process from the first isolation scope to the second isolation scope in a mini-filter.

12. The method of claim 1 further comprising the step of suspending execution by the process.

13. The method of claim 12 further comprising resuming execution by the process.

14. A method for moving an executing process into an isolation scope, the method comprising the steps of:

(a) determining that a process is in a state suitable for moving;
(b) associating the process with an isolation scope; and
(c) loading at least one rule associated with the isolation scope.

15. The method of claim 14 wherein step (a) comprises determining that a process is in a state suitable for moving by monitoring whether the process is processing a request.

16. The method of claim 15 wherein step (a) comprises putting a process in a state suitable for moving.

17. The method of claim 16 further comprising putting the process in the state suitable for moving via one of a user interface and an administration program.

18. The method of claim 16 further comprising the step of prohibiting new requests to the process.

19. The method of claim 16 further comprising the step of queuing requests to the process.

20. The method of claim 19 further comprising the step of processing the queued requests after associating the process with the isolation scope.

21. The method of claim 14 wherein step (b) comprises writing to a rules engine information associating the process with an isolation scope.

22. The method of claim 14 further comprising the step of associating the process to the isolation scope in a file system filter driver.

23. The method of claim 14 further comprising the step of associating the process to the isolation scope in one of a kernel hooking function and user mode hooking function.

24. The method of claim 14 further comprising the step of associating the process to the isolation scope in a mini-filter.

25. The method of claim 14 further comprising the step of suspending execution by the process.

26. The method of claim 25 further comprising resuming execution by the process.

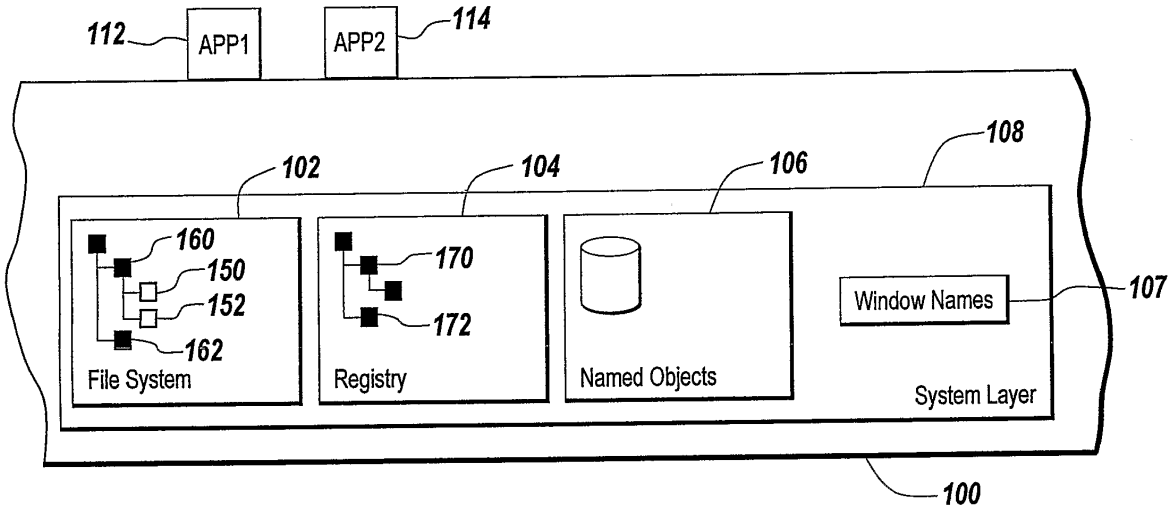


Fig. 1A
(Prior Art)

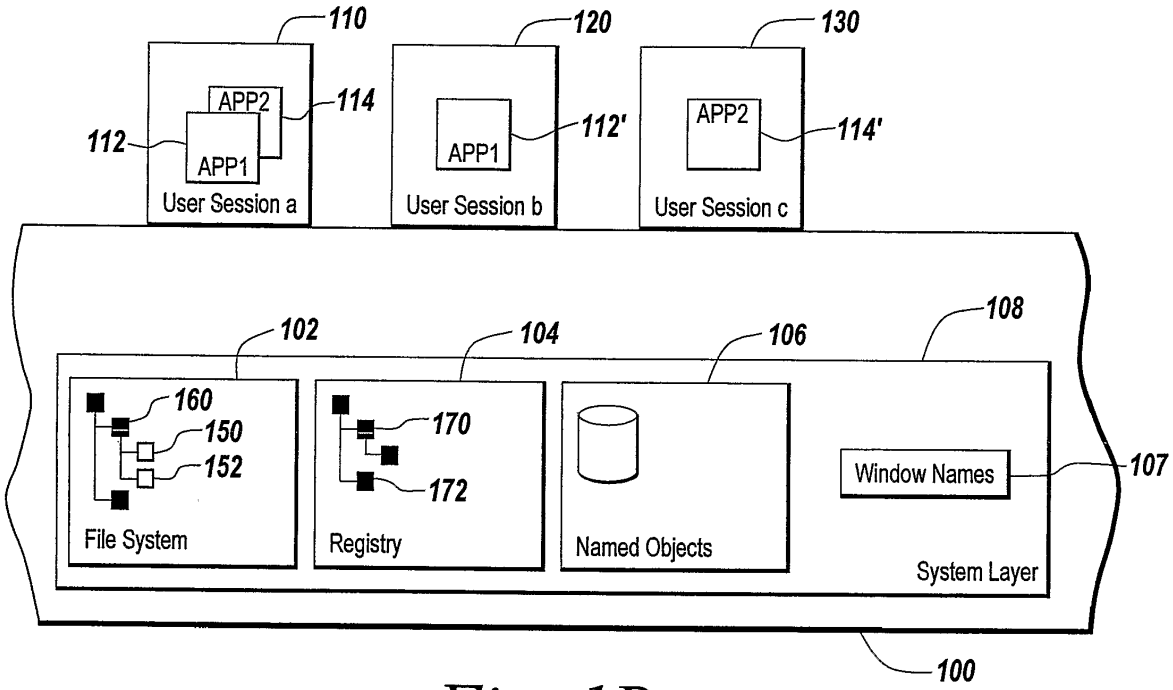


Fig. 1B
(Prior Art)

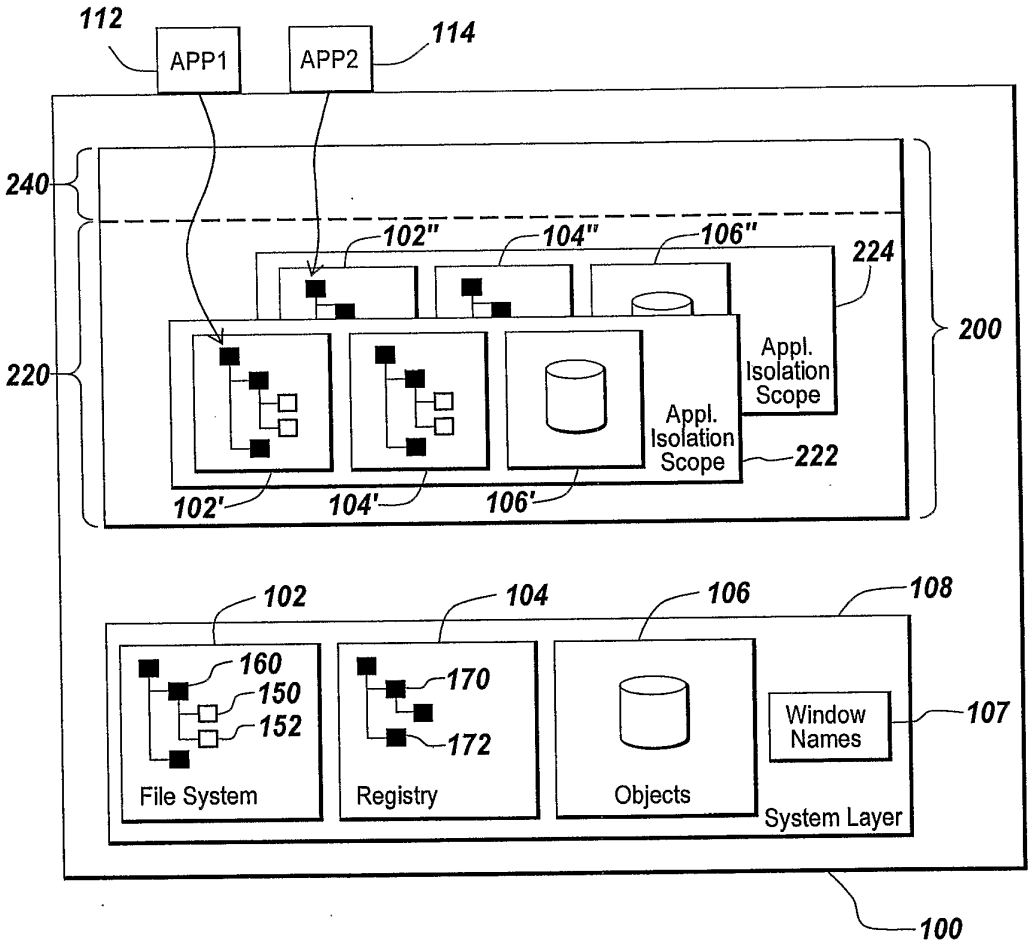

112
APP1
APP2
114
240
220
102"
104"
106"
224
Appl. Isolation Scope
Appl. Isolation Scope
222
200
102'
104'
106'
102
104
106
108
160
150
152
170
172
File System
Registry
Objects
Window Names
107
System Layer
100

Fig. 2A

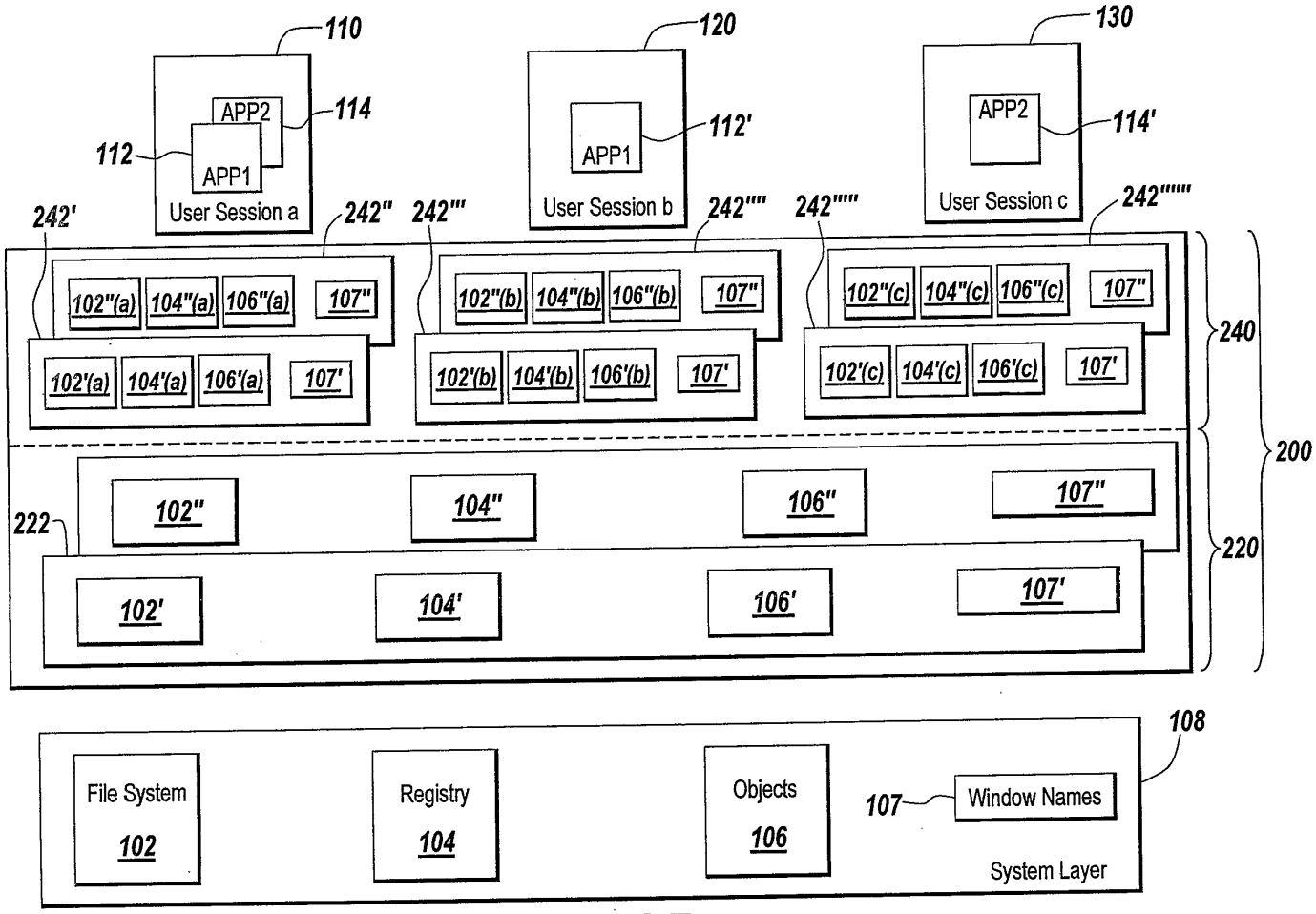

110
120
130
APP2
114
112
APP1
User Session a
112'
APP1
User Session b
APP2
114'
User Session c
242'
242"
242'''
242''''
242'''''
242''''''
102"(a)
104"(a)
106"(a)
107"
102'(a)
104'(a)
106'(a)
107'
102"(b)
104"(b)
106"(b)
107"
102'(b)
104'(b)
106'(b)
107'
102"(c)
104"(c)
106"(c)
107"
102'(c)
104'(c)
106'(c)
107'
240
200
220
222
102"
104"
106"
107"
102'
104'
106'
107'
108
File System
102
Registry
104
Objects
106
107
Window Names
System Layer

Fig. 2B

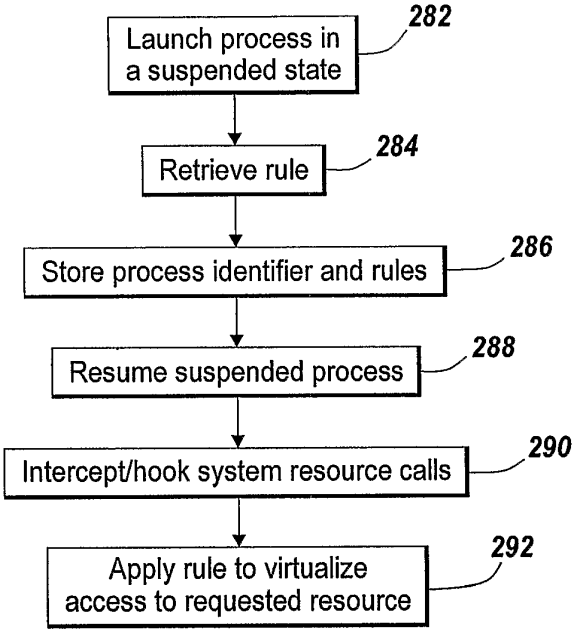

Launch process in a suspended state
282
Retrieve rule
284
Store process identifier and rules
286
Resume suspended process
288
Intercept/hook system resource calls
290
Apply rule to virtualize access to requested resource
292

Fig. 2C

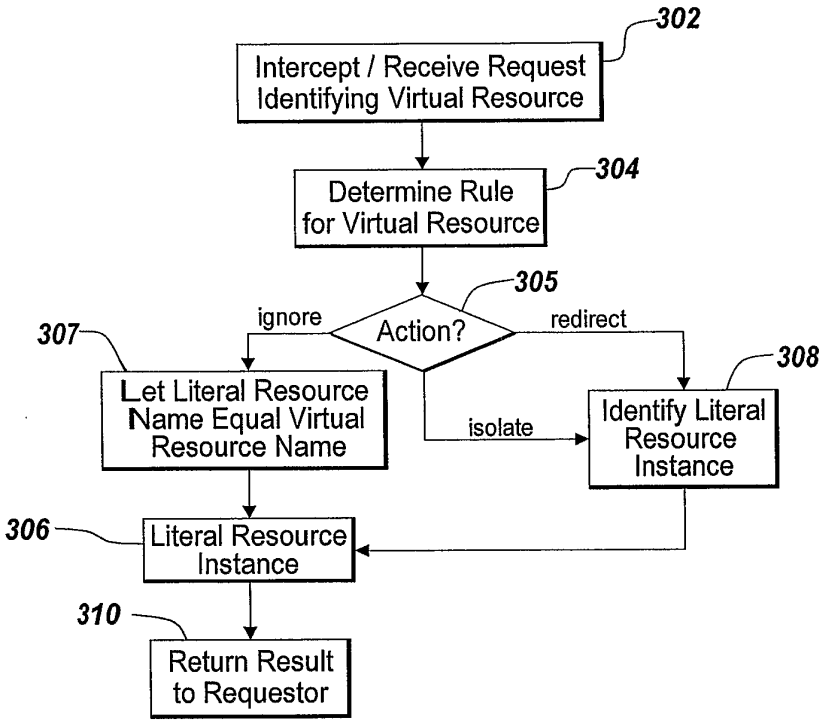

302
Intercept / Receive Request Identifying Virtual Resource
304
Determine Rule for Virtual Resource
305
Action?
ignore
redirect
isolate
307
Let Literal Resource Name Equal Virtual Resource Name
308
Identify Literal Resource Instance
306
Literal Resource Instance
310
Return Result to Requestor

Fig. 3A

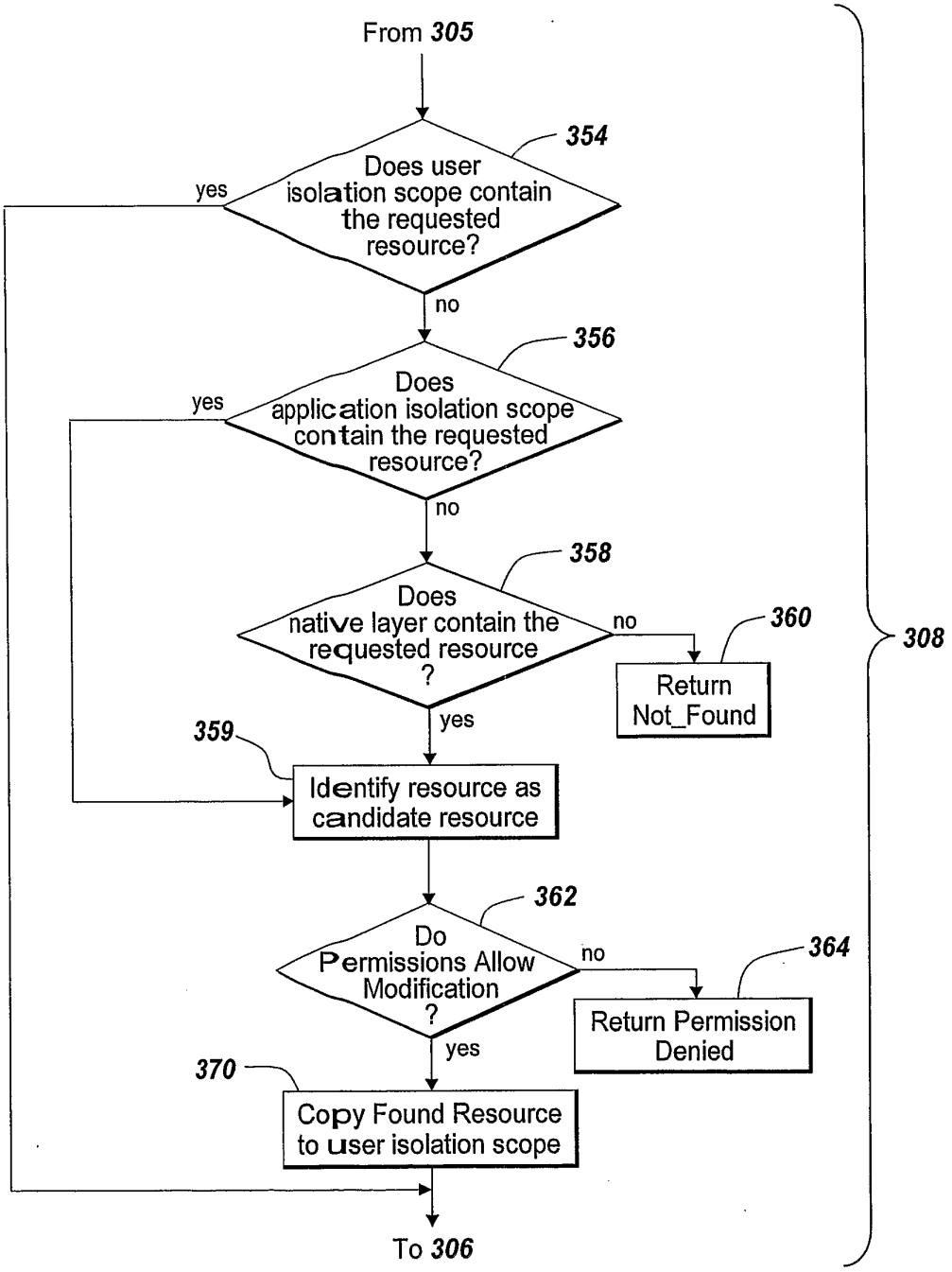

From 305
354
Does user isolation scope contain the requested resource?
yes
no
356
Does application isolation scope contain the requested resource?
yes
no
358
Does native layer contain the requested resource ?
no
yes
360
Return Not_Found
308
359
Identify resource as candidate resource
362
Do Permissions Allow Modification ?
no
yes
364
Return Permission Denied
370
Copy Found Resource to user isolation scope
To 306

Fig. 3B

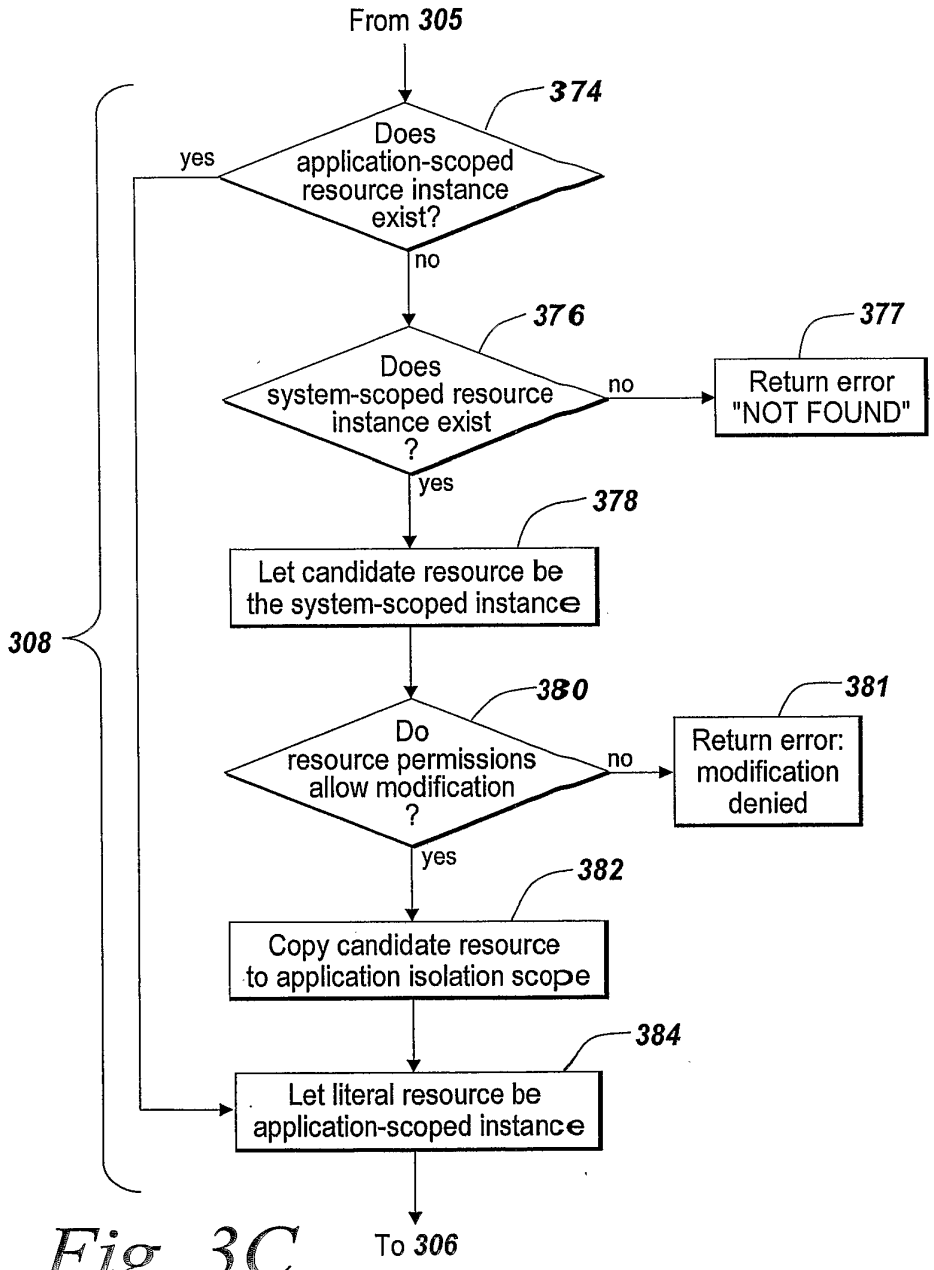

From 305
374
Does application-scoped resource instance exist?
yes
no
376
Does system-scoped resource instance exist ?
no
377
Return error "NOT FOUND"
yes
378
Let candidate resource be the system-scoped instance
308
380
Do resource permissions allow modification ?
no
381
Return error: modification denied
yes
382
Copy candidate resource to application isolation scope
384
Let literal resource be application-scoped instance
To 306

Fig. 3C

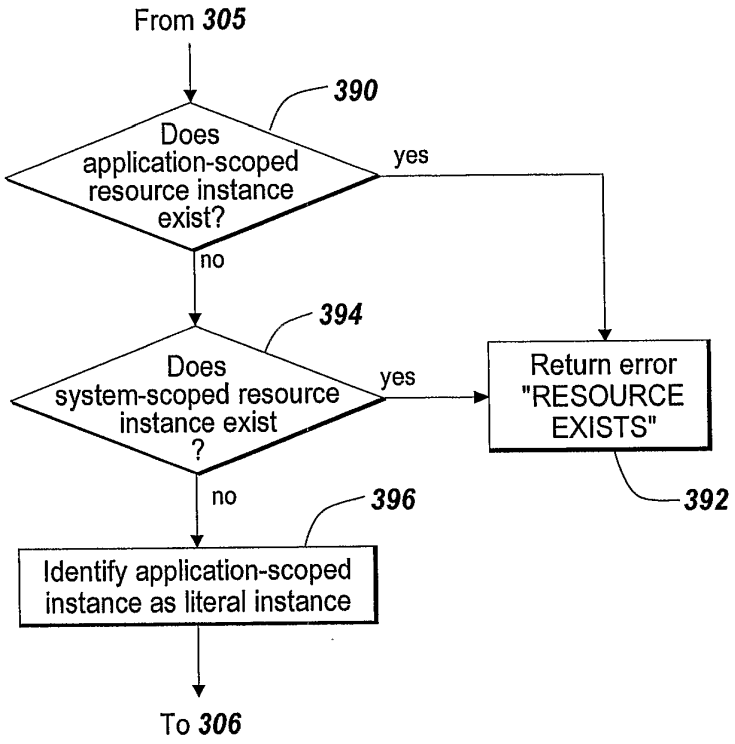

From 305
390
Does application-scoped resource instance exist?
yes
no
394
Does system-scoped resource instance exist ?
yes
Return error "RESOURCE EXISTS"
392
no
396
Identify application-scoped instance as literal instance
To 306

Fig. 3D

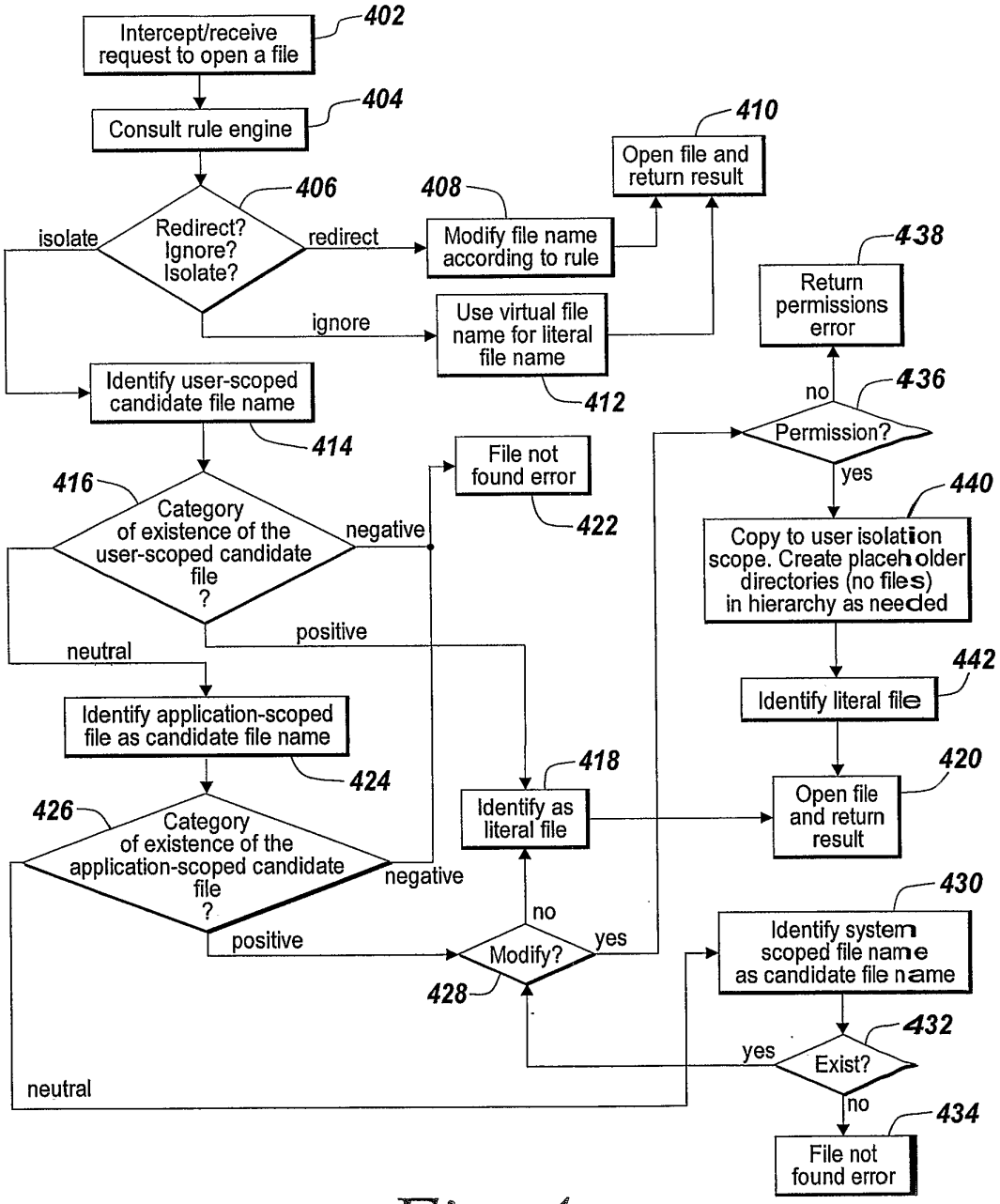

Intercept/receive request to open a file
402
Consult rule engine
404
Redirect? Ignore? Isolate?
406
isolate
redirect
ignore
Modify file name according to rule
408
Open file and return result
410
Use virtual file name for literal file name
412
Identify user-scoped candidate file name
414
Category of existence of the user-scoped candidate file ?
416
negative
positive
neutral
File not found error
422
Identify application-scoped file as candidate file name
424
Category of existence of the application-scoped candidate file ?
426
negative
positive
neutral
Identify as literal file
418
Modify?
428
no
yes
Permission?
436
no
yes
Return permissions error
438
Copy to user isolation scope. Create placeholder directories (no files) in hierarchy as needed
440
Identify literal file
442
Open file and return result
420
Identify system scoped file name as candidate file name
430
Exist?
432
yes
no
File not found error
434

Fig. 4

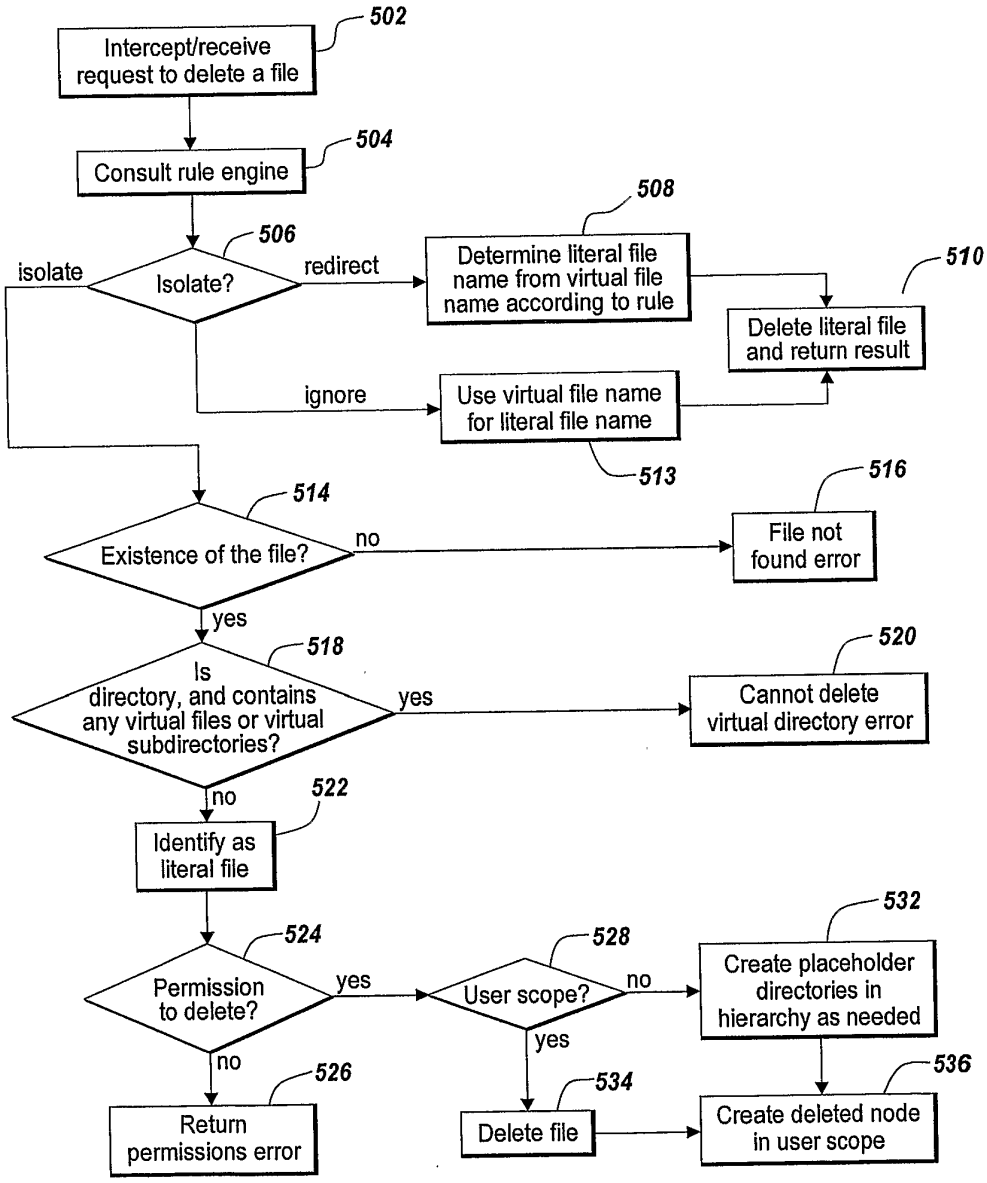

502
Intercept/receive request to delete a file
504
Consult rule engine
506
Isolate?
isolate
redirect
508
Determine literal file name from virtual file name according to rule
510
Delete literal file and return result
ignore
Use virtual file name for literal file name
513
514
Existence of the file?
no
516
File not found error
yes
518
Is directory, and contains any virtual files or virtual subdirectories?
yes
520
Cannot delete virtual directory error
no
522
Identify as literal file
524
Permission to delete?
yes
528
User scope?
no
532
Create placeholder directories in hierarchy as needed
no
526
Return permissions error
yes
534
Delete file
536
Create deleted node in user scope

Fig. 5

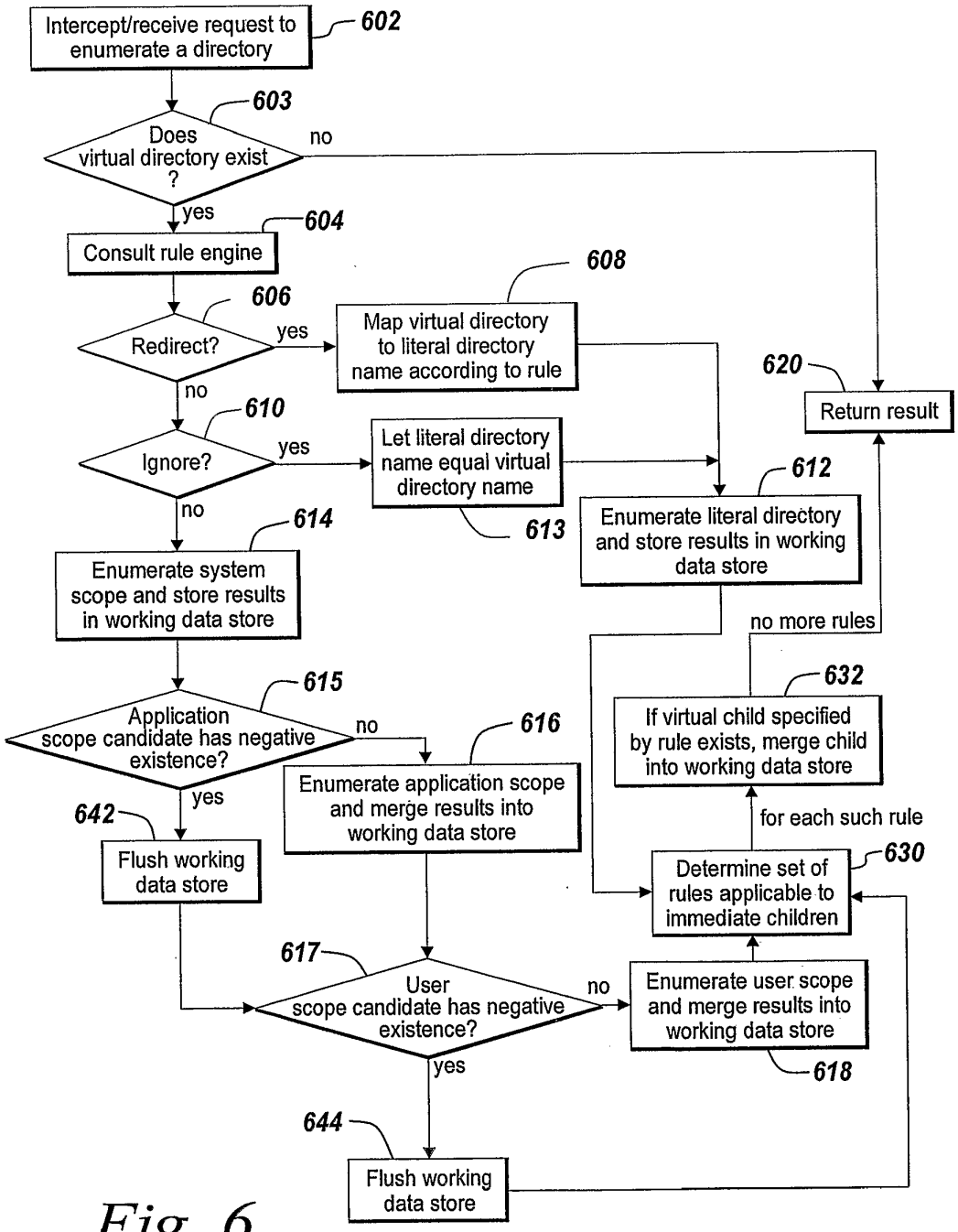

Intercept/receive request to enumerate a directory
602
603
Does virtual directory exist ?
no
yes
604
Consult rule engine
606
Redirect?
yes
608
Map virtual directory to literal directory name according to rule
no
610
Ignore?
yes
Let literal directory name equal virtual directory name
613
no
614
Enumerate system scope and store results in working data store
620
Return result
612
Enumerate literal directory and store results in working data store
no more rules
632
If virtual child specified by rule exists, merge child into working data store
615
Application scope candidate has negative existence?
no
616
Enumerate application scope and merge results into working data store
yes
642
Flush working data store
for each such rule
630
Determine set of rules applicable to immediate children
617
User scope candidate has negative existence?
no
Enumerate user scope and merge results into working data store
618
yes
644
Flush working data store

Fig. 6

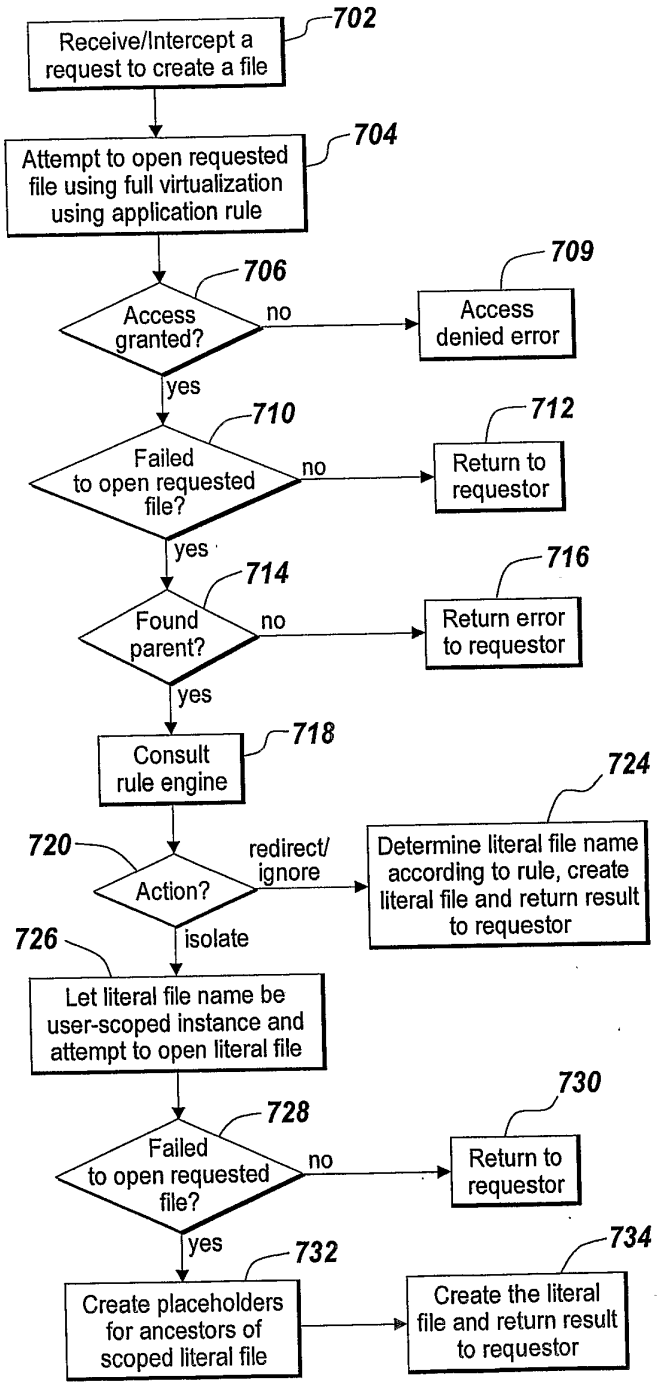

Receive/Intercept a request to create a file
702
Attempt to open requested file using full virtualization using application rule
704
Access granted?
706
no
Access denied error
709
yes
Failed to open requested file?
710
no
Return to requestor
712
yes
Found parent?
714
no
Return error to requestor
716
yes
Consult rule engine
718
Action?
720
redirect/ ignore
Determine literal file name according to rule, create literal file and return result to requestor
724
isolate
Let literal file name be user-scoped instance and attempt to open literal file
726
Failed to open requested file?
728
no
Return to requestor
730
yes
Create placeholders for ancestors of scoped literal file
732
Create the literal file and return result to requestor
734

Fig. 7

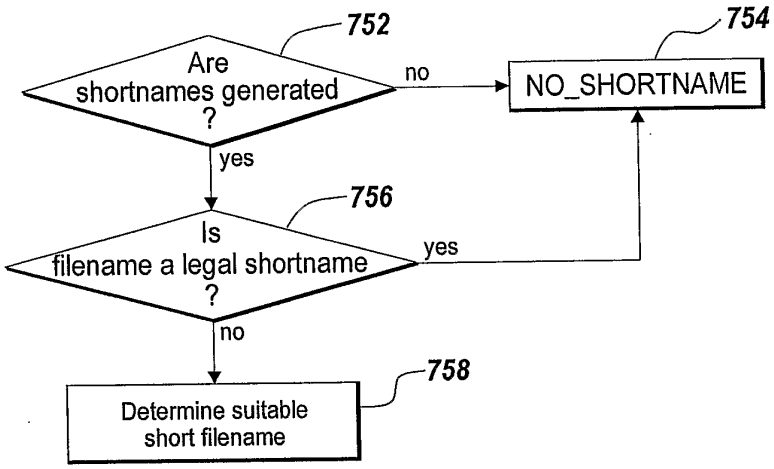

752
Are shortnames generated ?
no
754
NO_SHORTNAME
yes
756
Is filename a legal shortname ?
yes
no
758
Determine suitable short filename

Fig. 7A

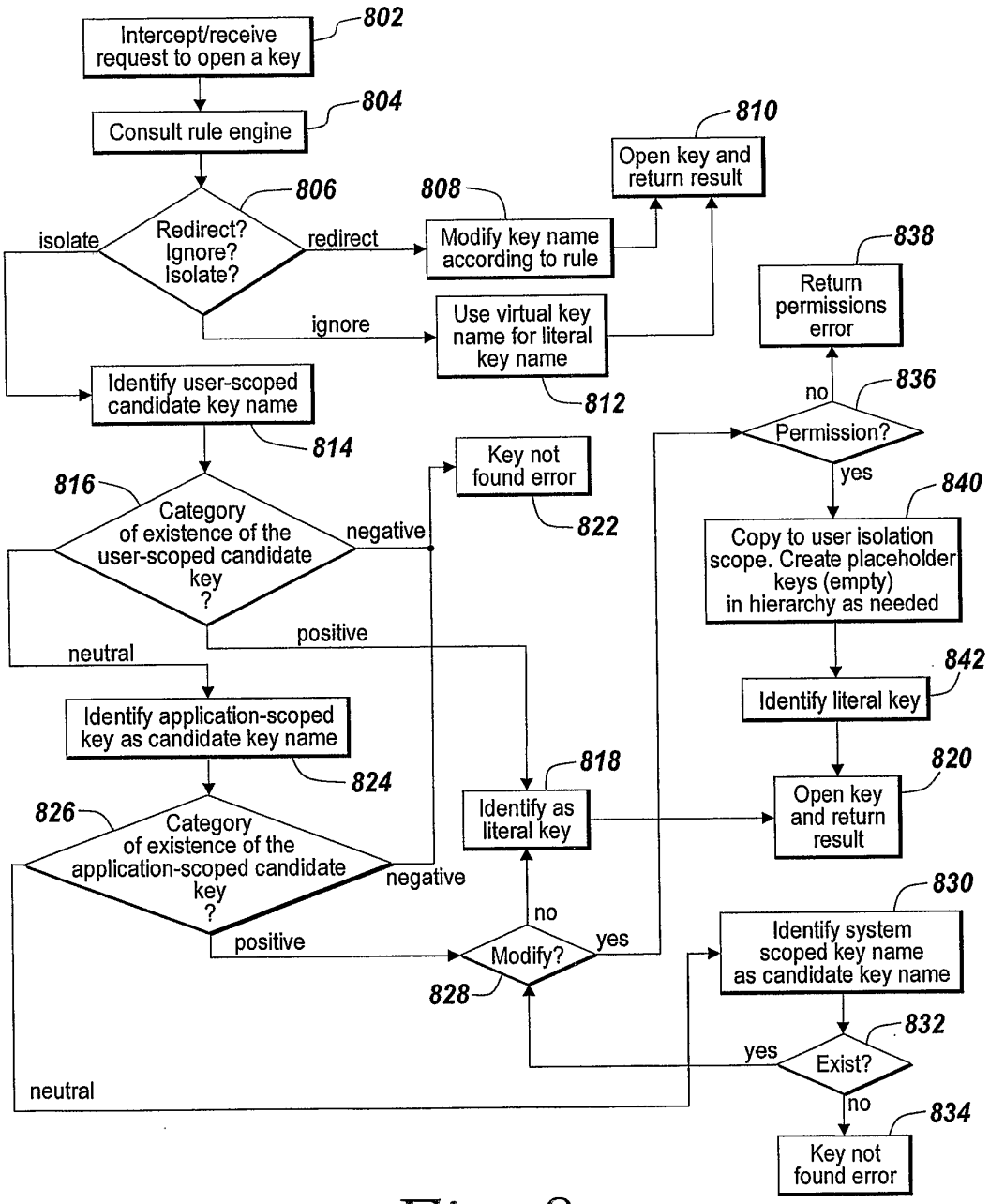

Intercept/receive request to open a key
802
Consult rule engine
804
Redirect? Ignore? Isolate?
806
isolate
redirect
ignore
Modify key name according to rule
808
Open key and return result
810
Use virtual key name for literal key name
812
Identify user-scoped candidate key name
814
Category of existence of the user-scoped candidate key ?
816
negative
positive
neutral
Key not found error
822
Identify application-scoped key as candidate key name
824
Category of existence of the application-scoped candidate key ?
826
negative
positive
neutral
Modify?
828
no
yes
Identify as literal key
818
Permission?
836
no
yes
Return permissions error
838
Copy to user isolation scope. Create placeholder keys (empty) in hierarchy as needed
840
Identify literal key
842
Open key and return result
820
Identify system scoped key name as candidate key name
830
Exist?
832
yes
no
Key not found error
834

Fig. 8

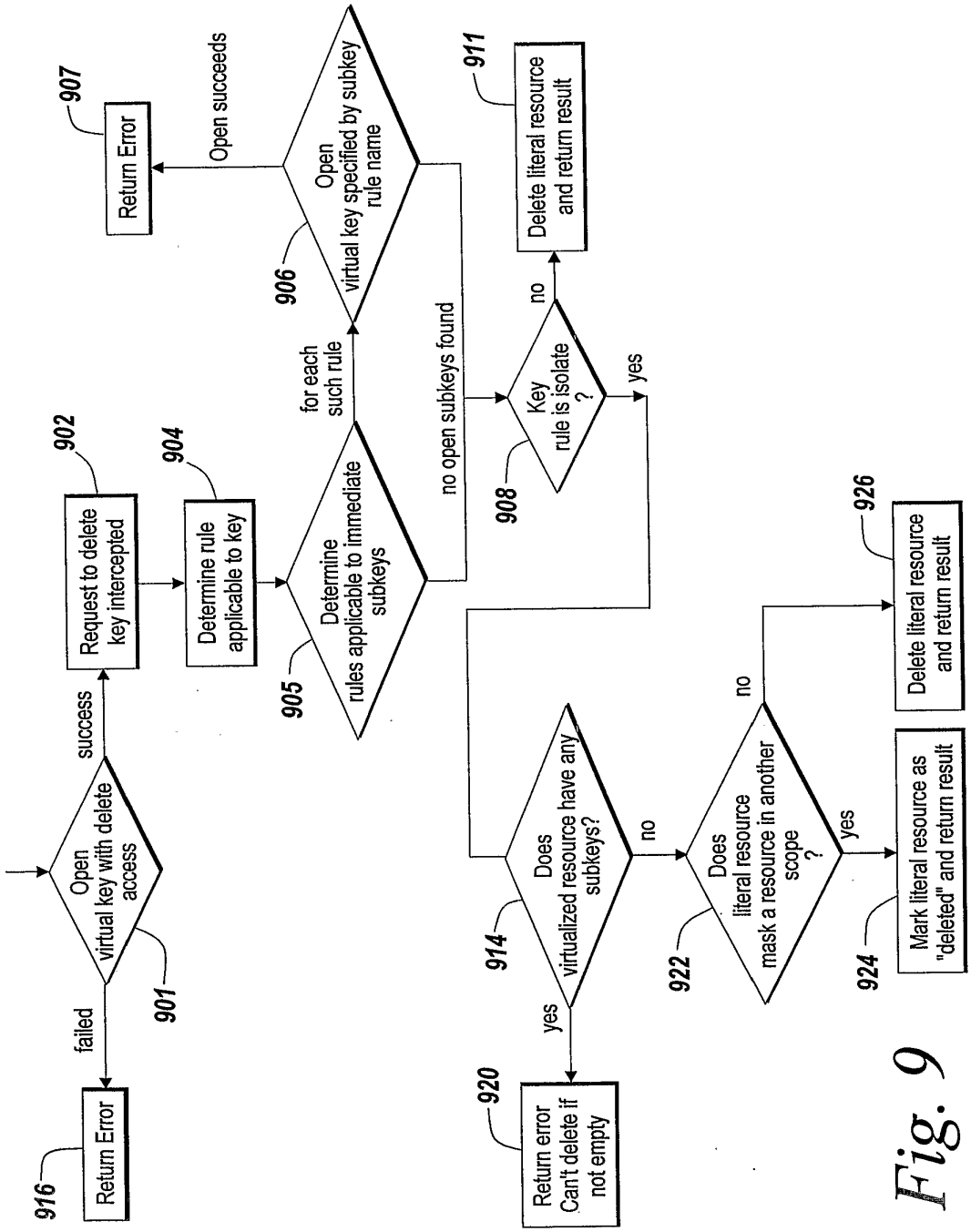

Open virtual key with delete access
901
failed
success
Return Error
916
Request to delete key intercepted
902
Determine rule applicable to key
904
Determine rules applicable to immediate subkeys
905
for each such rule
Open virtual key specified by subkey rule name
906
Open succeeds
Return Error
907
no open subkeys found
Key rule is isolate ?
908
no
yes
Delete literal resource and return result
911
Does virtualized resource have any subkeys?
914
yes
Return error Can't delete if not empty
920
no
Does literal resource mask a resource in another scope ?
922
yes
Mark literal resource as "deleted" and return result
924
no
Delete literal resource and return result
926

Fig. 9

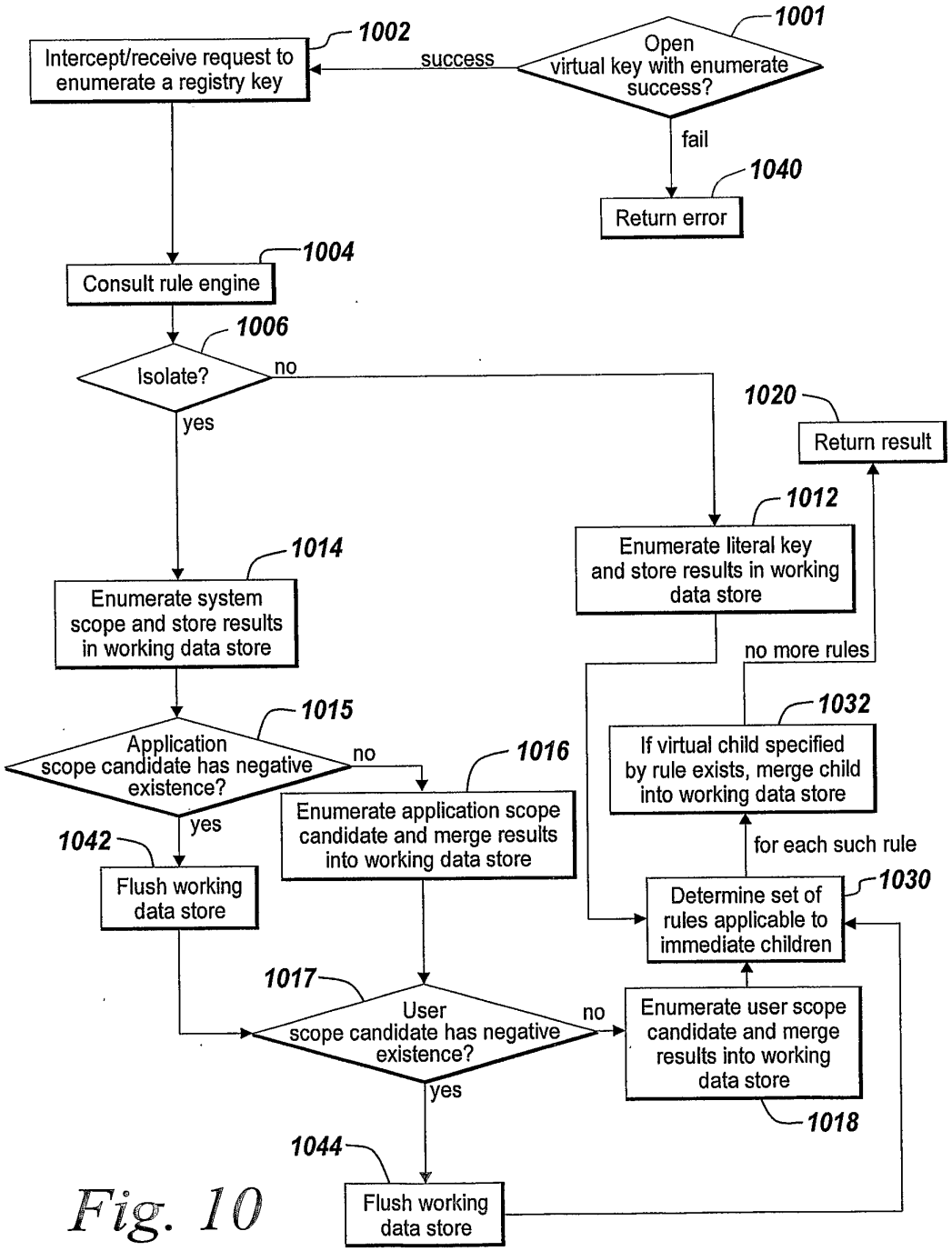

1001
Open virtual key with enumerate success?
success
1002
Intercept/receive request to enumerate a registry key
fail
1040
Return error
1004
Consult rule engine
1006
Isolate?
no
yes
1020
Return result
1012
Enumerate literal key and store results in working data store
1014
Enumerate system scope and store results in working data store
no more rules
1032
If virtual child specified by rule exists, merge child into working data store
1015
Application scope candidate has negative existence?
no
1016
Enumerate application scope candidate and merge results into working data store
yes
1042
Flush working data store
for each such rule
1030
Determine set of rules applicable to immediate children
1017
User scope candidate has negative existence?
no
Enumerate user scope candidate and merge results into working data store
1018
yes
1044
Flush working data store

Fig. 10

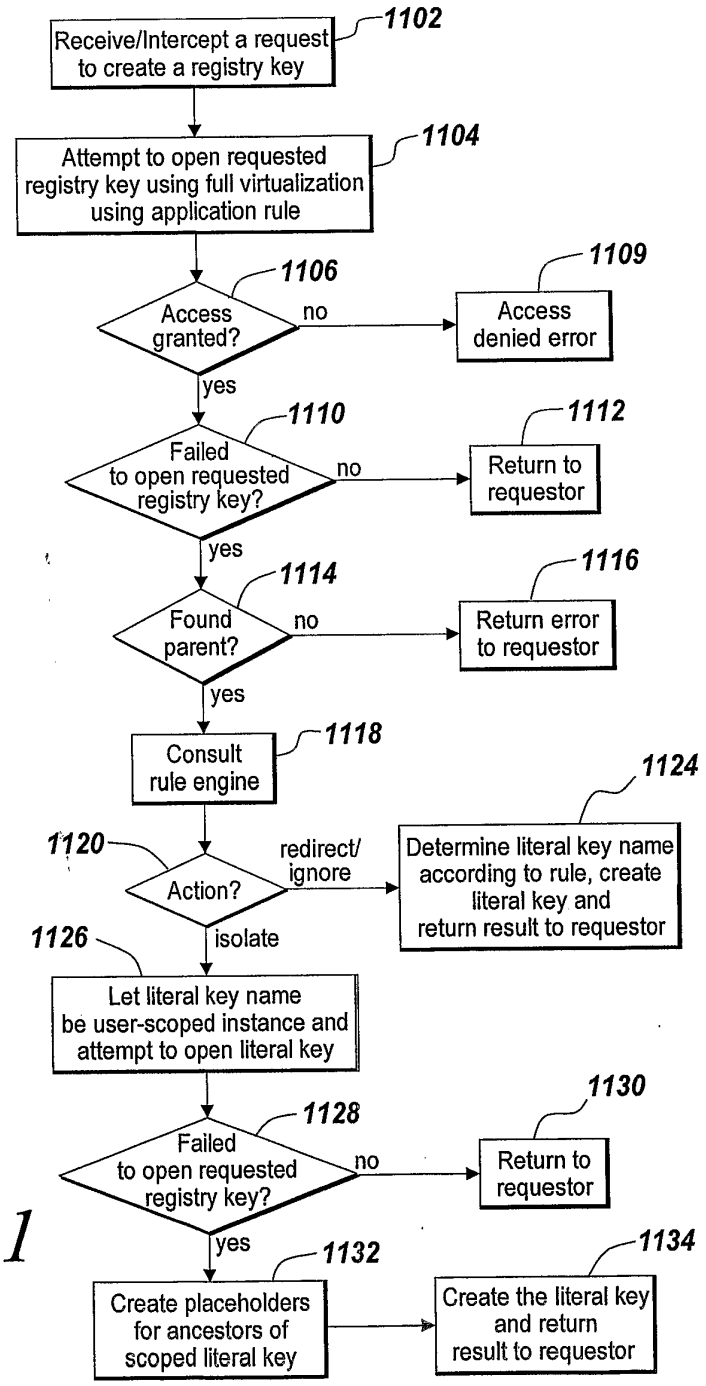

1102
Receive/Intercept a request to create a registry key
1104
Attempt to open requested registry key using full virtualization using application rule
1106
Access granted?
no
1109
Access denied error
yes
1110
Failed to open requested registry key?
no
1112
Return to requestor
yes
1114
Found parent?
no
1116
Return error to requestor
yes
1118
Consult rule engine
1120
Action?
redirect/ ignore
1124
Determine literal key name according to rule, create literal key and return result to requestor
isolate
1126
Let literal key name be user-scoped instance and attempt to open literal key
1128
Failed to open requested registry key?
no
1130
Return to requestor
yes
1132
Create placeholders for ancestors of scoped literal key
1134
Create the literal key and return result to requestor

Fig. 11

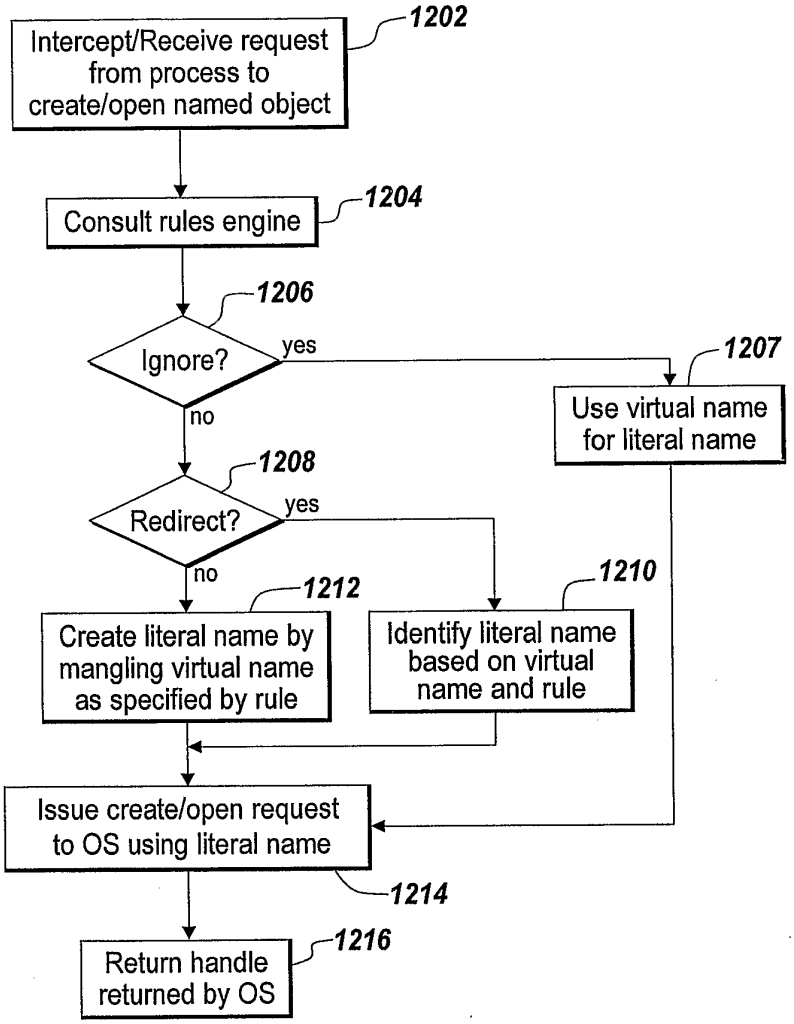

Intercept/Receive request from process to create/open named object
1202
Consult rules engine
1204
1206
Ignore?
yes
no
1207
Use virtual name for literal name
1208
Redirect?
yes
no
1212
1210
Create literal name by mangling virtual name as specified by rule
Identify literal name based on virtual name and rule
Issue create/open request to OS using literal name
1214
Return handle returned by OS
1216

Fig. 12

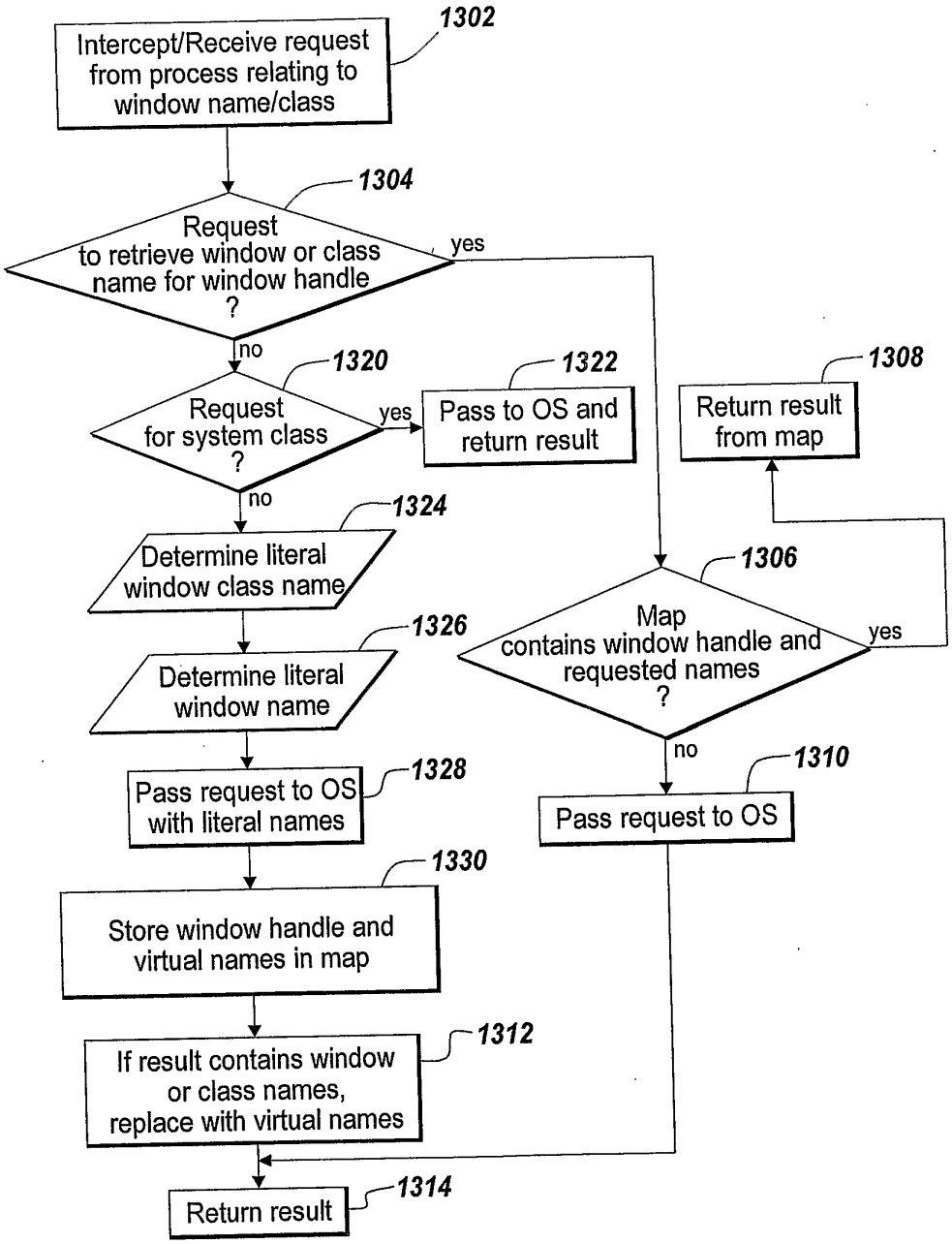

1302
Intercept/Receive request from process relating to window name/class
1304
Request to retrieve window or class name for window handle ?
yes
no
1320
Request for system class ?
yes
1322
Pass to OS and return result
no
1324
Determine literal window class name
1326
Determine literal window name
1328
Pass request to OS with literal names
1330
Store window handle and virtual names in map
1312
If result contains window or class names, replace with virtual names
1314
Return result
1306
Map contains window handle and requested names ?
yes
1308
Return result from map
no
1310
Pass request to OS

Fig. 13

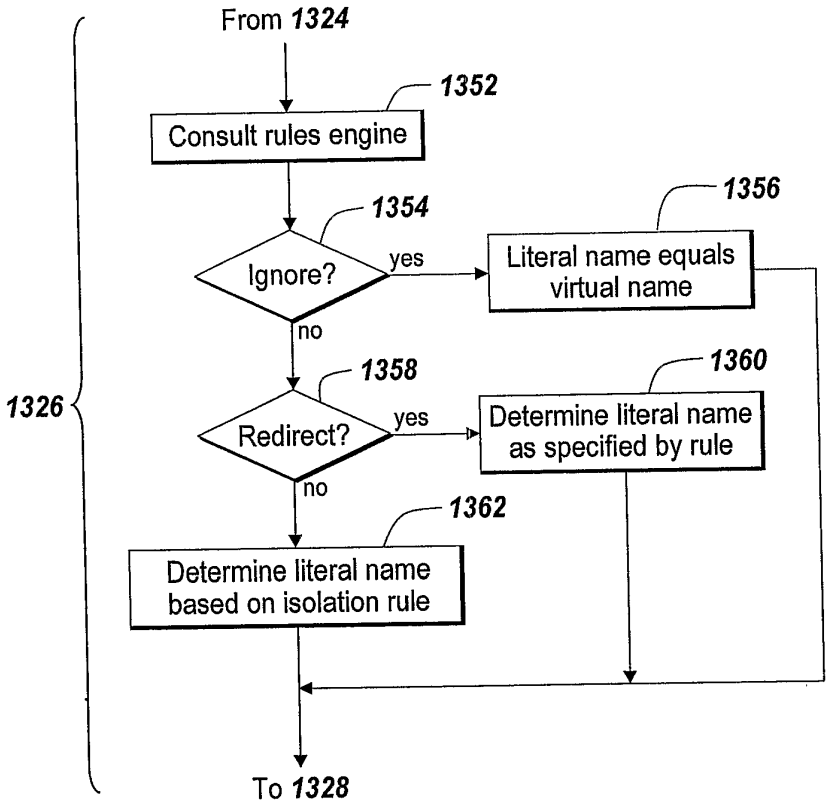

From 1324
1352
Consult rules engine
1354
Ignore?
yes
1356
Literal name equals virtual name
no
1358
Redirect?
yes
1360
Determine literal name as specified by rule
no
1362
Determine literal name based on isolation rule
1326
To 1328

Fig. 13A

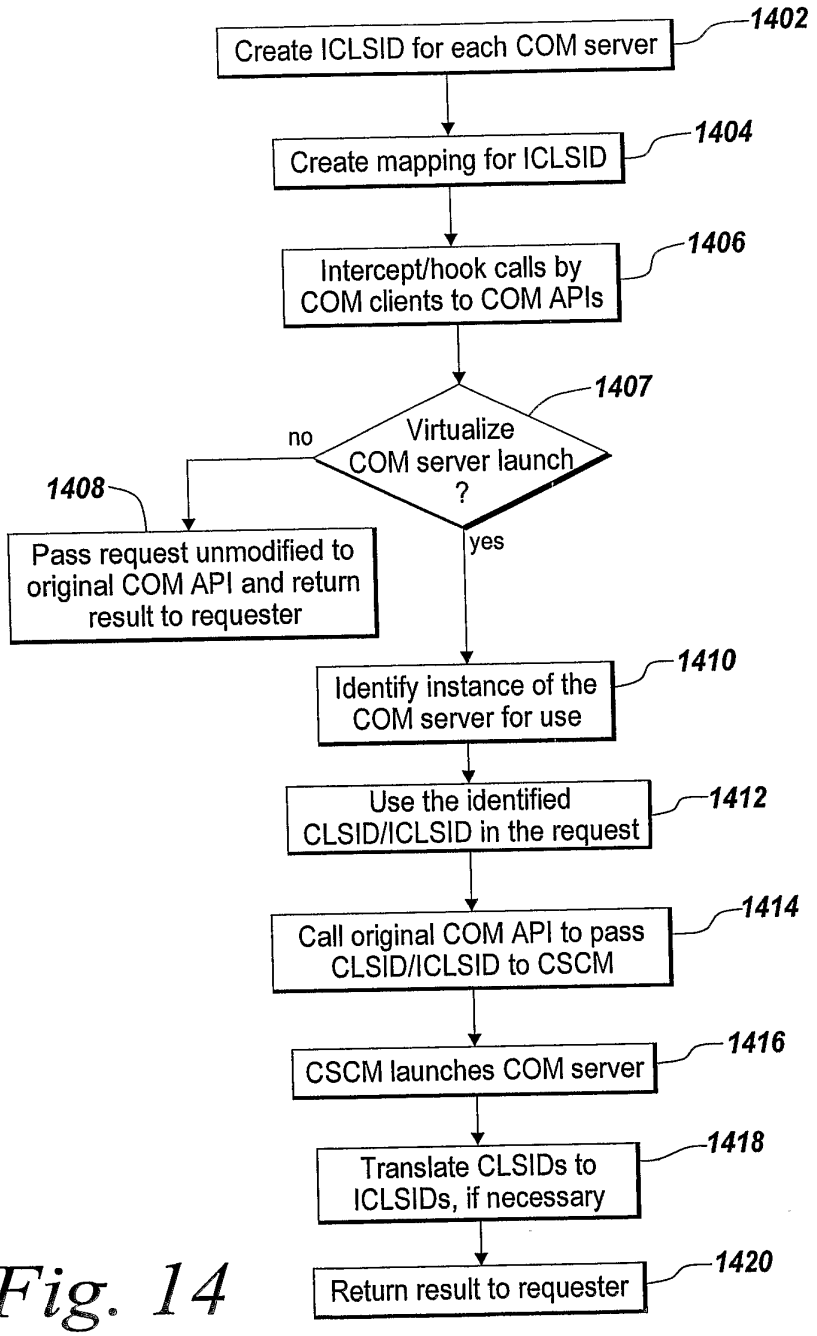

Create ICLSID for each COM server
1402
Create mapping for ICLSID
1404
Intercept/hook calls by COM clients to COM APIs
1406
Virtualize COM server launch ?
1407
no
yes
1408
Pass request unmodified to original COM API and return result to requester
Identify instance of the COM server for use
1410
Use the identified CLSID/ICLSID in the request
1412
Call original COM API to pass CLSID/ICLSID to CSCM
1414
CSCM launches COM server
1416
Translate CLSIDs to ICLSIDs, if necessary
1418
Return result to requester
1420

Fig. 14

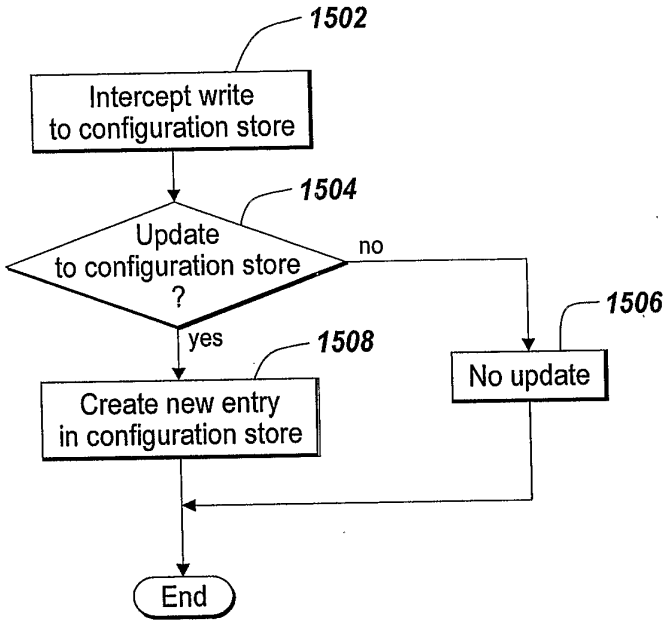

1502
Intercept write to configuration store
1504
Update to configuration store ?
no
yes
1506
No update
1508
Create new entry in configuration store
End

Fig. 15

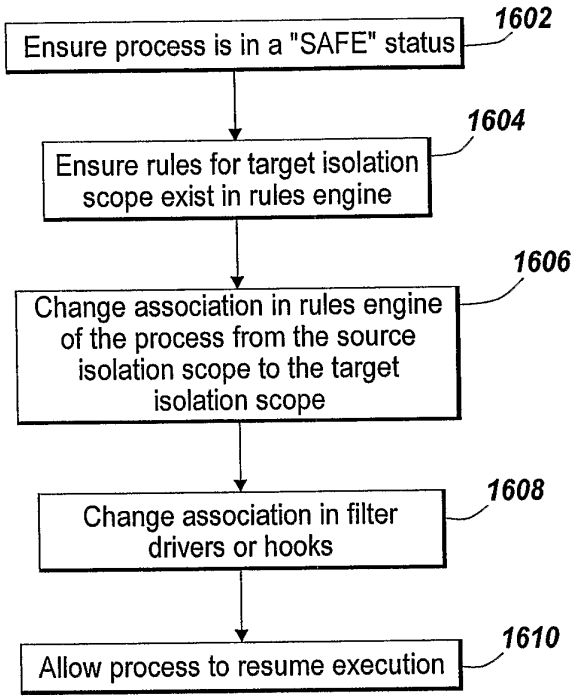

Ensure process is in a "SAFE" status
1602
Ensure rules for target isolation scope exist in rules engine
1604
Change association in rules engine of the process from the source isolation scope to the target isolation scope
1606
Change association in filter drivers or hooks
1608
Allow process to resume execution
1610

Fig. 16

INTERNATIONAL SEARCH REPORT

:ional application No
'US2005/033994

A. CLASSIFICATION OF SUBJECT MATTER
G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, PAJ, IBM-TDB, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 01/25894 A (EJASENT INC; HIPP, BURTON, A; BHARADHWAJ, RAJEEV) 12 April 2001 (2001-04-12) page 1, line 12 - page 3, line 10 page 9, line 16 - page 18, line 30	1-26
X	US 2003/101292 A1 (FISHER JOSEPH A ET AL) 29 May 2003 (2003-05-29) page 1, left-hand column, paragraph 1 - page 1, right-hand column, paragraph 10	1-26
A	WO 01/95094 A (SUN MICROSYSTEMS, INC) 13 December 2001 (2001-12-13) page 1, line 1 - page 7, line 3	1-26
A	WO 00/45262 A (SUN MICROSYSTEMS, INC) 3 August 2000 (2000-08-03) page 2, line 4 - page 6, line 11	1-26

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
7 February 2006	15/02/2006
Name and mailing address of the ISA/ European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Authorized officer Lo Turco, S

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

al application No
2005/033994

Patent document cited in search report		Publication date	Patent family member(s)			Publication date
WO 0125894	A	12-04-2001	AU	1074801	A	10-05-2001
			AU	1075101	A	10-05-2001
			AU	7864700	A	10-05-2001
			AU	7867000	A	10-05-2001
			AU	7867100	A	10-05-2001
			AU	7872100	A	10-05-2001
			AU	7996200	A	10-05-2001
			AU	8000800	A	10-05-2001
			WO	0125949	A1	12-04-2001
			WO	0126267	A1	12-04-2001
			WO	0125950	A1	12-04-2001
			WO	0125951	A1	12-04-2001
			WO	0126031	A2	12-04-2001
			WO	0125920	A1	12-04-2001
			WO	0125926	A1	12-04-2001
US 2003101292	A1	29-05-2003	NONE			
WO 0195094	A	13-12-2001	AU	6491401	A	17-12-2001
			EP	1299800	A2	09-04-2003
			US	6934755	B1	23-08-2005
WO 0045262	A	03-08-2000	AT	269558	T	15-07-2004
			AU	763958	B2	07-08-2003
			AU	4165700	A	18-08-2000
			CN	1338071	A	27-02-2002
			DE	1190316	T1	06-03-2003
			DE	60011615	D1	22-07-2004
			DE	60011615	T2	07-07-2005
			EP	1190316	A2	27-03-2002
			JP	2003518279	T	03-06-2003
			US	6907608	B1	14-06-2005
			US	2005102679	A1	12-05-2005

Form PCT/ISA/210 (patent family annex) (April 2005)

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 January 2002 (24.01.2002)

PCT

(10) International Publication Number
WO 02/06941 A2

(51) International Patent Classification[7]: G06F 3/00

(21) International Application Number: PCT/US01/22278

(22) International Filing Date: 16 July 2001 (16.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/617,669 17 July 2000 (17.07.2000) US

(71) Applicant: **CONNECTIX CORPORATION** [US/US]; 2955 Campus Drive, Suite 100, San Mateo, CA 94403 (US).

(72) Inventors: **TRAUT, Eric, P.**; 3 Iris Lane, San Carlos, CA 94070 (US). **MARTZ, Benjamin**; 258 W. 40th Ave., San Mateo, CA 94403 (US).

(74) Agents: **FULGHUM, Roger**; Baker Botts L.L.P., One Shell Plaza, 910 Louisiana, Houston, TX 77002 et al. (US).

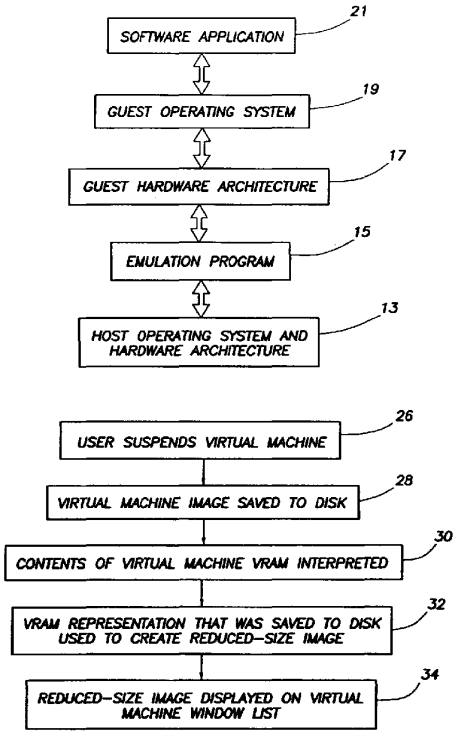
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR DISPLAYING CURRENT IMAGES OF VIRTUAL MACHINE ENVIRONMENTS



(57) Abstract: A system and method for displaying thumbnail images of the video output of one or more software applications in a window or similar graphical interface to allow the user of a computer system to conveniently and quickly monitor the overall status and progress of several software applications that are running simultaneously. The thumbnail images are generated from the VRAM associated with the software application and are preferably displayed with information corresponding to the associated software application. The thumbnail images may be static or generated at regular intervals according to user preference and the status of the software application.

WO 02/06941 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM AND METHOD FOR DISPLAYING CURRENT IMAGES OF VIRTUAL MACHINE ENVIRONMENTS

TECHNICAL FIELD OF THE INVENTION

The present disclosure relates in general to the field of virtual machines, and, more particularly, to a system and method for displaying images from virtual machine environments.

BACKGROUND OF THE INVENTION

Computers include general purpose central processing units (CPUs) that are designed to execute a specific set of system instructions. A group of processors that have similar architecture or design specifications may be considered to be members of the same processor family. Examples of current processor families include the Motorola 680X0 processor family, manufactured by Motorola, Inc. of Phoenix, Arizona; the Intel 80X86 processor family, manufactured by Intel Corporation of Sunnyvale, California; and the PowerPC processor family, which is manufactured by Motorola, Inc. and used in computers manufactured by Apple Computer, Inc. of Cupertino, California. Although a group of processors may be in the same family because of their similar architecture and design considerations, processors may vary widely within a family according to their clock speed and other performance parameters.

Each family of microprocessors executes instructions that are unique to the processor family. The collective set of instructions that a processor or family of processors can execute is known as the processor's instruction set. As an example, the instruction set used by the Intel 80X86 processor family is incompatible with the instruction set used by the PowerPC processor family. The Intel 80X86 instruction set is based on the Complex Instruction Set Computer (CISC) format. The Motorola PowerPC instruction set is based on the Reduced Instruction Set Computer (RISC) format. CISC processors use a large number of instructions, some of which can perform rather complicated functions, but which require generally many clock cycles to execute. RISC processors use a smaller number of available instructions to perform a simpler set of functions that are executed at a much higher rate.

The uniqueness of the processor family among computer systems also typically results in incompatibility among the other elements of hardware architecture of the computer systems. A computer system manufactured with a processor from the Intel 80X86 processor family will have a hardware architecture that is different from the hardware architecture of a computer system manufactured with a processor from the PowerPC processor family. Because of the uniqueness of the processor instruction set and a computer system's hardware architecture, application software programs are typically written to run on a particular computer system running a particular operating system.

A computer manufacturer will want to maximize its market share by having more rather than fewer applications run on the microprocessor family associated with the computer manufacturer's product line. To expand the number of operating systems and application programs that can run on a computer system, a field of technology has developed in which a given computer having one type of CPU, called a host, will include an emulator program that allows the host computer to emulate the instructions of an unrelated type of CPU, called a guest. Thus, the host computer will execute an application that will cause one or more host instructions to be called in response to a given guest instruction. Thus, the host computer can both run software design for its own hardware architecture and software written for computers having an unrelated hardware architecture. As a more specific example, a computer system manufactured by Apple Computer, for example, may run operating systems and program written for PC-based computer systems. It may also be possible to use an emulator program to operate concurrently on a single CPU multiple incompatible operating systems. In this arrangement, although each operating system is incompatible with the other, an emulator program can host one of the two operating systems, allowing the otherwise incompatible operating systems to run concurrently on the same computer system.

When a guest computer system is emulated on a host computer system, the guest computer system is said to be a virtual machine, as the host computer system exists only as a software representation of the operation of the hardware architecture of the guest computer system. The terms emulator and virtual machine are sometimes used interchangeably to denote the ability to mimic or emulate the hardware architecture of an entire computer system. As an example, the Virtual PC software

created by Connectix Corporation of San Mateo, California emulates an entire computer that includes an Intel 80X86 Pentium processor and various motherboard components and cards. The operation of these components is emulated in the virtual machine that is being run on the host machine. An emulator program executing on the operating system software and hardware architecture of the host computer, such as a computer system having a PowerPC processor, mimics the operation of the entire guest computer system. The emulator program acts as the interchange between the hardware architecture of the host machine and the instructions transmitted by the software running within the emulated environment.

Over the years, the number of operating systems able to execute on a given processor family has increased markedly. For example, at present, the Windows operating system alone has several versions, such as Windows 3.1, Windows 95, Windows 98, Windows 98SE, Windows 2000, Windows NT, and Windows Millennium. Thus, a user of a computer system having an Intel 80X86 processor architecture may choose among several operating systems. Similarly, if the hardware architecture of a computer system running an Intel 80X86 architecture is being emulated in a host computer system, it may be desirable to run several virtual machines simultaneously, with each virtual machine operating according to a different operating system. As an example, it may be desirable to emulate on a Macintosh computer system several guest computers systems, each running a separate version of Windows. By doing so, the Macintosh user can take advantage of the wide variety of software applications designed to run on PC computer systems.

As the user increases the number of software applications and operating systems that is running on a computer system, it becomes more difficult for the user to manage the various tasks performed on the computer system. Accordingly, as the user increases the number of virtual machines running on the computer system, it becomes more difficult for the user to keep track of both the virtual machines and the respective applications running on the computer system's native operating system and the various virtual machines. For example, in operating systems that use windowing environments, such as Windows, the Macintosh operating system, and OS/2, for example, users may resize or move windows that correspond to various applications. Windows that are minimized or suspended are typically moved to a task bar, application bar or

application list where they can be later accessed. Unfortunately, these application bars or lists generally do not present the user with enough information about the particular applications contained therein to allow the user to quickly find the particular application the user is interested in resuming. This problem is exacerbated as the number of applications and corresponding windows increases.

SUMMARY OF THE INVENTION

In accordance with teachings of the present disclosure, a system and method for managing multiple virtual computer environments are disclosed that provide significant advantages over prior developed systems.

The system and method described herein allow a user of a host computer system running multiple virtual machine environments to view the thumbnail images of the video output of each virtual machine application. According to one embodiment of the present invention, the thumbnail images are generated by interpreting the contents of the virtual machine's video RAM (VRAM) and scaling the resulting representations into several thumbnail images that may be conveniently viewed, for example, in a single window. Interpretation of the virtual machine's VRAM is generally dependent on the current video mode of the virtual machine, such as text, planar video, linear video, among other examples. According to one embodiment of the present invention, the thumbnail images are generated using bilinear sampling techniques to create an anti-aliased miniature image. In the case of active emulated computer systems, thumbnail images are preferably continuously generated to give a real-time representation of the virtual machine application within the thumbnail view. The thumbnail or reduced-size images are preferably saved whenever the respective virtual machine is placed in a suspension mode and accordingly saved to the storage device of the computer system running the virtual machine software. The image of these suspended images are likewise displayed for the user, allowing the user to quickly view the status of the virtual machine regardless of whether the virtual machine is active or suspended.

The disclosed system and method provide several technical advantages over conventional systems and methods for managing multiple computer applications, such as multiple virtual computer environments. One advantage of the system and method

of the present invention is that it allows a user of a computer system that is running multiple virtual machine environments to view the status of several multiple virtual machine environments at one glance. This allows the user to easily manage several virtual machine environments that are running simultaneously. Another advantage of the present system and method is that the user may quickly ascertain the status of a given virtual machine without having to activate or unsuspend the virtual machine. This allows the user to save time, especially if the virtual machine is running at a relatively slow speed, or if the user is working on several projects.

Other technical advantages should be apparent to one of ordinary skill in the art in view of the specification, claims, and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the present embodiments and advantages thereof may be acquired by referring to the following description taken in conjunction with the accompanying drawings, in which like reference numbers indicate like features, and wherein:

Figure 1 is a diagram of the logical relationship of the elements of an emulated computer system running in a host computer system;

Figure 2 is diagram depicting the screen display from a computer system utilizing an embodiment of the present invention;

Figure 3 is a flow diagram of method for displaying a thumbnail image of an emulated computer system; and

Figure 4 is a flow diagram of the manner in which the emulator program polls each virtual machine environment to determine the status of the virtual machine and to locate the reduced-size image to be displayed for each virtual machine

DETAILED DESCRIPTION OF THE INVENTION

Shown in Figure 1 is a diagram of the logical layers of the hardware and software architecture for an emulated operating environment in a computer system 11. An emulation program 15 runs on a host operating system and/or hardware architecture 13. Emulation program 15 emulates a guest hardware architecture 16 and a guest operating system 19. Software application 21 in turn runs on guest operating system

19. In the emulated operating environment of Figure 1, because of the operation of emulation program 15, software application 21 can run on the computer system 11 even though software application 21 is designed to run on an operating system that is generally incompatible with the host operating system and hardware architecture 13.

Shown in Figure 2 is a screen display 10 from a computer system 12 that implements the system and method of the present invention. Computer system 12 may use any type of processor and any type of operating system that are suitable for running one or more virtual machines. For example, computer system 12 may include a processor from one of several processor families such as the PowerPC processor, the Intel 80X86 Pentium processor, or the Motorola 680X0 processor, among other examples. The operating system of computer system 12 is preferably an operating system that may use a windowing environment. Examples of suitable operating systems include Windows, the Macintosh OS, or OS/2, among other examples. For example, the computer system 12 depicted in Figure 2 uses the Macintosh OS and a PowerPC processor.

Computer system 12 is running one or more virtual machines or emulators on its native or host operating system. As discussed above, an emulator or virtual machine allows a host computer to run a software application that is designed for an operating system or processor that is not native to the host computer. Thus, computer system 12 is a host computer system. If the native operating system for computer system 12 uses a windowing environment or other similar graphical user interface that represents programs, files, and options by means of icons, menus, and dialog boxes on the screen, then the virtual machines or emulators may be displayed in a virtual machine list window 14 that can be displayed on the desktop 16 of the operating system of computer system 12. From the virtual machine list window 14, the user of the computer system may monitor or modify the function of the virtual machines or emulators running on computer system 12. The user may modify the function of the virtual machines or emulators using control buttons 18 or any similar graphical interface functionality. For example, the user may choose additional virtual machines or emulators to run, choose new software applications to run on virtual machines or emulators that are already running, modify the settings associated with any of the virtual machines or emulators,

terminate any of the virtual machines or emulators, or suspend any of the virtual machines or emulators, among other examples.

One or more virtual machine status windows are displayed within virtual machine list window 14. Shown in list window 14 are virtual machine status windows 20a-20i. Each virtual machine status window is preferably associated with one of the virtual machines running on computer system 12. Alternatively, if a particular virtual machine is running more than one software application, then a virtual machine status window can be associated with each software application. Each virtual machine status window displays information regarding the associated virtual machine or emulator. For example, the virtual machine status window may display information regarding the operating system or computer system that is being emulated, the software application that is running under the associated virtual machine or emulator, and the current status of the software application, among other examples.

In addition, each virtual machine status window also contains a thumbnail image. Shown in list window are thumbnail images 22a-22i, each of which is associated with a virtual machine status window 20a-20i. Each thumbnail image is a representation of the image or graphical output that is currently being generated by the software application or operating system running under the virtual machine associated with the respective virtual machine status window. If the software application is currently suspended or saved, then the associated thumbnail image is the image that was saved in VRAM of the virtual machine at the time that the virtual machine environment was suspended. Each thumbnail image is preferably sized to allow the user to conveniently view all or several of the virtual machine status windows 20 on the computer screen of computer system 12. If the operating system of the virtual machines displays images in color, then thumbnail image is preferably also displayed in color. As discussed below, thumbnail image may be a static image or an image that is updated in real-time or periodically.

The computer system 12 illustrated in Figure 2 is using the Macintosh OS and is running through an emulator program several virtual PC machines on the host Macintosh OS. The virtual machine status windows 20a-20i in Figure 2 display information associated with the various virtual PC machines that are operating on computer system 12. Virtual machine status window 20a indicates that computer

system 12 includes a virtual machine that is emulating a computer system with a DOS operating system. Virtual machine status window 20a also indicates that the operation of a video game software application is associated with this virtual machine. In addition, virtual machine status window 20a indicates that the operation of this video game is currently saved or suspended. Because the software application is suspended, thumbnail image 22a is a representation of the image that was in VRAM of the virtual machine at the time that the virtual machine was suspended. Similarly, virtual machine status windows 20b through 20i indicate that computer system 12 also includes virtual machines that emulate computer systems running Windows 98, Windows 2000, Windows 3.1, Windows 95, Windows 98SE, Windows Millennium, and Windows NT operating systems. Thumbnail images 20b through 20i are representations of the images that were created by the operating systems or software applications associated with virtual machine status windows 20b through 20i. The term Windows is a trademark of Microsoft Corporation.

Generally, each software application running on an operating system implementing a windowing environment will be running in its own window. Therefore, the other active software applications of the computer system 12 will be running in one of windows 24. Preferably, the user does not operate the software application depicted in the virtual machine list window 14 directly from the virtual machine list window 14. Thus, if software application depicted in virtual machine list 14 is running, it will be running on an active window 24 in addition to being depicted in virtual machine window list 14. However, the user may preferably be able to restore or maximize the window 24 associated with a software application running under a virtual machine or emulator application by clicking on, or otherwise activating, the virtual machine status window 22 associated with that software application and the corresponding virtual machine or emulator.

Figure 3 depicts a process diagram for suspending the contents of a virtual machine and later displaying those contents in a reduced-size representation for the user. As discussed above, the user of computer system 12 may have several virtual machines or emulated operating system environments running at the same time. Due to resource limitations, a computer system 12 running several virtual machine applications simultaneously may incur adverse performance effects. Therefore, it may be desirable

for a user of computer 12 to suspend a particular virtual machine or operating system environment. At step 26 of Figure 3, the user of computer system 12 suspends a virtual machine application. By temporarily halting the virtual machine, the user can conserve resources and improve the management or performance of other applications. When a virtual machine application is suspended, the operating parameters associated with the virtual machine are stored, so that the operation of the virtual machine can be quickly resumed if the user of computer system 12 decides to activate or unsuspend the virtual machine. Thus at step 28, the contents of the RAM, VRAM, registers, parameters and other data associated with the virtual machine application are saved to a storage device of computer system 12. Typically, the storage device will be a hard drive, disk, or some other nonvolatile storage medium. For instance, if the virtual machine is emulating a processor, network card and video card of a particular hardware architecture, then the registers for these various components must be saved to memory as part of the suspension process. Saving the contents of VRAM of the virtual machine saves the video image most recently displayed by the virtual machine. Similarly, various parameters for these components must be saved, such as the video mode and resolution for the video card, for example. The step of saving the virtual machine environment to a storage device is preferably performed with a compression algorithm or similar functionality to reduce the storage space or time required for this step.

To successfully manage a computer system 12 with several virtual machines, the user must be able to quickly ascertain the status of each virtual machine, such as whether the virtual machine is running a software application, and whether the virtual machine is active or suspended. In the case of suspended virtual machines, after the suspended virtual machine environment has been saved to the storage device, the thumbnail image 22 and other information must be generated for the virtual machine status window 20. To generate thumbnail image 22, the contents of the virtual machine's VRAM, which have been stored to memory, must be interpreted at step 30. Interpreting the virtual machine's VRAM involves taking into account the video mode or video adapter setting of the virtual machine at the time it was suspended. The video mode is the manner in which the virtual machine's video adapter displays on-screen images to the monitor of computer system 12. The most common video modes are text mode and graphics mode. In text mode, only characters such as letters, numbers, and

symbols may be displayed, and these characters are not drawn as pixel representations. On the other hand, graphics mode produces all monitor images, whether text or art, as patterns of pixels. Examples of graphics modes include planar video and linear video, among other examples. Another consideration for interpreting the VRAM is the associated bit depth or color depth. Because color in a computer image is a function of the number of bits available to provide different shades for each pixel, bit depth is a measure of the number of different colors that may be displayed in the image. Other graphical settings and parameters may need to be processed or taken in account as well. In the case of active virtual machines, the contents of the VRAM for active virtual machines is saved in the main memory of the host system. To display a reduced-size image of the active virtual machine, the contents of the main memory of the host system must be likewise interpreted.

With reference to Figure 3, once the contents of the virtual VRAM have been interpreted, the resulting representation is used to create thumbnail image 22 at step 32. At this point, the contents of the virtual VRAM have been saved to disk, and this saved image from disk is the image that is used to create the thumbnail image 22. Generally thumbnail images 22 are preferably created using bilinear sampling techniques to create an anti-aliased miniature image. Anti-aliasing, or oversampling, is a graphics technique that smoothes out the jagged appearance of curved or diagonal lines, a phenomenon known as "jaggies." Examples of anti-aliasing techniques include surrounding selected pixels in the image with intermediate shades and manipulating the size and alignment of the pixels. An anti-aliased image is preferred because the thumbnail image 22 is preferably small and the presence of jaggies or other visual distortions will typically result in a poor image. A bilinear sampling technique is used to downsample the source image on the VRAM by averaging the pixels in the source image to create a scaled-down thumbnail image 22. Another technique that may be used to create thumbnail image 22 is spot or point sampling to map a pixel in the source image to a pixel in the thumbnail image 22. Unfortunately, point sampling tends to result in a significant loss of data. Other techniques suitable for creating a scaled-down image may be used, such as area sampling and gaussian filtering, among other examples. The thumbnail image 22 is then displayed in the virtual machine list window 14 at step 34. Preferably, thumbnail image 22 is saved to the storage device along with the virtual

machine that is suspended so that the thumbnail image 22 may be quickly displayed at a later time, as the image does not need to be created again from the VRAM.

In the case of active virtual machines, a user of computer system 12 may wish to keep several virtual machine applications active and monitor their overall performance from time to time while directly operating the virtual machine application the user has chosen as a primary action item. For example, the user of computer system 12 may be running a virtual machine that is running a word processor software application for the purposes of creating a document and at least one other virtual machine that is running computer-aided design software applications (CAD). While the process of creating a document requires continuous user input to the word processor software application, the CAD applications do not necessarily require continuous user input if, for example, the CAD applications are rendering wire-frame skeleton models. However, the user may be interested in monitoring the progress of the CAD applications from time to time, but preferably only at a cursory level, as the user is only interested in gaining an overall impression of the rendering job before returning his attention to the word processor application. Therefore, the technique of the present invention allows the user, in the case of active virtual machines, to monitor a thumbnail image of the active virtual machines on a real time basis.

Shown in Figure 4 is a flow diagram of the manner in which the emulator program polls each virtual machine environment to determine the status of the virtual machine and to locate the reduced-size image to be displayed for each virtual machine. At step 36, the emulator routine scans through each emulated system. For each system, the emulator program performs the decision of step 38 and then steps through the remainder of the flow diagram of Figure 4. If the virtual machine is determined to be active at step 38, then at step 40 a reduced-size image of the VRAM of the virtual machine is interpreted by the emulator program and displayed for the user. In the case of active emulated systems, the contents of their VRAM are maintained by the host system in the main memory of the host system. If it is determined at step 38, that the virtual machine is not active, processing continues at step 42, where it is determined whether the virtual machine has been suspended, *i.e.*, whether the image contents of the virtual machine have been saved to disk. If the image contents of the virtual machine have been suspended to disk, the emulator program at step 44 loads from disk and

displays a thumbnail image of the image of the virtual machine at the time of its suspension. If it is determined at step 42 that the virtual machine is not suspended, then at step 46 a blank reduced-size image is loaded and displayed for the user. After each of steps 40, 42, or 46, the processing continues at step 36 with a scan of each virtual machine. The processing steps of Figure 4 permit the emulator program to continually monitor the status of each virtual machine. If a virtual machine transitions from suspended mode to active mode, for example, the emulator program is able to monitor this transition and display the correct image of the virtual machine. Additionally, for those virtual machines that are active, the processing steps of Figure 4 permit the emulator to display for the user a continually updated rendering of the image of the active virtual machine. The processing steps of Figure 4 may occur roughly once per second. At this processing rate, a continually update image of each active machine is displayed for the user to permit the user to monitor the state of each active virtual machine.

Although the disclosed embodiments have been discussed in reference to the field of virtual machines and emulators, it can be readily appreciated that the present invention has equal applicability to other types of software applications. For example, the user of computer system 12 may be running several software applications simultaneously and would like to monitor the status of all of the applications in a convenient manner. The virtual machine list window 14 could alternatively display all of the software applications that are currently active or otherwise on the desktop 16. Each software application would be associated with a virtual machine status window 20 that would provide information on the software application and its status. A thumbnail image 22 would be generated for each virtual machine status window 20 that would represent the video or graphical output of the software application. As discussed above, the thumbnail image 22 could be static or continuously updated depending on the user's preference and the status of the associated software application.

It should be noted that the choice of computer system that can be emulated according to the technique of the present invention is not limited to computers systems that operate according to the Intel 80X86 system architecture and run a Windows operating system. The invention described herein applies to any emulated environment in which one or more virtual machines can be displayed in a windowed setting. Thus,

the invention disclosed herein can be used in any environment in which the contents of the virtual video memory buffer may be displayed for the user. The method disclosed herein is especially advantageous in that it permits a user to monitor the activities or suspended state of one or more virtual machine environments. Thus, the user, when operating in a first virtual machine environment can readily determine the operating state or suspended condition of one or more other virtual machine environments that typically comprise unique operating system environments. The reduced-size representations of the virtual machine environments are rendered on the basis of the contents of the saved virtual VRAM of each environment. Thus, from a location in main memory, the emulator program can create for the user an easily accessible visual directory of the other virtual machine environments being emulated by the host system.

Although the disclosed embodiments have been described in detail, it should be understood that various changes, substitutions, and alterations can be made to the embodiments without departing from their spirited scope.

WHAT IS CLAIMED IS:

1. A computer system for running one or more software applications, wherein the software applications are suitable for generating a video output, comprising:

a host operating system suitable for displaying a graphical user interface;

multiple emulated operating systems being emulated by one or more emulator programs running on the host operating system; and

wherein the host operating system is able to display for a user a reduced-size representation of the video output of the emulated operating systems that are being operated in a background mode.

2. The computer system of claim 1, further comprising one or more virtual video memory components suitable for storing the video output of the emulated operating systems.

3. The computer system of claim 2, wherein one or more of the video memory components are VRAM memory.

4. The computer system of claim 1,

wherein the emulated operating systems operating in a background mode are active; and

wherein the thumbnail images for the emulated operating systems are generated from the video information stored on the video memory components at predetermined intervals while the software applications are active.

5. The computer system of claim 4, wherein the predetermined intervals are such that the thumbnail images are real-time representations of the video output from the active software applications.

6. The computer system of claim 1,

wherein the graphical user interface is a windowing environment suitable for displaying one or more windows; and

wherein the portion of the graphical user interface comprising the reduced-size representation is a window.

7. The computer system of claim 1, wherein the reduced-size representations are created using a bilinear sampling technique.

8. A computer system for running one or more software applications, wherein the software applications are suitable for generating a video output, comprising:

a host operating system suitable for displaying a graphical user interface;

multiple emulated virtual machines being emulated by one or more emulator programs running on the host operating system; and

wherein the host operating system is able to display for a user a reduced-size representation of the video output of each virtual machine being operated in a background mode.

9. The computer system of claim 8, wherein the reduced-size representations are representations of the video outputs of the virtual machines that are being operated in the background mode.

10. The computer system of claim 9,

further comprising a virtual video memory associated with each of the virtual machines; and

wherein the reduced-size representations are generated from the video information stored in the virtual video memory associated with each virtual machine.

11. A method for displaying a reduced-size image of multiple emulated computer systems, comprising the steps of:

suspending one or more of the multiple emulated computer systems by saving to memory in the host computer system the image of the emulated computer system;

reading in at the emulator program from memory in the host computer system the image of the suspended emulated computer system;

interpreting in the emulator program the contents of the saved image of the suspended emulated computer system;

displaying a reduced-size representation of the suspended emulated computer system.

12. A method for displaying a reduced-size image of multiple emulated computer systems, comprising the steps of:

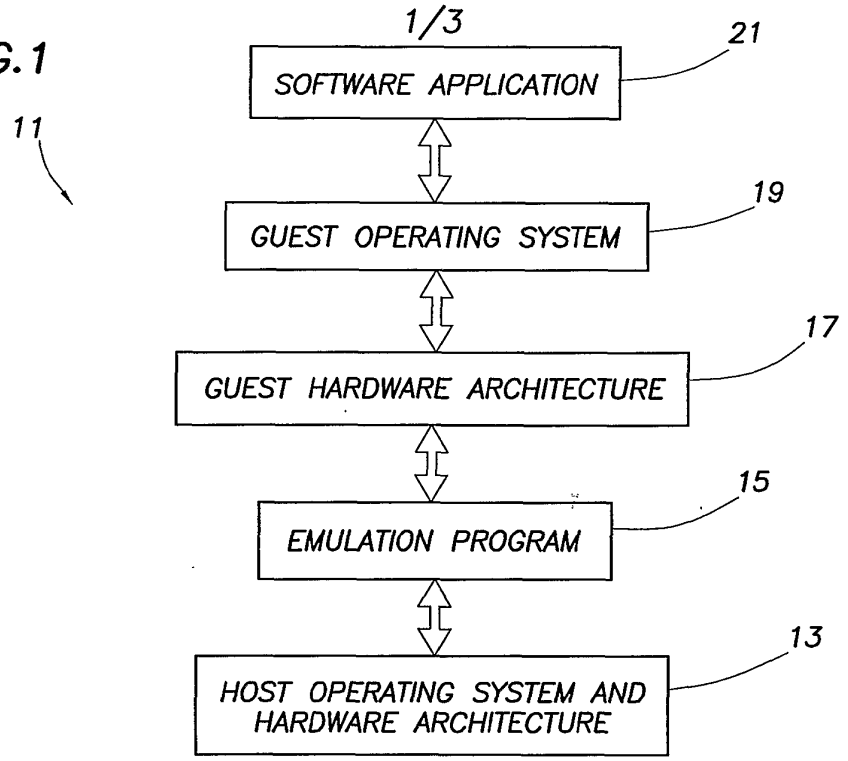
reading in at the emulator program from memory in the host computer system the image of the emulated computer system;

interpreting in the emulator program the contents of the image of the emulated computer system;

displaying a reduced-size representation of the emulated computer system;

periodically updating the reduced-size representation of the emulated computer system.

FIG. 1


11
SOFTWARE APPLICATION
21
GUEST OPERATING SYSTEM
19
GUEST HARDWARE ARCHITECTURE
17
EMULATION PROGRAM
15
HOST OPERATING SYSTEM AND HARDWARE ARCHITECTURE
13

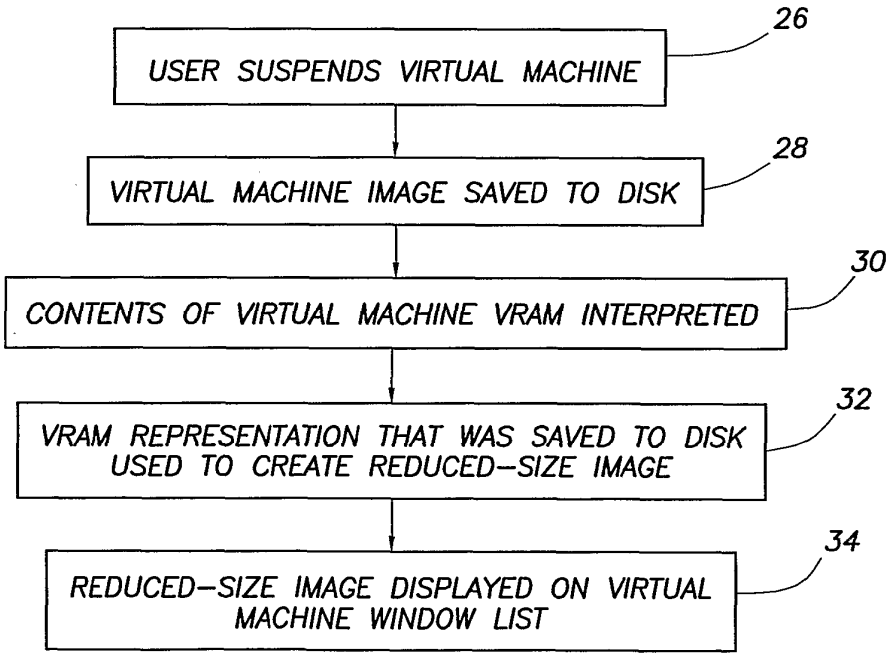

USER SUSPENDS VIRTUAL MACHINE
26
VIRTUAL MACHINE IMAGE SAVED TO DISK
28
CONTENTS OF VIRTUAL MACHINE VRAM INTERPRETED
30
VRAM REPRESENTATION THAT WAS SAVED TO DISK USED TO CREATE REDUCED-SIZE IMAGE
32
REDUCED-SIZE IMAGE DISPLAYED ON VIRTUAL MACHINE WINDOW LIST
34

FIG. 3

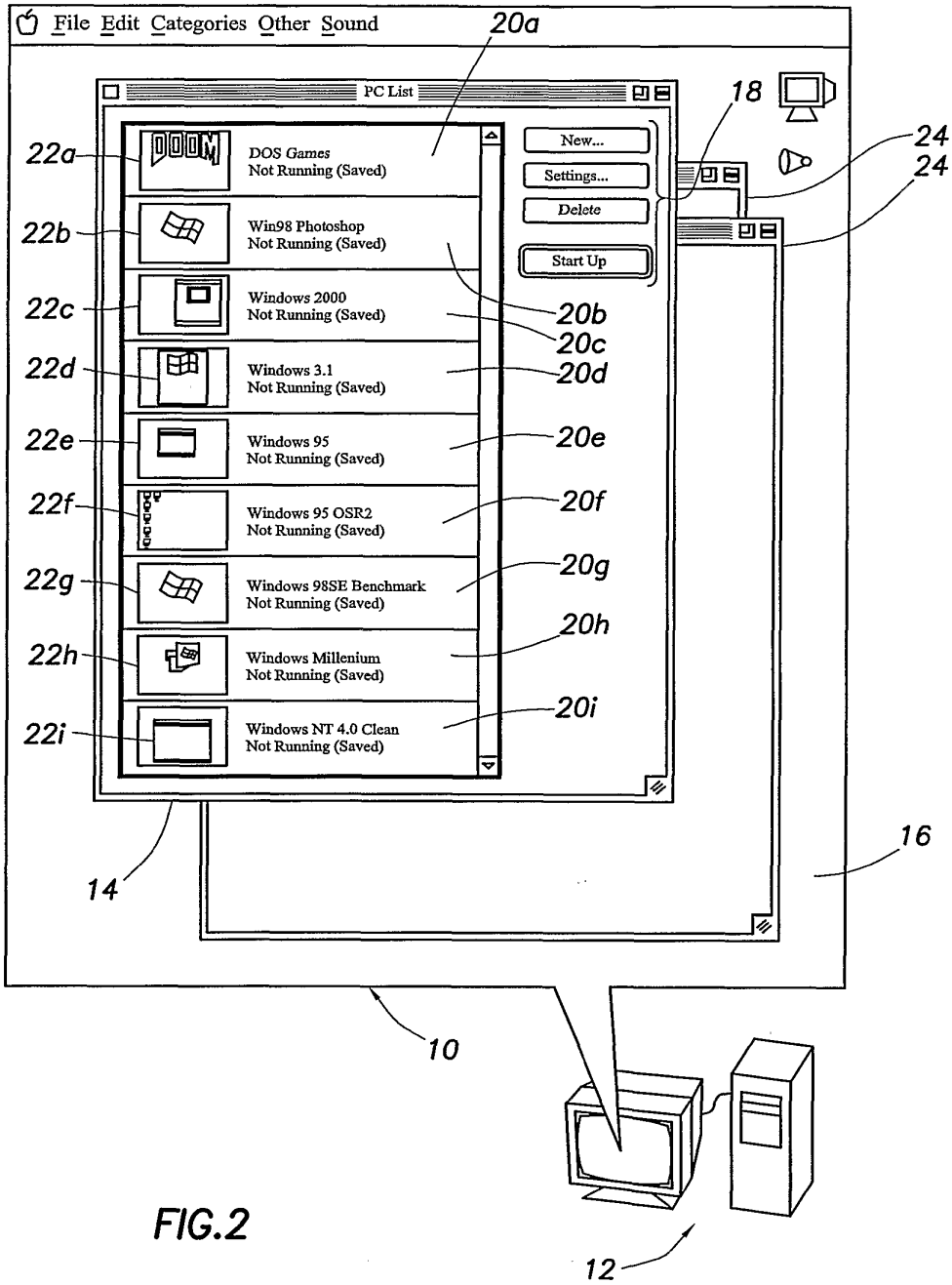

File Edit Categories Other Sound
20a
PC List
18
New...
Settings...
Delete
Start Up
24
24
22a
DOS Games
Not Running (Saved)
22b
Win98 Photoshop
Not Running (Saved)
22c
Windows 2000
Not Running (Saved)
20b
20c
22d
Windows 3.1
Not Running (Saved)
20d
22e
Windows 95
Not Running (Saved)
20e
22f
Windows 95 OSR2
Not Running (Saved)
20f
22g
Windows 98SE Benchmark
Not Running (Saved)
20g
22h
Windows Millenium
Not Running (Saved)
20h
22i
Windows NT 4.0 Clean
Not Running (Saved)
20i
14
16
10
12

FIG.2

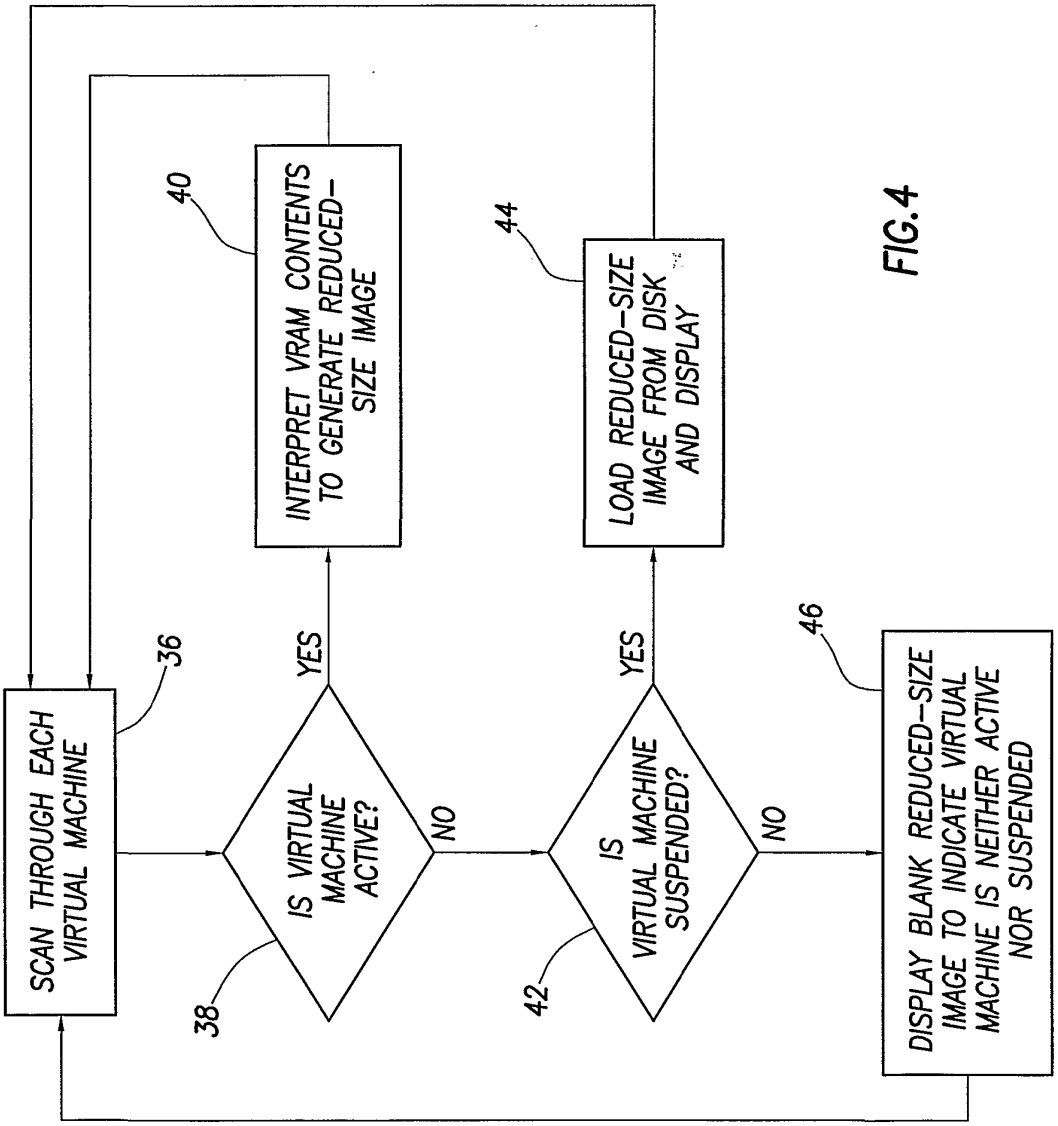

SCAN THROUGH EACH VIRTUAL MACHINE
36
IS VIRTUAL MACHINE ACTIVE?
38
YES
NO
INTERPRET VRAM CONTENTS TO GENERATE REDUCED-SIZE IMAGE
40
IS VIRTUAL MACHINE SUSPENDED?
42
YES
NO
LOAD REDUCED-SIZE IMAGE FROM DISK AND DISPLAY
44
DISPLAY BLANK REDUCED-SIZE IMAGE TO INDICATE VIRTUAL MACHINE IS NEITHER ACTIVE NOR SUSPENDED
46

FIG.4

PLUS Search Results for S/N 10946536, Searched Fri Feb 29 10:27:30 EST 2008

The Patent Linguistics Utility System (PLUS) is a USPTO automated search system for U.S. Patents from 1971 to the present PLUS is a query-by-example search system which produces a list of patents that are most closely related linguistically to the application searched. This search was prepared by the staff of the Scientific and Technical Information Center, SIRA.

20050066303 99 7054468 42
5047917 42
5185861 42
5339430 42
5390301 42
5434927 42
5546584 42
5737523 42
5764897 42
5808911 42
5825909 42
5864683 42
5930781 42
5966543 42
5987590 42
6105050 42
6113544 42
6145089 42
6182226 42
6209101 42
6226665 42
6263108 42
6298171 42
6312385 42
6373071 42
6393569 42
6418460 42
6421458 42
6421787 42
6477269 42
6526523 42
6553136 42
6563461 42
6567570 42
6600794 42
6658571 42
6658654 42
6728711 42
6747660 42
6760486 42
6804718 42
6823467 42
6868450 42
6981140 42
6992625 42
6993746 42
7016884 42
7035744 42
7043697 42

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	09/21/2004	Donn Rochette	78803 (120-2 US)	7612

27975 7590 11/18/2008
ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER
RONI, SYED A

ART UNIT	PAPER NUMBER
4113	

NOTIFICATION DATE	DELIVERY MODE
11/18/2008	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

creganoa@addmg.com

PTOL-90A (Rev. 04/07)

Office Action Summary	Application No.	Applicant(s)	
	10/946,536	ROCHETTE ET AL.	
	Examiner	Art Unit	
	SYED RONI	4113	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☐ Responsive to communication(s) filed on _____.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☐ Claim(s) _____ is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-20 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on 09/21/2004 is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date 20051018, 20061128.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

Information Disclosure Statement

The references cited within the IDS documents, submitted on October 18, 2005 and November 28, 2006 have been considered.

Claim Rejections - 35 USC § 112

Claims 1 – 20 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claim 1 recites the phrase “said software applications” in line 9 of the claim. There are three different “software applications” in lines 1, 6 and 8. It is unclear which “software applications” is being referred by the phrase.

Claims 2 to 20 are dependent claims and are also rejected since they depend from claim 1.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

Claims 1 - 20 are rejected under 35 U.S.C. 102 (b) as being anticipated by Cabrero et al. (6,260,075 B1).

Regarding **claim 1**, Cabrero et al. teaches a computer system employing a microkernel [i.e., kernel] uses common shared libraries [i.e., shared library]. Rather than each task [i.e., application] setting up its own libraries, during compile a Global Offset Table [i.e., SLCSE] is set up for each task so that the task can use common shared libraries (col 3, lines 29 - 34); (Note; according to claim 1, instances of some of the Shared Library Critical System Elements (SLCSE) are accessible to a plurality of software applications. Likewise, in Cabrero et al, a Global Offset Table (GOT) which is an instance of a Shared Library is accessible to all of the tasks).

Cabrero et al. further teaches that a feature of the microkernel system is to provide a simple, extensible communication kernel. The objective of the microkernel system is to permit the flexible configuration of services in either user or kernel space (col 6 lines 28 - 30). (Note; the services offered by kernel mode is also extensible or available in user mode. Thus, the task is accessing GOT in the use mode [i.e., Context of Software Applications]).

Cabrero et al. further teaches that each task has an associated address map [Instances of Shared Library Critical System Elements] that is separate from the address maps for the other tasks (col 7 lines 11 -13); (Note; each tasks gets the unique instances of GOT. So each task can access shared libraries without conflicting with another task).

Caberero further teaches that user mode tasks have the ability to participate in the address mapping mechanism (col 7 lines 22 - 23). (Note; implies that the task address mapping is done in the context of user task [i.e., context of application]).

Regarding **claim 2**, Cabrero et al. teaches a task [application] can allocate new ranges of memory within its address space (col 7 line 31 - 32). (Note; multiple ranges of address map [instances of SLCSE] is allocated by the task. Thus, multiple different tasks are being run since each address map uniquely identifies each task).

Regards **claim 3**, Cabrero et al. teaches that the Common Shared Libraries (CSL) because of its shared nature, it's Global Offset Table [SLCSE] can only be instrumented to the microkernel [i.e., kernel] common services [essential function] (col 8 lines 40 to 42); (Note; the GOT can also perform kernels common services).

Cabrero et al. further teaches that a feature of the microkernel system is to provide a simple, extensible communication kernel. The objective of the microkernel system is to permit the flexible configuration of services in either user or kernel space (col 6 lines 28 - 30). (Note; the services offered by kernel mode is also extensible or available in user mode or kernel mode. Thus, the functions available to GOT are also available to Kernel [OSCSE]).

Regards **claim 4**, Cabrero et al. teaches that each task [application] has an associated address map, separate from the address maps for the other tasks that is maintained by the kernel and controls the translation of virtual address in the task's address space into physical addresses (col 7 lines 11 - 16). (Note; memory address is being mapped to its respected tasks that separate task A from task B.)

Regarding **claim 5,** Cabrero et al. further teaches that a feature of the microkernel system is to provide a simple, extensible communication kernel. The objective of the microkernel system is to permit the flexible configuration of services in either user or kernel space (col 6 lines 28 - 30). (Note; the services offered by kernel mode is also extensible or available in user mode. Thus, the task is accessing GOT in the use mode [i.e., Context of Software Applications]).

Regards **claim 6,** Cabrero et al. teaches a Global Offset Table (GOT) used by first application task; and generating a second applications task by another of said personality servers, and using said GOT by second application task (col 12 lines 1 - 4).

Regards **claim 7**, the Cabrero et al. teaches that a microkernel task [kernel module] must communicate with many other tasks that provide the required services (col 7 lines 55 - 56).

Cabrero et al. further teaches that the microkernel system uses a client/server system architecture in which tasks access services by making request of other tasks through messages sent [i.e., notification of events] over a communication channel. (col 7 lines 50 - 55)

Cabrero et al. further teaches that this message transfer is an asynchronous operation [i.e., asynchronous event] (col 8 line 4).

Regarding **claim 8**, Cabrero et al. teaches that the microkernel system uses a client/server system architecture in which tasks access services by making request of other tasks through messages sent [i.e., notification of events] over a communication channel (col 7 lines 50 - 55).

Cabrero et al. further teaches that the communication channels of the interprocess communication (IPC) mechanism [i.e., Up Call Mechanism] are called ports [i.e., handler] (Note; In light of the specification, Up Call Mechanism is defined to be a means by which a service in kernel mode executes a functions in a user mode application context).

Regarding **claim 9**, Cabrero et al. teaches that the communication channels of the interprocess communication (IPC) mechanism [i.e., Up Call Mechanism] are called ports [i.e., handler] (Note; In light of the specification, Up Call Mechanism is defined to be a means by which a service in kernel mode executes a functions in a user mode application context).

Regarding **claim 10**, Cabrero et al. teaches that the Common Shared Libraries (CSL) because of its shared nature, it's Global Offset Table [SLCSE] can only be instrumented to the microkernel [i.e., kernel] common services [essential function] (col 8 lines 40 to 42); (Note; the GOT can also perform kernels common services).

Regarding **claim 11**, Cabrero et al. teaches that each task has an associated address map [Instances of Shared Library Critical System Elements] that is separate from the address maps for the other tasks (col 7 lines 11 -13); (Note; each tasks gets the unique instances of GOT. So each task can access shared libraries without conflicting with another task).

Regarding **claim 12**, Cabrero et al. teaches that Global Offset Table (GOT) can only be instrumented to the microkernel common services [i.e., kernel services] (col 8

lines 41 - 42). (Note, services offered by the operating system kernel for device access, interrupt delivery, and virtual memory mapping are considered standard services.)

Regarding **claim 13**, Cabrero et al. teaches that the microkernel system uses a client/server system architecture in which tasks access services by making request of other tasks through messages sent over a communication channel [i.e., Network Protocol Process] (col 7 lines 50 - 55).

Cabrero et al. further teaches that CSL typically needs the following types of services: (a) threading-package-related services; (b) heap management; [i.e., management of files] (c) error reporting/logging; (d) trace message output; (e) dynamic loader services; (f) terminal character output; (g) locks; and (h) semaphores (col 9 lines 25 - 29).

Regarding **claim 14**, Cabrero et al. teaches that each task has an associated address map [Instances of Shared Library Critical System Elements] that is separate from the address maps for the other tasks (col 7 lines 11 -13); (Note; the address map of each task is different from another task).

Regarding **claim 15**, Cabrero et al. teaches that it is an objective of the microkernel system to permit the flexible configuration of services in either user or kernel space (col 6 lines 29 - 30).

Regarding **claim 16**, Cabrero et al. teaches that a feature of the microkernel system is to provide a simple, extensible communication kernel. The objective of the microkernel system is to permit the flexible configuration of services in either user or

kernel space (col 6 lines 28 - 30). (Note; the services offered by kernel mode is also available in user mode and vise versa).

Regarding **claim 17**, Cabrero et al. teaches rather than each task [i.e., application] setting up its own libraries during compile a Global Offset Table (GOT) [i.e., SLCSE] set up for each task (col 3 lines 32 - 35).

Regarding **claim 18**, Cabrero et al. teaches that the Common Shared Libraries (CSL) because of its shared nature, it's Global Offset Table [SLCSE] can only be instrumented to the microkernel [i.e., kernel] common services [essential function] (col 8 lines 40 to 42); (Note; the GOT can also perform kernels common services).

Cabrero et al. further teaches that a feature of the microkernel system is to provide a simple, extensible communication kernel. The objective of the microkernel system is to permit the flexible configuration of services in either user or kernel space (col 6 lines 28 - 30). (Note; the services can be used in either kernel or user mode).

Regarding **claim 19**, Cabrero et al. teaches a computer system employing a microkernel [i.e., kernel] uses common shared libraries [i.e., shared library]. Rather than each task [i.e., application] setting up its own libraries, during compile a Global Offset Table [i.e., SLCSE] is set up for each task so that the task can use common shared libraries (col 3, lines 29 - 34); (Note; according to claim 1, instances of some of the Shared Library Critical System Elements (SLCSE) are accessible to a plurality of software applications. Likewise, in Cabrero et al., a Global Offset Table (GOT) which is an instance of a Shared Library is accessible to all of the tasks).

Regarding **claim 20**, Cabrero et al. teaches that it is an object of the invention to provide an improved microkernel [i.e., Operating System] to provide improved microkernel architecture for a data processing system (col 3 lines 31 -33)

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to SYED RONI whose telephone number is (571)270-7806. The examiner can normally be reached on Monday to Friday, 8:30 AM to 5:00 PM. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Scott B. Geyer can be reached on (571)272-1958. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO

Customer Service Representative or access to the automated information system, call

800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/SYED RONI/
Examiner, Art Unit 4113

/Scott B. Geyer/
Supervisory Patent Examiner, Art Unit 4113

Notice of References Cited	Application/Control No. 10/946,536	Applicant(s)/Patent Under Reexamination ROCHETTE ET AL.	
	Examiner SYED RONI	Art Unit 4113	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-6,260,075	07-2001	Cabrero et al.	719/310
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Index of Claims	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10946536	ROCHETTE ET AL.
	Examiner	**Art Unit**
	SYED RONI	4113

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	11/06/2008								
	1	✓								
	2	✓								
	3	✓								
	4	✓								
	5	✓								
	6	✓								
	7	✓								
	8	✓								
	9	✓								
	10	✓								
	11	✓								
	12	✓								
	13	✓								
	14	✓								
	15	✓								
	16	✓								
	17	✓								
	18	✓								
	19	✓								
	20	✓								

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10946536	ROCHETTE ET AL.
	Examiner	**Art Unit**
	SYED RONI	4113

SEARCHED

Class	Subclass	Date	Examiner
719	310	10/30/2008	SR

SEARCH NOTES

Search Notes	Date	Examiner
East Search notes attached	11/6/2008	SR

INTERFERENCE SEARCH

Class	Subclass	Date	Examiner

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIB DATA SHEET

CONFIRMATION NO. 7612

SERIAL NUMBER	FILING or 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
10/946,536	09/21/2004 **RULE**	718	4113	78803 (120-2 US)

APPLICANTS

Donn Rochette, Fenton, IA;
Paul O'Leary, Kanata, CANADA;
Dean Huffman, Kanata, CANADA;

/SR/

**** CONTINUING DATA ***************************

This appln claims benefit of 60/504,213 09/22/2003 /SR/

**** FOREIGN APPLICATIONS ****************************

none /SR/

**** IF REQUIRED, FOREIGN FILING LICENSE GRANTED **** ** SMALL ENTITY **

11/05/2004

Foreign Priority claimed	☐ Yes ☑ No		STATE OR COUNTRY	SHEETS DRAWINGS	TOTAL CLAIMS	INDEPENDENT CLAIMS
35 USC 119(a-d) conditions met	☐ Yes ☑ No	☐ Met after Allowance				
Verified and Acknowledged	/Syed Roni/ Examiner's Signature	Initials	IA	7	20	1

ADDRESS

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791
UNITED STATES

TITLE

Computing system having user mode critical system elements as shared libraries

FILING FEE RECEIVED		
385	FEES: Authority has been given in Paper No._____ to charge/credit DEPOSIT ACCOUNT No._____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S1	8796	"719".CLAS.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:49
S2	7	((DONN) near2 (ROCHETTE)).INV.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:49
S3	26	((PAUL) near2 (O"LEARY)).INV.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:50
S4	4	((DEAN) near2 (HUFFMAN)).INV.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:50
S5	2	S2 and S3 and S4	US-PGPUB; USPAT	OR	ON	2008/11/06 09:51
S6	8261	application with unique with (element unit code device entity)	US-PGPUB; USPAT	OR	ON	2008/11/06 09:52
S7	3644815	library database memory storage table array	US-PGPUB; USPAT	OR	ON	2008/11/06 09:53
S8	7292	S6 and S7	US-PGPUB; USPAT	OR	ON	2008/11/06 09:53
S9	53021	kernel	US-PGPUB; USPAT	OR	ON	2008/11/06 09:54
S10	561	S8 and S9	US-PGPUB; USPAT	OR	ON	2008/11/06 09:54
S11	8796	"719"/$.ccls.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:55
S12	30	S10 and S11	US-PGPUB; USPAT	OR	ON	2008/11/06 09:55
S14	85677	((shared adj library) database memory storage table array) with multiple with (element unit code device entity)	US-PGPUB; USPAT	OR	ON	2008/11/06 10:37
S15	84556	S14 and "6"	US-PGPUB; USPAT	OR	ON	2008/11/06 10:38
S16	47	(shared adj (service or resource)) with kernel	US-PGPUB; USPAT	OR	ON	2008/11/06 10:39
S17	8	S15 and S16	US-PGPUB; USPAT	OR	ON	2008/11/06 10:39

S19	17	common adj shared adj library	US-PGPUB; USPAT	OR	ON	2008/11/06 10:40

11/6/2008 6:39:02 PM
C:\Documents and Settings\sroni\My Documents\EAST\Workspaces\test.wsp

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	85677	((shared adj library) database memory storage table array) with multiple with (element unit code device entity)	US-PGPUB; USPAT	OR	ON	2008/11/06 10:37
L2	84556	1 and "6"	US-PGPUB; USPAT	OR	ON	2008/11/06 10:38
L3	47	(shared adj (service or resource)) with kernel	US-PGPUB; USPAT	OR	ON	2008/11/06 10:39
L4	8	2 and 3	US-PGPUB; USPAT	OR	ON	2008/11/06 10:39
L6	17	common adj shared adj library	US-PGPUB; USPAT	OR	ON	2008/11/06 10:40
S1	8796	"719".CLAS.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:49
S2	7	((DONN) near2 (ROCHETTE)).INV.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:49
S3	26	((PAUL) near2 (O"LEARY)).INV.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:50
S4	4	((DEAN) near2 (HUFFMAN)).INV.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:50
S5	2	S2 and S3 and S4	US-PGPUB; USPAT	OR	ON	2008/11/06 09:51
S6	8261	application with unique with (element unit code device entity)	US-PGPUB; USPAT	OR	ON	2008/11/06 09:52
S7	3644815	library database memory storage table array	US-PGPUB; USPAT	OR	ON	2008/11/06 09:53
S8	7292	S6 and S7	US-PGPUB; USPAT	OR	ON	2008/11/06 09:53
S9	53021	kernel	US-PGPUB; USPAT	OR	ON	2008/11/06 09:54
S10	561	S8 and S9	US-PGPUB; USPAT	OR	ON	2008/11/06 09:54
S11	8796	"719"/$.ccls.	US-PGPUB; USPAT	OR	ON	2008/11/06 09:55

S12	30	S10 and S11	US-PGPUB; USPAT	OR	ON	2008/11/06 09:55

11/6/2008 10:41:42 AM
C:\Documents and Settings\sroni\My Documents\EAST\Workspaces\test.wsp

Form PTO 1449A U.S. Department of Commerce Patent and Trademark Office Information Disclosure Statement by Applicant	ATTY. DOCKET NUMBER: 78803 (120-2 US)	SERIAL NUMBER: 10/946,536
	APPLICANT: ROCHETTE ET AL	
	FILING DATE: SEPTEMBER 21, 2004	GROUP: 4113 ~~2127~~

U.S. Patent Documents

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING APPL
/S.R./		60/512,103	Oct 20, 2003	Rochette et al			

Foreign Patent Documents

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO

Other Documents (Including Author, Title, Date Pertinent Pages, Etc.)

EXAMINER /Syed Roni/	DATE CONSIDERED 11/06/2008

EXAMINER: Initial if citation is considered, whether or not citation is in conformance with MPEP 609; draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant

PAGE 3/3 * RCVD AT 10/18/2005 12:16:47 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-6/24 * DNIS:2738300 * CSID:13219847078 * DURATION (mm-ss):00-56

Form PTO 1449A U.S. Department of Commerce Patent and Trademark Office Information Disclosure Statement by Applicant	ATTY. DOCKET NUMBER: 78803 (120-2 US)	SERIAL NUMBER: 10/946,536
	APPLICANT: ROCHETTE ET AL.	
	FILING DATE: September 21, 2004	GROUP: 4113 ~~2127~~

U.S. Patent Documents

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING APPE
/S.R./		2002/0004854 A1	Jan 10, 2002	Hartley			
/S.R./		2003/0101292 A1	May 20, 2003	Fisher			
/S.R./		2002/0174215 A1	Nov 21, 2002	Schaefer	709	224	

Foreign Patent Documents

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO
/S.R./		WO 02/06941 A	Jan 24, 2002	Connectix Corporation				
/S.R./		WO 06/039181 A	Apr 13, 2006	Citrix Systems, Inc.				

Other Documents (Including Author, Title, Date Pertinent Pages, Etc.)

EXAMINER /Syed Roni/	DATE CONSIDERED 11/06/2008

EXAMINER: Initial if citation is considered, whether or not citation is in conformance with MPEP 609; draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: **ROCHETTE ET AL.**	)	Atty. Docket No.: **78803 (120-2 US)**
Serial No. **10/946,536**	)	Art Unit: **4113**
Filing Date: **SEPTEMBER 21, 2004**	)	Examiner: **SYED A. RONI**
Confirmation No. **7612**	)	
For: **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**	)	

AMENDMENT

EFILED

Dear Sir:

In response to the Office Action dated November 18, 2008 please amend the above-identified application as follows:

Amendments to the Claims are reflected in the listing of claims, which begins on page 2 of this paper.

Remarks begin on page 7 of this paper.

****

Amendments to the Claims

1. (currently amended) A computing system for executing a plurality of software applications comprising:

a) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

b) a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are replicas of OSCSEs and are accessible to some of the [[a]] plurality of ~~the~~ software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it ~~form~~ forms a part of the one or more of the plurality of software applications, and

ii) wherein an instance of a SLCSE provided to [[an]] one or more of the plurality of software ~~application~~ applications from the shared library is run in a context of said one or more of the plurality of software ~~application~~ applications without being shared with other of the plurality of software applications and where ~~some~~ one or more other of the plurality of software applications running under the operating system ~~each~~ have use of a unique instance of a corresponding critical system element for performing essentially the same function.

2. (original) A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in

the shared library run simultaneously within the operating system.

3. (original) A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing essentially the same function as SLCSEs remain in the operating system kernel.

4. (currently amended) A computing system according to claim 1 wherein the one or more SLCSEs provided to [[an]] one of the plurality of software ~~application~~ applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. (original) A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between a SLCSE in the context of an application program and a device driver.

6. (currently amended) A computing system as defined in claim 1, wherein an SLCSE related to a predetermined function is provided to a first of the plurality of software ~~application~~ applications for running [[an]] first instance of the SLCSE, and wherein an SLCSE for performing essentially a same function is provided to a second of the plurality of software ~~application~~ applications for running a second instance of the SLCSE simultaneously.

7. (original) A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous

event and requires information to be passed to the SLCSE from outside the application.

8. (currently amended) A computing system according to claim 7 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications ~~an application~~ through the use of an up call mechanism.

9. (original) A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

10. (currently amended) A computing system according to claim 2, wherein a function overlay is used to provide [[a]] one of the plurality of software ~~application~~ applications access to operating system services.

11. (currently amended) A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

12. (original) A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

13. (original) A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

14. (currently amended) A computing system according to claim 11 wherein some SLCSEs are modified for a particular one of the plurality of software ~~application~~ applications.

15. (original) A computing system according to claim 14 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

16. (original) A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

17. (currently amended) A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

18. (original) A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

19. (original) A computer system as defined in claim 2 wherein SLCSEs are not copies of OSLCEs.

20. (original) An operating system comprising the computing system of claim 2.

REMARKS

Claims 1-20 are pending in this application and are rejected under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claim 1 recites the phrase "said software applications" in line 9 of the claim. As the Examiner correctly points out, there are three different "software applications" recited in lines 1, 6, and 8, and it is unclear which "software applications" are being referred to by the phrase.

Claims 1, 6 and 8 have been amended to obviate this indefiniteness and the term "plurality of software applications" recited in the preamble is consistently used within the body of the claims.

Claims 1-20 are rejected under 35 U.S.C. § 102 (b) as being anticipated by U.S. Patent 6,260,075 B1 ("Cabrero et al.").

Claim 1 of the instant application has been amended so as to more clearly distinguish from the teachings of Cabrero et al. and to more distinctly define the invention.

Applicants have carefully considered the Examiner's reasons for rejecting claim 1 and would like to clarify substantial differences between Cabrero et al. and the newly amended claim 1 of the instant application.

The Examiner states in the Office Action that Cabrero et al. teach a computer system employing a micro-kernel (i.e. a kernel), which uses common shared libraries. This is correct, however, Cabrero et al. make no mention of "operating system critical system elements" or "shared library critical system elements". It is fully acknowledged that Cabrero et al. describe shared libraries. However, the claimed invention utilizes such shared libraries as a way of duplicating critical system elements provided by the operating system and Cabrero et al. do not do this. In a system having a micro-kernel, critical services such as TCP IP, and file services are stand alone applications that are not part of the micro-kernel, thus, they do not reside within the micro-kernel, this in fact being a difference in the structure of a micro-kernel from a monolithic kernel. This difference, in fact, gives rise to the claimed invention. Applicants' claim is to a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode wherein the OSCSEs are replicated as SLCSEs stored in the shared libraries. Therefore, a TCP IP application within the user library is used by a voice processing application like Skype, such that TCP IP becomes part of the application requesting it, wherein it can be tailored to handing Skype and general purpose requests for TCP IP can be provided by the OS, since the CSE resides in both the kernel and in the shared library.

Claims 2 and 3 define embodiments, with further distinctions over Cabrero et al.

Claim 2 defines: A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

Once again, Applicants acknowledge that Cabrero et al. have and make use of shared libraries. What Cabrero et al. do not do is replicate CSEs present in the OS so that they can also be stored within the shared libraries; and they do not have the capability of running a CSE "simultaneously" from the shared library and the OS. In fact, Cabrero et al.'s TCP IP and other essential services are independent applications that neither reside in the OS nor in shared libraries.

Claim 3 defines a computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing essentially the same function as SLCSEs remain in the operating system kernel. This claim reinforces the fact that OSCSEs that correspond to SLCSEs remain within the kernel. Once again, it must be stressed that CSEs, such as TCP IP, do not reside in Cabrero et al.'s micro-kernel or in the shared libraries.

Amended claim 4 defines a computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

Applicants would like to put claim 4 in context of claim 1 from which it depends. Claim 1 defines a system wherein critical system elements normally residing in the OS under the

control of the kernel are replicated and stored within shared libraries so that applications requiring the use of the CSE can incorporate it into the application as if it would be a part of the application. At the same time, the CSE can be utilized by the kernel for general purpose requests. An example given before of such a CSE is TCP IP. Claim 4 further recites the one or more SLCSEs provided to one of the plurality of software applications having exclusive use of the SLCSE and wherein system calls are used to access services in the operating system kernel. Cabrero et al. teach no such system.

Claim 5 defines a computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between a SLCSE in the context of an application program and a device driver.

There is no doubt that Cabrero et al. do not teach or suggest the feature defined in claim 5. Firstly, Cabrero et al. have no SLCSEs that are replicas of OS CSEs, which claim 5 defines. Secondly, Cabrero et al. have a micro-kernel, which is not adapted to serve as an interface between SLCSEs in the context of an application program and a device driver. It should be noted that a SLCSE in the context of an application is, for example, TCP IP under the request/control of an application. This SLCSE has a corresponding OS CSE, which is not the case in Cabrero et al.

Claim 6 defines: A computing system as defined in claim 1, wherein an SLCSE related to a predetermined function is provided to a first of the plurality of software applications for running first instance of the SLCSE, and wherein an SLCSE

for performing essentially a same function is provided to a second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

Applicants appreciate the Examiner's explanation of the global offset table (GOT) taught by Cabrero et al. Notwithstanding, claim 6 defines an entirely different invention than Cabrero et al. teach.

For example, claim 6 provides a system where critical system elements or services are replicated and can be accessed simultaneously by the kernel or by an application in user mode by way of the application using an instance of the CSE within a shared library. As was pointed out before, this is in contradistinction to what is taught by Cabrero et al. Secondly, claim 6 allows different instances of an SLCSE performing a same function to be executed simultaneously by another one or more applications. However, in the context of the system defined in claim 1, claim 6 adds further distinctions to claim 1 which clearly distinguishes from Cabrero et al.

Claim 8 is rejected as being obvious in view of Cabrero et al. The Examiner indicates that Cabrero et al. further teach that this message transfer is an asynchronous operation [i.e., a synchronous event]. In rejecting claim 8 the Examiner states that Cabrero et al. teach that a micro-kernel system uses client/server system architecture in which tasks access services by making request of other tasks through messages sent over a communication channel; and that Cabrero et al. further teach that the communication channels of the interprocess communication (IPC) mechanism [i.e., Up Call

Mechanism] are called ports [i.e., handler]. The Examiner further suggests that in light of the specification, Up Call Mechanism is defined to be a means by which a service in kernel mode executed a function in a user mode application context.

Firstly, Applicants would like to point out that claim 8 depends upon claims 7, 5, and 1 and includes the recitations from these claims from which it depends. More importantly, claim 8 distinguishes from Cabrero et al. who do not teach an up-call mechanism. Message passing in an Inter-process communication (IPC) in a micro-kernel is significantly different than an Up Call Mechanism. For example, with a message passing paradigm a process makes a system call to the kernel, causing a message to be placed in a queue. A second process makes a system call to the kernel to receive any message in said queue. A response is delivered by the second process through a system call to the kernel, resulting in the response message being placed in a queue. The first process receives the response by making a system call to the kernel to retrieve any message on said queue. An upcall mechanism allows a function in kernel mode to execute code in user mode in the following manner: parameters for the call being made in user mode are taken from the kernel stack and used to build a stack in the user mode process address space, as part of a larger stack frame for a trap return operation. A CPU trap return instruction is executed to transition the CPU from privileged/kernel mode to protected/user mode. It is clear that the two mechanisms are vastly different.

Claim 9 further distinguishes from the teachings of Cabrero et al. by defining that the Up Call mechanism allows a

function in kernel mode to execute code resident in a user mode CSE. The benefit of this is significant. The difference between user and kernel mode is significant. Furthermore, the ability to execute code in this manner has a profound impact on performance and simplicity implications. One effective way to provide context to the efficacy of an Up Call Mechanism is to use an analogy. Where an RPC (Remote Procedure Call) allows a process on one computer to execute functions on another computer, an Up Call allows kernel code to execute user code.

Amended Claim 10 defines the computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services. The use of a functional overlay provides an efficiency mechanism allowing user mode CSE to more effectively access kernel mode code. This is simply not suggested by Cabrero et al.

Claim 11 is rejected in view of Cabrero et al. The Examiner suggests that Cabrero et al. teach that each task has an associated address map [Instances of Shared Library CSE] that is separate from the address maps for other tasks (col 7 lines 11-13). It is further suggested by the Examiner that since each task gets the unique instance of the GOT, each task can access shared libraries without conflicting with another task.

Claim 11 defines: A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications <u>of the plurality of software applications</u> as the <u>particular</u> software applications are

loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

As was indicated above, Cabrero et al's system, which uses a micro-kernel has critical system elements, such as TCP IP, as separate programs distinct from those that run in shared libraries. Claim 11 distinctly claims that user mode CSEs are part of an application, distinct from the micro-kernel design where user mode CSEs are independent applications. Furthermore, these CSEs are replicated CSEs that remain a part of the kernel. The Examiner suggests that the GOT allows a unique instance of a task within a shared library. The invention defined in claim 11 is simply not suggested by Cabrero et al. Their teaching of a micro-kernel, and shared libraries with all the typical functionality of a shared library does not include the Applicants' claimed invention, wherein CSEs are replicated and can be accessed by the kernel or by applications from shared libraries. Claim 11 still further defines this functionality in a manner not suggested by Cabrero et al.

Claim 12 is rejected in view of Cabrero et al. The Examiner again suggests that Cabrero et al.'s GOT can only be instrumented to the micro-kernel common services and that services offered by the OS kernel for device access, interrupt delivery, and virtual mapping are considered standard services.

Claim 12 defines a computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the

operating system kernel for device access, interrupt delivery, and virtual memory mapping.

Once again, claim 12 must be taken in context of its full recitations.

Claim 12 recites a computing system for executing a plurality of software applications comprising:
a) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,
b) a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, and

ii) wherein an instance of a SLCSE provided to one or more of the plurality of software applications from the shared library is run in a context of said one or more of the plurality of software applications without being shared with other of the plurality of software applications and where one or more other of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing essentially the same function, and

wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

It should be evident after reading claim 12 that its recitations are not taught by Cabrero et al. who teach a micro-kernel system where user mode CSEs are independent programs not residing in shared libraries and where OSCSEs are not replicated into user mode CSEs and wherein SLCSEs residing in a shared library (i.e. replicated OSCSEs) utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

Claim 13 further defines the replicated services to include services related to at least one of network protocol processes and the management of files. Cabrero et al.'s network protocol processes and file management processes are independent programs neither residing in the kernel nor in a shared library.

Claim 14 has been rejected in view of Cabrero et al. Amended claim 14 defines computing system according to claim 11 wherein some SLCSEs are modified for a particular <u>one of the plurality of</u> software application<u>s</u>.

In rejecting claim 14, which incorporates all of the recitations of the claims from which it depends and which are patentable, the Examiner states that Cabrero et al. teach that each task has an associated address map [Instances of shared library CSEs] that are separate from the address maps of other tasks. Notwithstanding, the system defined in claim 14 is

simply not taught by Cabrero et al. One of the important features of the claimed invention is that an OS critical system element may have a replica that is tailored to a specific application while at the same time a general purpose OS CSE that is directed to a same service resides in the kernel for general use. Having the service reside with the kernel for general use and having a replica of that service with specific attributes within the shared library offers much greater efficiency. There is no mention of this in Cabrero et al.

Claim 15 further defines this feature wherein the SLCSEs that are application specific reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel. The Examiner, in rejecting claim 15, has stated that Cabrero et al. teach that a feature of a micro-kernel system is to provide the flexible configuration of services in either user space or kernel space. Firstly, this is certainly not a recitation of Applicants' claim 15, and secondly, this implies that some services will be in "either" the user space or kernel space. This is not a teaching of having the services in both user space and kernel space. In a micro-kernel architecture, critical services neither reside in user space nor kernel space; they reside as separate applications accessible by the micro-kernel.

In summary, the claimed invention provides services normally found within the kernel to be duplicated so as to be present within shared libraries so as to form a part of applications requesting these critical services. Thus, these CSEs are present in the kernel and in the shared libraries and

can be run simultaneously. Furthermore, a CSE may have several unique instances running from different applications requesting the service from a shared library, where at the same time the same service, i.e. TCP IP, may be running in a more general purpose manner from within the kernel.

In view of the amended claims and the Applicants' comments regarding the patentable differences over the system taught by Cabrero et al., Applicants believe that the claims of the instant application are now in condition for allowance.

Early and favorable reconsideration of the application would be appreciated.

Should any minor informalities need to be addressed, the Examiner is encouraged to contact the undersigned attorney at the telephone number listed below.

Please charge any shortage in fees due in connection with the filing of this paper, including Extension of Time fees, to Deposit Account No. 50-2810 and please credit any excess fees to such deposit account.

Respectfully submitted,

/CHRISTOPHER F. REGAN/
Christopher F. Regan
REG. NO. 34,906

Customer No.: 27975

Telephone: (407) 841-2330 Date: February 18, 2009

Electronic Acknowledgement Receipt

EFS ID:	4812976
Application Number:	10946536
International Application Number:	
Confirmation Number:	7612
Title of Invention:	Computing system having user mode critical system elements as shared libraries
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	Christopher F. Regan/joellen murphy
Filer Authorized By:	Christopher F. Regan
Attorney Docket Number:	78803 (120-2 US)
Receipt Date:	18-FEB-2009
Filing Date:	21-SEP-2004
Time Stamp:	13:31:26
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		78803amd1.pdf	110203 f93c399dde635cbf5252e14e3c2db34ffa556ebe	yes	18

	Multipart Description/PDF files in .zip description		
	Document Description	**Start**	**End**
	Amendment/Req. Reconsideration-After Non-Final Reject	1	1
	Claims	2	6
	Applicant Arguments/Remarks Made in an Amendment	7	18
Warnings:			
Information:			
	Total Files Size (in bytes):	110203	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

<u>New Applications Under 35 U.S.C. 111</u>
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

<u>National Stage of an International Application under 35 U.S.C. 371</u>
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

<u>New International Application Filed with the USPTO as a Receiving Office</u>
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (07-06)
Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number	Filing Date	
10/946,536	09/21/2004	☐ To be Mailed

APPLICATION AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	SMALL ENTITY ☒ RATE ($)	SMALL ENTITY FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =		OR	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =			X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT 02/18/2009

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	* 20	Minus	** 20	= 0	X $26 =	0	OR	X $ =	
Independent (37 CFR 1.16(h))	* 1	Minus	***3	= 0	X $110 =	0	OR	X $ =	
☐ Application Size Fee (37 CFR 1.16(s))									
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
					TOTAL ADD'L FEE	0	OR	TOTAL ADD'L FEE	

AMENDMENT

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)		RATE ($)	ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
☐ Application Size Fee (37 CFR 1.16(s))									
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
					TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Legal Instrument Examiner:
/LINDA WISE/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**
If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	09/21/2004	Donn Rochette	78803 (120-2 US)	7612

27975 7590 04/02/2009
ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER
RONI, SYED A

ART UNIT	PAPER NUMBER
4113	

NOTIFICATION DATE	DELIVERY MODE
04/02/2009	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

creganoa@addmg.com

PTOL-90A (Rev. 04/07)

Office Action Summary	Application No.	Applicant(s)	
	10/946,536	ROCHETTE ET AL.	
	Examiner	Art Unit	
	SYED RONI	4113	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☐ Responsive to communication(s) filed on _____.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-20 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-20 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on 21 September 2004 is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

Claims 1-20 are rejected under 35 U.S.C. 102 (b) as being anticipated by Cabrero et al. (6,260,075 B1).

Regarding **claim 1**, (currently amended) A computing system for executing a plurality of software applications comprising: a) **an operating system having an operating system kernel** [i.e., Cabrero et al. teaches an improved microkernel architecture for a data processing system (Column 3, lines 28 - 29)] **having OS critical system elements (OSCSEs) for running in kernel mode** [i.e., Cabrero et al. teaches in figure 1 that microkernel includes virtual memory module 23, tasks and threads module 24, host and processor sets 25 and I/O support and interrupts module 26 (see figure 1)]; and, b) **a shared library having critical system elements (SLCSEs) stored therein for use by the <u>plurality of</u> software applications in user mode** [i.e., Cabrero et al. teaches in figure 2 that application tasks 40a – 41b share a common share library 44. An example of a common shared library is an interface library for the microkernel OS or operating system of figure 1 (Column 8, lines 36 – 40)] and

i) wherein **some of the SLCSEs stored in the shared library are <u>replicas of OSCSEs</u>**

[i.e., Cabrero et al. teaches that the CSL 44, because of its shared nature, e.g., its global offset table, can only be instrumented to the microkernel common services (Column 8, lines 40 - 42)] and **are accessible to some of the [[a]] plurality** of t-be **software applications** [i.e., Cabrero et al. teaches that rather than each task setting up its own libraries, during compile a global offset table is set up for each task so the tasks can use common shared libraries (Column 3, lines 33 - 35)] and **when one of the SLCSEs is accessed by one or more of the plurality of software applications it** 4em-m **forms a part of the one or more of the plurality of software applications**, and

ii) wherein **an instance of a SLCSE provided to [[an]] one or more of the plurality of software** application **applications from the shared library** [i.e., Cabrero et al. teaches that various tasks 40 and 41 executing in the computer system 10 under the operating system of figure 1, will, according to a feature of the present invention, use the same GOT (Column 9, lines 4 - 8)] **is run in a context of said one or more of the plurality of software** ~~application~~ **applications** [i.e., Cabrero et al. teaches that unlike traditional virtual memory designs, the microkernel system 10 does not implement all the caching itself. It gives user mode tasks the ability to participate in these mechanism (Column 7, lines 20 - 23)] **without being shared with other of the plurality of software applications** and where **one or more other of the plurality of software applications running under the operating** ~~system each~~ **have use of a unique instance of a corresponding critical system element** for **performing essentially the same function** [i.e., Cabrero et al. teaches that dominant personality applications 38 are

UNIX type applications which would run on top of the UNIX operation system personality 32. The alternative personality application 39 are OS/2 applications which run on top of the OS/2 alternative personality operation system 35 (Column 5, lines 37 - 43), (see figure 1)].

Regarding **claim 2**, (original) A computing system as defined in claim 1, wherein in operation, **multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system** [i.e., Cabrero et al. teaches that the CSL because of its shared nature, e.g., its global offset table, can only be instrumented to the microkernel common services (Column 8, lines 40 - 42)], [i.e., Cabrero et al. further teaches in figure 1 that multiple personality applications can be run on a microkernel at the same time (see figure 1)].

Regarding **claim 3**, (original) A computing system according to claim 1 wherein **OSCSEs corresponding to and capable of performing essentially the same function as SLCSEs** [i.e., Cabrero et al. teaches that the CSL because of its shared nature, e.g., its global offset table, can only be instrumented to the microkernel common services (Column 8, lines 40 - 42)] **remain in the operating system kernel** [i.e., Cabrero et al. teaches that it is an objective of the microkernel system to permit the flexible configuration of services in either user or kernel space (Column 6, lines 28 - 30)]

Regarding **claim 4**, (currently amended) A computing system according to claim 1 wherein **the one or more SLCSEs provided to** [[an]] **one <u>of the plurality of software</u> applications having exclusive use thereof** [i.e., Cabrero et al. teaches that application tasks 40a – 41b share a common shared library (Column 8, lines 37 - 38)],

[i.e., Cabrero et al. further teaches that the global offset table of common shared library is instrumented with the microkernel common services (Column 8, lines 40 - 42)],, **use system calls to access services in the operating system kernel** [i.e., Cabrero et al. teaches in figure 1 that tasks communicate with microkernel via IPC module (see figure 1)], [i.e., Cabrero et al. further teaches that since the microkernel provides very few services of its own, a microkernel task must communicate with many other tasks that provide the required services (Column 7, lines 53 - 56)].

Regarding **claim 5**, (original) A computing system according to claim 1 wherein **the operating system kernel comprises a kernel module** [(see figure 1, number 12)] **adapted to serve as an interface between a SLCSE in the context of an application program and a device driver** [(see figure1, number 30)], [i.e., Cabrero et al. teaches that an example of common shared library is an interface library for the microkernel (Column 8, lines 38 - 40)] [i.e., Cabrero et al. further teaches that it is an objective of the microkernel system to permit flexible configuration of services like virtual memory 23, task and threads 24 and I/O supports 26 etc,. in either user or kernel space (Column 6, lines 27 - 30)]

Regarding **claim 6**, (currently amended) A computing system as defined in claim 1, wherein **an SLCSE related to a predetermined function is provided to a first <u>of the plurality of software</u> applications for running [[an]] <u>first</u> instance of the SLCSE** [i.e., Cabrero et al. teaches that application tasks 40a – 41b share a common shared library (Column 8, lines 37 - 38)], [i.e., Cabrero et al. further teaches that the global offset table of common shared library is instrumented with the microkernel common

services (Column 8, lines 40 - 42)], and wherein an **SLCSE for performing essentially a same function is provided to a second <u>of the plurality of software</u> applications** [i.e., Cabrero et al. teaches in figure 1 that dominant personality and alternative personality servers both could perform a user task such as default pager that provides paging for the main memory (see figure 1)] for running a second instance of the SLCSE **simultaneously** [i.e., Cabrero et al. teaches that an OS2 task 40 and a "Personality Neutral" PN task 41 may be executing at the same time (Column 8, lines 13 - 14)].

Regarding **claim 7**, (original) A computing system according to claim 5 wherein **the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program** [i.e., Cabrero et al. teaches that the personality servers 14 use the message-passing services of the microkernel to communicate with the personality neutral services (Column 4, lines 62 - 64)], wherein **the event is an asynchronous event** [i.e., Cabrero et al. teaches that this message transfer is an asynchronous operation (Column 8, line 4)] and **requires information to be passed to the SLCSE from outside the application** [i.e., Cabrero et al. teaches that it is an objective of the microkernel system 10 to permit the flexible configuration of services in either user or kernel space (Column 6, lines 28 - 30)]

Regarding **claim 8**, (currently amended) A computing system according to claim 7 wherein **a handler is provided for notifying the SLCSE in the context of <u>one of the plurality of software applications</u>** [i.e., Cabrero et al. teaches that any personality server can send a message to a personality neutral disk driver and ask it to read a block of data from the disk. The disk driver reads the block and returns it in a message. The

message system is optimized so large amounts of data are transferred rapidly by manipulation pointers (Column 4, lines 65 -67) and (Column 5, lines 1 - 3)] through the use of **an up call mechanism** [i.e., Cabrero et al. teaches that common kernel services can be configured either in user or kernel mode with the minimum amount of function in the kernel proper (Column 6, lines 28 - 31)].

Regarding **claim 9**, (original) A computing system according to claim 7 wherein the up call mechanism in operation, **executes instructions from an SLCSE resident in user mode space, in kernel mode** [i.e., Cabrero et al. teaches that common kernel services can be configured either in user or kernel mode with the minimum amount of function in the kernel proper (Column 6, lines 28 - 31)].

Regarding **claim 10**, (currently amended) A computing system according to claim 2, wherein a function overlay is used to **provide [[a]]** **<u>one of the plurality of</u> software** ~~application~~ **applications access** [i.e., i.e., Cabrero et al. teaches that application tasks share a common share library (Column 8, lines 37 - 38)] to **operating system services** [i.e., Cabrero et al. teaches that global offset table of common shared library is instrumented to the microkernel common services (Column 8, lines 41 - 42)], [i.e., Cabrero et al. further teaches that abstraction layer mechanism used by common shared library to access the common interface (Column 8, lines 57 - 62)].

Regarding **claim 11**, (currently amended) A computing system according to claim 2 wherein **SLCSEs stored in the shared library are linked to particular software applications** **<u>of the plurality of software applications</u>** [i.e., Cabrero et al. teaches that application tasks 40 and 41 have a common shared library and the global offset table of

shared library is instrumented with the microkernel common services (Column 8, lines 36 - 41)] as the particular software applications are loaded such that **the particular software applications have a link that provides unique access to a unique instance of a CSE** [i.e., Cabrero et al. further teaches that each task has an associated address map that is separated from the address maps for the other tasks (Column 7, lines 11 - 13)].

Regarding **claim 12**, (original) A computing system according to claim 2 wherein **the SLCSEs utilize kernel services supplied by the operating system kernel** [i.e., Cabrero et al. teaches that global offset table of common shared library is instrumented to the microkernel common services (Column 8, lines 41 - 42)] **for device access, interrupt delivery, and virtual memory mapping** [i.e., Cabrero et al. teaches in figure 1 that microkernel 12 includes virtual memory 23, I/O support and interrupts 26 and device drivers (see figure 1)].

Regarding **claim 13**, (original) A computing system according to claim 1, wherein **SLCSEs include services related to at least one of, network protocol processes** [i.e., network services (see 31, figure 1)]**, and the management of files** [i.e., see 31, figure 1], [i.e., Cabrero et al. teaches that global offset table of common shared library is instrumented to the microkernel common services (Column 8, lines 41 - 42)].

Regarding **claim 14**, (currently amended) A computing system according to claim 11 wherein **some SLCSEs are modified for a particular one of the plurality of software** ~~application~~ **applications** [i.e., Cabrero et al. teaches that the global offset table is generated at compile time, then fixed-up at runtime (Column 9, lines 9 - 10)].

Regarding **claim 15**, (original) A computing system according to claim 14 wherein the **SLCSEs that are application specific, reside in user mode** [i.e., Personality neutral servers (see 13, figure 1)], [i.e., Cabrero et al. teaches in figure 1 that device drivers, personality servers, file servers reside in user mode (see figure 1)], while **critical system elements, which are platform specific, reside in the operating system kernel** [i.e., Cabrero et al. teaches that IPC module, virtual memory, tasks & threads, I/O support & interrupts reside in kernel mode (see 12 figure 1)].

Regarding **claim 16**, (original) A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode [i.e., Cabrero et al. teaches that a feature of the microkernel system is to provide a simple, extensible communication kernel. The objective of the microkernel system is to permit the flexible configuration of services in either user or kernel space (Column 6, lines 28 - 30)].

Regarding **claim 17**, (currently amended) A computing system according to claim 1 wherein SLCSEs form a part of at least some of the <u>plurality of</u> software applications, by being linked thereto [i.e., Cabrero et al. teaches rather than each task setting up its own libraries during compile a Global Offset Table (GOT) set up for each task (Column 3, lines 32 - 35)].

Regarding **claim 18**, (original) A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel [i.e., Cabrero et al. teaches that the Common Shared Libraries because of its shared nature, it's global offset table can only be instrumented to the microkernel common services (Column 8, lines 40 to 42)].

Regarding **claim 19**, (original) A computer system as defined in claim 2 wherein SLCSEs are not copies of OSLCEs [i.e., Cabrero et al. teaches a compute system employing a microkernel uses common shared libraries. Rather than each task setting up its own libraries, during compile a global offset table is set up for each task so that the task can use common shared libraries (Column 3, lines 29 - 34)].

Regarding **claim 20**, (original) An operating system comprising the computing system of claim 2 [i.e., Cabrero et al. teaches that it is an object of the invention to provide an improved microkernel to provide improved microkernel architecture for a data processing system (Column 3, lines 31 -33)].

Response to Arguments

Applicant's arguments filed 02/18/2009 have been fully considered but they are not persuasive.

Regarding **claim 1**, Applicant's arguments were found to be unpersuasive because Cabrero et al. disclosed a microkernel with critical system elements such as IPC, virtual memory management and I/O supports. Thus, the above mentioned

microkernel meets the limitation of having a kernel with OS critical system elements. The microkernel architecture does reduced the size of monolithic kernel to basic process communication and I/O control in kernel space and moving other services to user space in a form of normal processes called servers.

The claimed invention further recites a shared library having critical system elements (SLCSEs). Cabrero et al. also disclosed a common shared library with its global offset table that is instrumented with the microkernel common services. Thus, the global offset table meets the limitation of critical system elements of shared library that duplicates critical system elements provided by the operation system kernel.

Regarding **claim 2**, Applicant's arguments were found to be unpersuasive because Cabrero et al. also disclosed a common shared library with its global offset table that is instrumented with the microkernel common services. Thus, Critical System Elements (CSEs) present in the OS kernel are also present in the Shared Library within a global offset table.

Cabrero et al. disclosed common Critical System Elements (CSEs) in both kernel space and user space and application tasks 40 and 41 both can access the common services through shared library and through abstraction layer of OS.

Regarding **claim 3**, Applicant's arguments were found to be unpersuasive because Cbrero et al. teaches that it is an objective of the microkernel system is to permit the flexible configuration of services in either user or kernel space. Thus, it meets the limitation of OSCSEs corresponding to SLCSEs remain in the OS kernel of the claimed invention.

In rejection of claim 3 based on the limitation of the claimed invention, the TCP/IP protocols were not considered or relied upon by the examiner.

Regarding **claim 4**, Applicant's arguments were found to be unpersuasive because the dependency of claim 4 from claim 1 has been fully acknowledged. Applications accessing the critical system elements of shared library and using IPC module of the kernel to access OS critical elements meet the limitation of the claimed invention.

In rejection of claim 4 based on the limitation of the claimed invention, the TCP/IP protocols were not considered or relied upon by the examiner.

Regarding **claim 5**, Applicant's arguments were found to be unpersuasive because Cabrero et al. also disclosed a common shared library with its global offset table that is instrumented with the microkernel common services.

According to Cabrero et al. Shared library is an interface library of the microkernel which interfaces between kernel common services and processes in user space such as device support and other personality neutral products.

In rejection of claim 5 based on the limitation of the claimed invention, the TCP/IP protocols were not considered or relied upon by the examiner.

Regarding **claim 6**, Applicant's arguments were found to be unpersuasive because Cabrero et al. disclosed the limitation of providing an SLCSE to a first application and providing SLCSE to second application simultaneously of claimed invention.

Cabrero et al. teaches that the global offset table of common shared library is instrumented with the microkernel common services and application tasks access the global offset table to use the common services available in Shared library in user space and kernel space.

Regarding **claims 8 and 9**, Applicant's arguments were found to be unpersuasive because the dependency of claims 8 and 9 from claims 7, 5, and 1 respectively has been fully acknowledged.

Cabrero et al. teaches that common kernel services can be configured either in user or kernel mode with the minimum amount of function in the kernel proper. Thus, the limitation of upcall mechanism, allows a function in kernel mode to execute code in user mode, meets the recitation of Cabrero et al.

Regarding **claim 10**, Applicant's arguments were found to be unpersuasive because Cabrero et al. teaches that global offset table of common shared library is instrumented to the microkernel common services. The common shared library uses a mechanism called OS abstraction layer to access a common interface meets the limitation of claimed invention.

Regarding **claim 11**, Applicant's arguments were found to be unpersuasive because Cabrero et al. discloses that application tasks 40 and 41 have a common shared library and the global offset table of shared library is instrumented with the microkernel common services meets the limitation of SLCSE are linked to particular software application of the claimed invention.

Regarding **claim 12**, Applicant's argument found to be unpersuasive because microkernel's common services are also present in the global offset table of the shared library. Microkernel allows services such as virtual memory, task & threads, I/O support and interrupt and device drivers to be accessed by the application tasks in either user or kernel space. Thus, CSEs of shared library are not independent programs that reside outside of shared library and OSCSEs are being replicated into user mode CSEs.

Regarding **claim 13**, Applicant's argument found to be unpersuasive because network services and file server of microkernel architecture resides in user space. Cabrero et al. further teaches that shared library's CSE are replicas of OS CSE.

Regarding **claim 14**, Applicant's arguments were found to be unpersuasive because Cabrero et al. disclosed that the global offset table of a shared library instrumented with the common services of microkernel. The global offset table is generated at compile time, then fixed-up at runtime according to user mode applications. The common services could be configured by application at user space or kernel space. Thus, the common services are present both in user mode and kernel mode and meet the limitation recited in claimed invention.

Regarding **claim 15**, Applicant's arguments were found to be unpersuasive because the concept behind the microkernel architecture is to reduce the kernel space to basic process communication and I/O control, and let the other services reside in user space in form of normal processes called servers. Thus, the limitations of critical system elements in the claimed invention are also present in the microkernel of Cabrero et al.

Conclusion

THIS ACTION IS MADE FINAL. Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to SYED RONI whose telephone number is (571)270-7806. The examiner can normally be reached on Monday to Friday, 8:30 AM to 5:00 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Scott B. Geyer can be reached on (571)272-1958. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/SYED RONI/
Examiner, Art Unit 4113

/Scott B. Geyer/

Supervisory Patent Examiner, Art Unit 4113

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	09/21/2004	Donn Rochette	78803 (120-2 US)	7612

27975 7590 06/16/2009
ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER
RONI, SYED A

ART UNIT	PAPER NUMBER
2194	

NOTIFICATION DATE	DELIVERY MODE
06/16/2009	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

creganoa@addmg.com

PTOL-90A (Rev. 04/07)

Interview Summary	Application No.	Applicant(s)	
	10/946,536	ROCHETTE ET AL.	
	Examiner	Art Unit	
	SYED RONI	2194	

All participants (applicant, applicant's representative, PTO personnel):

(1) *SYED RONI*.

(2) *David S. Carus*.

(3)*Hyung Sub Sough*.

(4)_____.

Date of Interview: *03 June 2009*.

Type: a)☒ Telephonic b)☐ Video Conference
c)☐ Personal [copy given to: 1)☐ applicant 2)☐ applicant's representative]

Exhibit shown or demonstration conducted: d)☐ Yes e)☒ No.
If Yes, brief description: _____.

Claim(s) discussed: *1*.

Identification of prior art discussed: *Cabrero et al. (US 6,260,075 B1)*.

Agreement with respect to the claims f)☐ was reached. g)☐ was not reached. h)☒ N/A.

Substance of Interview including description of the general nature of what was agreed to if an agreement was reached, or any other comments: *Applicant argued that the prior art Cabrero et al. do not disclose SLCSEs stored in the shared library are replicas of OSCSEs. Examiner pointed out the limitation in Cabrero et al. (Column 8, lines 40 - 42) Examiner gave his interpretation of the " insturmented to the microkerne common service" as replicas of OSCEs. Applicant further argued that Cabrero et al. do not teach elements of the shared library are not shared by different software application. Examiner would further consider the applicant's argument after receiving the applicant's written response.*.

(A fuller description, if necessary, and a copy of the amendments which the examiner agreed would render the claims allowable, if available, must be attached. Also, where no copy of the amendments that would render the claims allowable is available, a summary thereof must be attached.)

THE FORMAL WRITTEN REPLY TO THE LAST OFFICE ACTION MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW. (See MPEP Section 713.04). If a reply to the last Office action has already been filed, APPLICANT IS GIVEN A NON-EXTENDABLE PERIOD OF THE LONGER OF ONE MONTH OR THIRTY DAYS FROM THIS INTERVIEW DATE, OR THE MAILING DATE OF THIS INTERVIEW SUMMARY FORM, WHICHEVER IS LATER, TO FILE A STATEMENT OF THE SUBSTANCE OF THE INTERVIEW. See Summary of Record of Interview requirements on reverse side or on attached sheet.

/Hyung S. SOUGH/
Supervisory Patent Examiner, Art Unit 2194

Summary of Record of Interview Requirements

Manual of Patent Examining Procedure (MPEP), Section 713.04, Substance of Interview Must be Made of Record
A complete written statement as to the substance of any face-to-face, video conference, or telephone interview with regard to an application must be made of record in the application whether or not an agreement with the examiner was reached at the interview.

Title 37 Code of Federal Regulations (CFR) § 1.133 Interviews
Paragraph (b)

In every instance where reconsideration is requested in view of an interview with an examiner, a complete written statement of the reasons presented at the interview as warranting favorable action must be filed by the applicant. An interview does not remove the necessity for reply to Office action as specified in §§ 1.111, 1.135. (35 U.S.C. 132)

37 CFR §1.2 Business to be transacted in writing.
All business with the Patent or Trademark Office should be transacted in writing. The personal attendance of applicants or their attorneys or agents at the Patent and Trademark Office is unnecessary. The action of the Patent and Trademark Office will be based exclusively on the written record in the Office. No attention will be paid to any alleged oral promise, stipulation, or understanding in relation to which there is disagreement or doubt.

The action of the Patent and Trademark Office cannot be based exclusively on the written record in the Office if that record is itself incomplete through the failure to record the substance of interviews.

It is the responsibility of the applicant or the attorney or agent to make the substance of an interview of record in the application file, unless the examiner indicates he or she will do so. It is the examiner's responsibility to see that such a record is made and to correct material inaccuracies which bear directly on the question of patentability.

Examiners must complete an Interview Summary Form for each interview held where a matter of substance has been discussed during the interview by checking the appropriate boxes and filling in the blanks. Discussions regarding only procedural matters, directed solely to restriction requirements for which interview recordation is otherwise provided for in Section 812.01 of the Manual of Patent Examining Procedure, or pointing out typographical errors or unreadable script in Office actions or the like, are excluded from the interview recordation procedures below. Where the substance of an interview is completely recorded in an Examiners Amendment, no separate Interview Summary Record is required.

The Interview Summary Form shall be given an appropriate Paper No., placed in the right hand portion of the file, and listed on the "Contents" section of the file wrapper. In a personal interview, a duplicate of the Form is given to the applicant (or attorney or agent) at the conclusion of the interview. In the case of a telephone or video-conference interview, the copy is mailed to the applicant's correspondence address either with or prior to the next official communication. If additional correspondence from the examiner is not likely before an allowance or if other circumstances dictate, the Form should be mailed promptly after the interview rather than with the next official communication.

The Form provides for recordation of the following information:
- Application Number (Series Code and Serial Number)
- Name of applicant
- Name of examiner
- Date of interview
- Type of interview (telephonic, video-conference, or personal)
- Name of participant(s) (applicant, attorney or agent, examiner, other PTO personnel, etc.)
- An indication whether or not an exhibit was shown or a demonstration conducted
- An identification of the specific prior art discussed
- An indication whether an agreement was reached and if so, a description of the general nature of the agreement (may be by attachment of a copy of amendments or claims agreed as being allowable). Note: Agreement as to allowability is tentative and does not restrict further action by the examiner to the contrary.
- The signature of the examiner who conducted the interview (if Form is not an attachment to a signed Office action)

It is desirable that the examiner orally remind the applicant of his or her obligation to record the substance of the interview of each case. It should be noted, however, that the Interview Summary Form will not normally be considered a complete and proper recordation of the interview unless it includes, or is supplemented by the applicant or the examiner to include, all of the applicable items required below concerning the substance of the interview.

A complete and proper recordation of the substance of any interview should include at least the following applicable items:
1) A brief description of the nature of any exhibit shown or any demonstration conducted,
2) an identification of the claims discussed,
3) an identification of the specific prior art discussed,
4) an identification of the principal proposed amendments of a substantive nature discussed, unless these are already described on the Interview Summary Form completed by the Examiner,
5) a brief identification of the general thrust of the principal arguments presented to the examiner,
   (The identification of arguments need not be lengthy or elaborate. A verbatim or highly detailed description of the arguments is not required. The identification of the arguments is sufficient if the general nature or thrust of the principal arguments made to the examiner can be understood in the context of the application file. Of course, the applicant may desire to emphasize and fully describe those arguments which he or she feels were or might be persuasive to the examiner.)
6) a general indication of any other pertinent matters discussed, and
7) if appropriate, the general results or outcome of the interview unless already described in the Interview Summary Form completed by the examiner.

Examiners are expected to carefully review the applicant's record of the substance of an interview. If the record is not complete and accurate, the examiner will give the applicant an extendable one month time period to correct the record.

Examiner to Check for Accuracy

If the claims are allowable for other reasons of record, the examiner should send a letter setting forth the examiner's version of the statement attributed to him or her. If the record is complete and accurate, the examiner should place the indication, "Interview Record OK" on the paper recording the substance of the interview along with the date and the examiner's initials.

.

REQUEST FOR CONTINUED EXAMINATION (RCE) TRANSMITTAL		
Address to: **Mail Stop RCE** **Commissioner for Patents** **P.O. Box 1450** **Alexandria, VA 22313-1450**	Application Number	10/946,536
	Filing Date	September 21, 2004
	First Named Inventor	Donn Rochette
	Group Art Unit	2194
	Examiner Name	Syed Roni
	Attorney Docket Number	78803(120-2 US)

This is a Request for Continued Examination (RCE) under 37 C.F.R. § 1.114 of the above-identified application.
Request for Continued Examination (RCE) practice under 37 CFR 1.114 does not apply to any utility or plant application filed prior to June 8, 1995, or to any design application. See instruction Sheet for RCEs (not to be submitted to the USPTO) on page 2.

1. Submission required under 37 C.F.R. § 1.114
   a. _____ Previously submitted
      i. _____ Consider the amendments/reply under 37 CFR § 1.116 previously filed on _____
      (Any unentered amendment(s) referred to above will be entered).
      ii. _____ Consider the arguments in the Appeal Brief or Reply Brief previously filed on _____
      iii. _____ Other _____
   b. __X__ Enclosed
      i. __X__ Amendment/Reply
      ii. _____ Affidavit(s)/Declaration(s)
      iii. _____ Information Disclosure Statement (IDS)
      iv. _____ Other _____

2. _____ Miscellaneous
   a. _____ Suspension of action on the above-identified application is requested under 37 C.F.R. § 1.103(c) for a period of ____ months. (Period of suspension shall not exceed 3 months; Fee under 37 C.F.R. § 1.17(i) required)
   b. _____ Other _____

3. Fees (The RCE fee under 37 C.F.R. § 1.17(e) is required by 37 C.F.R. § 1.14) when the RCE is filed; Fee calculated as shown below.)
   a. _____ Checks in the amount of $_____ is enclosed.
   b. __X__ The Director is hereby authorized to charge or credit any discrepancies in fee amounts to Deposit Acct. No. 01-0484.
   c. __X__ Payment by EFS-Web.
      i. __X__ RCE fee ($810 large entity or $405 small entity) required under 37 C.F.R. § 1.17(e)
      ii. _____ Extension of time fee (37 C.F.R. § 1.137 and 1.17)
      iii. _____ Other _____

SIGNATURE OF APPLICANT, ATTORNEY OR AGENT REQUIRED

NAME	DAVID S. CARUS, Reg. No. 59,291		
SIGNATURE		DATE	July 1, 2009

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: **ROCHETTE ET AL.**	Atty. Docket No.: **78803 (120-2 US)**
Serial No. **10/946,536**	Art Unit: **4113**
Filing Date: **SEPTEMBER 21, 2004**	Examiner: **SYED A. RONI**
Confirmation No. **7612**	
For: **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**	

AMENDMENT AFTER FINAL

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Responsive to the Final Official Action of April 2, 2009, please enter the amendments and consider the remarks set out below.

In the Claims:

1. (Currently Amended) A computing system for executing a plurality of software applications comprising:

a) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

b) a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, and

ii) wherein an instance of a SLCSE provided to one or more of the plurality of software applications from the shared library is run in a context of said one or more of the plurality of software applications without being shared with other of the plurality of software applications and where one or more other of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing essentially the same function.

2. (Original) A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored

in the shared library run simultaneously within the operating system.

3. (Original) A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing essentially the same function as SLCSEs remain in the operating system kernel.

4. (Previously Presented) A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. (Original) A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between a SLCSE in the context of an application program and a device driver.

6. (Previously Presented) A computing system as defined in claim 1, wherein an SLCSE related to a predetermined function is provided to a first of the plurality of software applications for running first instance of the SLCSE, and wherein an SLCSE for performing essentially a same function is provided to a second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

7. (Original) A computing system according to claim 5 wherein the kernel module is adapted to provide a notification

of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

8. (Previously Presented) A computing system according to claim 7 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

9. (Original) A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

10. (Previously Presented) A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

11. (Previously Presented) A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

12. (Original) A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the

operating system kernel for device access, interrupt delivery, and virtual memory mapping.

13. (Original) A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

14. (Previously Presented) A computing system according to claim 11 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

15. (Original) A computing system according to claim 14 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

16. (Original) A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

17. (Previously Presented) A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

18. (Original) A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

19. (Original) A computer system as defined in claim 2 wherein SLCSEs are not copies of OSLCEs.

20. (Original) An operating system comprising the computing system of claim 2.

REMARKS

The Examiner is thanked for the thorough examination of the present application. The Examiner and his Supervisor are also thanked for the telephonic interview of June 3, 2009, during which the current claim rejections were discussed and wherein the Examiner agreed that claim amendments along the lines made herein to advance prosecution would define over the prior art. No new matter has been added. The patentability of the claims is discussed below.

I. The Claimed Invention

The present invention, as recited in amended independent Claim 1, for example, is directed to a computing system for executing a plurality of software applications. The computing system includes an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode. The computing system also includes a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode. Some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications. When one of the SLCSEs is accessed by one or more of the plurality of software applications, it forms a part of the one or more of the plurality of software applications. An instance of an SLCSE provided to one or more of the plurality of software applications from the shared library is run in a context of the one or more of the plurality of software applications without being shared with other of the plurality of software

applications. One or more other of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing essentially the same function.

II. The Claims are Patentable

The Examiner rejected independent Claim 1 over Cabrero et al. Cabrero et al. is directed to a computer system that uses a microkernel to execute two different tasks, for example, operating systems, and uses a common shared library. Rather than each task setting up its own libraries, a global offset table is set up for each task so that the tasks can use common shared libraries.

Independent Claim 1 has been amended to recite that some of the SLCSEs stored in the shared library are functional replicas of OSCSEs. The Examiner agreed that Cabrero et al. fails to disclose the SLCSEs stored in the shared library being functional replicas of OSCSEs. Instead, the Examiner indicated that Cabrero et al. discloses a common shared library 44 that includes addresses to the microkernel common services. (See Cabrero et al. Col. 8, lines 37-43). Nowhere does Cabrero et al. disclose the SLCSEs stored in the shared library being functional replicas of OSCSEs, or in other words, replacements. Accordingly, independent Claim 1 is patentable for at least this reason alone.

Additionally, Applicants submit that Cabrero et al. fails to disclose an instance of an SLCSE provided to one or more of the plurality of software applications from the shared library being run in a context of the one or more of the

plurality of software applications without being shared with other of the plurality of software applications. The Examiner contended that Figure 1, and Col. 5, lines 37-43, of Cabrero et al., which is reproduced below for reference, disclose the above-noted deficiency.

> Dominant personality applications 28 shown in FIG. 1, associated with the UNIX dominant personality example, are UNIX-type applications which would run on top of the UNIX operating system personality 32. The alternate personality applications 39 shown in FIG. 1, are OS/2 applications which run on top of the OS/2 alternate personality operating system 35.

As discussed during the telephonic interview, nowhere in the noted passages or anywhere else in Cabrero et al. does it disclose an instance of an SLCSE provided to one or more of the plurality of software applications from the shared library being run in a context of the one or more of the plurality of software applications without being shared with other of the plurality of software applications. Instead, Cabrero et al. merely discloses tasks 40 and 41, corresponding to the dominant and alternative personality operating systems 38, 39 accessing the common shared library 44 via an abstraction layer 45.

Indeed, as described in the Cabrero et al. Summary of the Invention section, and illustrated in Cabrero et al., Figure 2:

> A computer system employing a microkernel executes two different tasks, e.g., operating systems, yet uses common shared libraries. Rather than each task setting up its own libraries, during compile a global offset table is set up for each

task so that the tasks can use common shared libraries. An abstractions layer is established to allow the tasks to share the global offset table, and thus to use common shared libraries. (Emphasis Added).

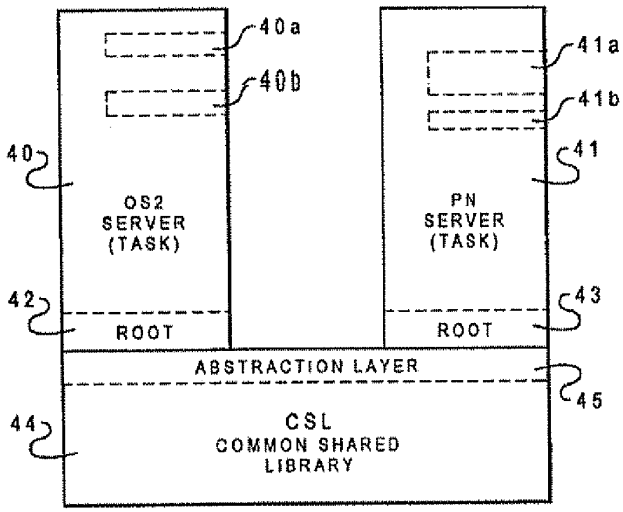


Fig. 2

Figure 2 of the Cabrero et al. '075 Patent

Indeed, nowhere in Cabrero et al. does it disclose an instance of an SLCSE provided to one or more of the plurality of software applications from the shared library being run in a context of the one or more of the plurality of software applications <u>without being shared with other of the plurality of software applications</u>.

It is submitted that independent Claim 1 is patentable over the prior art. Its respective dependent claims, which recite yet further distinguishing features, are also patentable over the prior art and require no further discussion herein.

III. Conclusion

In view of the amendments and arguments presented above, it is submitted that all of the claims are patentable. Accordingly, a Notice of Allowance is respectfully requested in due course. If the Examiner determines any remaining informalities exist, he is encouraged to contact the undersigned attorney at the telephone number listed below.

Respectfully submitted,

DAVID S. CARUS
Reg. No. 59,291
Allen, Dyer, Doppelt, Milbrath
& Gilchrist, P.A.
255 S. Orange Avenue, Suite 1401
Post Office Box 3791
Orlando, Florida 32802
407-841-2330
407-841-2343 fax
Attorney for Applicants

Electronic Patent Application Fee Transmittal

Application Number:	10946536
Filing Date:	21-Sep-2004
Title of Invention:	Computing system having user mode critical system elements as shared libraries
First Named Inventor/Applicant Name:	Donn Rochette
Filer:	David Scott Carus/Lisa Norberg
Attorney Docket Number:	78803 (120-2 US)

Filed as Small Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
Request for continued examination	2801	1	405	405
		Total in USD ($)		**405**

Electronic Acknowledgement Receipt	
EFS ID:	5626845
Application Number:	10946536
International Application Number:	
Confirmation Number:	7612
Title of Invention:	Computing system having user mode critical system elements as shared libraries
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	David Scott Carus/Lisa Norberg
Filer Authorized By:	David Scott Carus
Attorney Docket Number:	78803 (120-2 US)
Receipt Date:	01-JUL-2009
Filing Date:	21-SEP-2004
Time Stamp:	15:31:11
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Credit Card
Payment was successfully received in RAM	$405
RAM confirmation Number	1823
Deposit Account	010484
Authorized User	CARUS,DAVID S.

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		78803_RCEandAmendmentAfterFinal.pdf	383438 db5b09be7f6a67867a079f78a19c7e4670e27923	yes	12

Multipart Description/PDF files in .zip description

Document Description	Start	End
Request for Continued Examination (RCE)	1	1
Amendment After Final	2	2
Claims	3	7
Applicant Arguments/Remarks Made in an Amendment	8	12

Warnings:

Information:

2	Fee Worksheet (PTO-875)	fee-info.pdf	30085 48b8da5ab4fab57db4efae0acd5d99a853cab62f	no	2

Warnings:

Information:

Total Files Size (in bytes):	413523

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

<u>New Applications Under 35 U.S.C. 111</u>
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

<u>National Stage of an International Application under 35 U.S.C. 371</u>
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

<u>New International Application Filed with the USPTO as a Receiving Office</u>
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (07-06)
Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number	Filing Date	
10/946,536	09/21/2004	☐ To be Mailed

APPLICATION AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	SMALL ENTITY ☒ RATE ($)	SMALL ENTITY FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =		OR	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =			X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT

07/01/2009	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	* 20	Minus	** 20	= 0	X $26 =	0	OR	X $ =	
Independent (37 CFR 1.16(h))	* 1	Minus	***3	= 0	X $110 =	0	OR	X $ =	
☐ Application Size Fee (37 CFR 1.16(s))									
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
					TOTAL ADD'L FEE	0	OR	TOTAL ADD'L FEE	

AMENDMENT

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)		RATE ($)	ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
☐ Application Size Fee (37 CFR 1.16(s))									
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
					TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Legal Instrument Examiner:
/DALE HALL/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**
If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	09/21/2004	Donn Rochette	78803 (120-2 US)	7612

27975 7590 09/22/2009
ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER
RONI, SYED A

ART UNIT	PAPER NUMBER
2194	

NOTIFICATION DATE	DELIVERY MODE
09/22/2009	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

creganoa@addmg.com

PTOL-90A (Rev. 04/07)

Office Action Summary	Application No.	Applicant(s)	
	10/946,536	ROCHETTE ET AL.	
	Examiner	Art Unit	
	SYED RONI	2194	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *07/01/2009*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1 - 20* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1 - 20* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *21 September 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Continued Examination Under 37 CFR 1.114

A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on 07/01/2009 has been entered.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 1 – 9 and 11 – 20 are rejected under 35 U.S.C. 103(a) as being unpatentable over O'Rourke et al. (US 6,212,574 B1) in further view of Peek (5,481,706).

O'Rourke et al. disclose;

Regarding **claim 1**, (Currently Amended) A **computing system** [i.e., system (col. 1, lines 8 - 11)] for **executing** a **plurality** of **software applications** [i.e., software components in the user mode (col. 1, lines 8 - 11)] comprising:

a) an **operating system** [i.e., operating system (col. 1, lines 10 - 11)] **having** an **operating system kernel** [i.e., kernel mode (col. 1, line 10)] **having OS critical system elements** (**OSCSEs**) [i.e., software components (col. 1, line 10)], [i.e., drivers or filters (col. 1, line 52), (col. 4, line 52), (see figure 1 and 3)], [i.e., time critical functions (col. 1, lines 49 – 51, lines 58 - 59))] for **running** in **kernel mode** [i.e., kernel mode (col. 1, lines 66 - 67), (col. 2, lines 1 - 2)]; and,

b) **critical system elements** (**SLCSEs**) [i.e., reader proxy 64…66…68…sound rendering proxy 70 (col. 10, lines 19 - 25), (see figure 3)] stored therein **for use by** the **plurality of software applications** in **user mode** [i.e., user mode (col. 10, line 26), (see figure 3)], [i.e., software components….user mode (col. 1, lines 9 - 10)] and

i) **wherein some of the SLCSEs** stored in the shared library are **functional replicas of OSCSEs** [i.e., user mode proxy of kernel mode operations (col. 3, line 61)], [i.e., user...for kernel mode filter (col. 6, line 13), (col. 10, line 26), (see figure 3)] and are **accessible** [i.e., interconnecting (col. 5, lines 11 - 14)], [i.e., manipulating…proxies (col. 6, lines 15 - 17)], [i.e., communication with…proxies (col. 10, line 37)] to **some** of the **plurality of software applications** [i.e., user mode applications (col. 2, line 10)], [i.e., controlling agent 44 (col. 5, line 12), (col. 10, line 35), (see figure 3)], [i.e., controlling agents (col. 6, line 15)] and **when** one of the **SLCSEs** is **accessed by** one or more of the **plurality** of **software applications it forms** a **part** of the one or more of the plurality of **software applications** [i.e., controlling agent 44 (see figure 3)].

O'Rourke et al. do not disclose;

a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications wherein an instance of a SLCSE provided to one or more of the plurality of software applications from the shared library is run in a context of said one or more of the plurality of software applications without being shared with other of the plurality of software applications and where one or more other of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing essentially the same function.

However, Peek discloses;

a **shared library** [i.e., shared library 14 (col. 4, line 50), (see figure 1)] **having critical system elements** (**SLCSEs**) **stored** therein [i.e., data 24 (col. 4, line 53), (see figure 1)] for use **by** the **plurality** of **software applications** [i.e., process A 10 (see figure 1)], [i.e., process B 12 (see figure 1)] **wherein** an **instance of a SLCSE provided** to **one** or more of the **plurality** of **software applications from** the **shared library** is **run** in a **context** of said one or more of the **plurality** of **software applications** [i.e., Process A – private copy of shared library data 20 (see figure 1)] **without** being **shared** with **other** of the **plurality** of **software applications** [i.e., Process B – private copy of shared library data 22 (see figure 1)] and **where** one or more **other** of the **plurality** of **software applications** running under the operating system **have** use of a **unique instance of a corresponding critical system element** for performing essentially the **same function** [i.e., own copy of data (col. 4, lines 52 - 53)].

At the time of the invention, it would have been obvious to a person of ordinary skill in the art to modify the user mode proxy drivers for kernel mode drivers of O'Rourke et al. to be stored in a shared data structure such as a shared library as though by Peek to improve the sharing of resources among different applications and use those kernel filters as the library functions in a user mode.

O'Rourke et al. disclose;

Regarding **claim 2**, (Original) A computing system as defined in claim 1, wherein in operation, an **instance of an SLCSE** [i.e., reader proxy 64…66…68…sound rendering proxy 70 (col. 10, lines 19 - 25), (see figure 3)], [i.e., software components….user mode (col. 1, lines 9 - 10)] **run** within the **operating system** [i.e., operating system (col. 1, lines 9 - 10)].

O'Rourke et al. do not disclose;

multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

However, Peek discloses;

multiple instances of an SLCSE stored in the **shared library run simultaneously** within the operating system [i.e., Process A – private copy…data 20……Process B – private copy….data 22 (see figure 1)], [i.e., own copy of data (col. 4, lines 50 - 54)].

At the time of the invention, it would have been obvious to a person of ordinary skill in the art to modify the user mode proxy drivers for kernel mode drivers of O'Rourke et al. to include multiple copies of data in a shared library for multiple processes as

though by Peek to scale the applicability of user mode proxy of kernel mode drivers in variety of applications.

O'Rourke et al. disclose;

Regarding **claim 3**, (Original) A computing system according to claim 1 **wherein OSCSEs corresponding to** and capable of **performing** essentially the **same function as SLCSEs** [i.e., reader proxy filter 64 which acts...reader driver 50.....and sound rendering proxy 70 which acts....sound rendering driver 58 (col. 10, lines 19 - 25), (see figure 3)] **remain** in the **operating system kernel** [i.e., kernel mode filter (col. 10, line 26), (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 4**, (Previously Presented) A computing system according to claim 1 **wherein** the one or more **SLCSEs** provided to one of the plurality of **software applications** [i.e., application program in user mode (col. 1, lines 46 - 49)], [i.e., user mode (col. 10, line 26), (see figure 3)] having **exclusive use** [i.e., filters needed....applicable to teleconferencing (col. 11, lines 39 - 41)] thereof, **use system calls** [i.e., controlled interfaces or mechanism (col. 1, lines 39 - 41)], [i.e., interfaces (col. 1, lines 46 - 49)] to **access services** [i.e., hardware and other services (col. 1, lines 39 - 41)] in the **operating system kernel** [i.e.., operating system (col. 1, lines 39 - 41)], [i.e., driver or filter (col. 1, lines 49 – 51, lines 66 - 67) and (col. 2, lines 1 - 2)].

O'Rourke et al. disclose;

Regarding **claim 5**, (Original) A computing system according to claim 1 wherein the **operating system kernel** comprises a **kernel module** adapted to **serve as** an

interface [i.e., interfaces (col. 1, lines 47 - 50)], [i.e., kernel mode/user mode boundary (col. 3, lines 2 - 3), (see figures 1 - 3)] between a **SLCSE in the context of an application program** [i.e., application program in user mode (col. 1, lines 47 - 50)] and a **device driver** [i.e., driver or filter (col. 1, lines 50 - 52)], [i.e., reader driver 50, effect filter 54 (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 6**, (Previously Presented) A computing system as defined in claim 1, **wherein** an **SLCSE related** to a **predetermined function** [i.e., process audio data (col. 11, lines 39 - 41)] is **provided** to a **first of the plurality of software applications** [i.e., multimedia (col. 11, lines 37 - 41)] for **running first instance of the SLCSE** [i.e., filters (col. 11, lines 39 - 41), (see figure 3)], and **wherein** an **SLCSE** for **performing essentially** a **same function** [i.e., applicable (col. .11, lines 37 - 41)] is **provided** to a **second of the plurality of software applications** [i.e., teleconferencing (col. 11, lines 37 - 41)] for **running** a **second instance of the SLCSE simultaneously** [i.e., filters (col. 11, lines 39 - 41), (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 7**, (Original) A computing system according to claim 5 wherein the **kernel module** [i.e., proxy filters (col. 11, lines 1 - 2)] is adapted to **provide** [i.e., send (col. 11, line 1)] a **notification of an event** [i.e., notifications (col. 11, line 2)] to an **SLCSE** running in the **context of an application program** [i.e., controlling agent 44 (col. 11, line 2), (see figure 3)], [i.e., corresponding proxy filter (col. 11, lines 1 - 5)], wherein the **event** is an **asynchronous event** and requires **information** to be **passed**

to the **SLCSE** from **outside** the **application** [i.e., events would include end of processing,…..a data overrun situation (col. 9, lines 36 - 38)].

O'Rourke et al. disclose;

Regarding **claim 8**, (Previously Presented) A computing system according to claim 7 wherein a **handler** is provided for **notifying** [i.e., notifications (col. 11, line 2)] the **SLCSE in the context of one of the plurality of software applications** through the **use of** an **up call mechanism** [i.e., user mode proxy of kernel mode operations (col. 3, line 61)], [i.e., user...for kernel mode filter (col. 6, line 13), (col. 10, line 26), (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 9**, (Original) A computing system according to claim 7 wherein the up call mechanism in operation, **executes instructions** from an **SLCSE resident** in **user mode space**, in **kernel mode** [i.e., user mode proxy of kernel mode operations (col. 3, line 61)], [i.e., user...for kernel mode filter (col. 6, line 13), (col. 10, line 26), (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 11**, (Previously Presented) A computing system according to claim 2 wherein **SLCSEs** stored in the shared library are **linked** to **particular software applications** of the **plurality of software applications** as the particular software applications are loaded such that the particular software applications have a link that provides **unique access** to a **unique instance of a CSE** [i.e., filters needed....applicable to teleconferencing (col. 11, lines 39 - 41)].

O'Rourke et al. do not disclose;

SLCSEs are stored in the shared library

However, Peek discloses;

a **shared library** [i.e., shared library 14 (col. 4, line 50), (see figure 1)] **having critical system elements** (**SLCSEs**) **stored** therein [i.e., data 24 (col. 4, line 53), (see figure 1)]

At the time of the invention, it would have been obvious to a person of ordinary skill in the art to modify the user mode proxy drivers for kernel mode drivers of O'Rourke et al. to be stored in a shared data structure such as a shared library as though by Peek to improve the sharing of resources among different applications and use those kernel filters as the library functions in a user mode.

O'Rourke et al. disclose;

Regarding **claim 12**, (Original) A computing system according to claim 2 wherein the SLCSEs utilize **kernel services** [i.e., software components (col. 1, line 10)], [i.e., drivers or filters (col. 1, line 52), (col. 4, line 52), (see figure 1 and 3)] supplied by the operating system kernel for **device access** [i.e., speaker 62 (see figure 3)], **interrupt delivery** [i.e., notifications (col. 11, line 2), (col. 9, lines 36 - 38)], and **virtual memory mapping** [i.e., filter graph (col. 11, line 8), (col. 11, line 12)], [i.e., disk driver 48 (see figure 2)].

O'Rourke et al. disclose;

Regarding **claim 13**, (Original) A computing system according to claim 1, wherein **SLCSEs** include **services** related to **at least one of**, network protocol

processes, and the **management of files** [i.e., stream of data…processing functions (col. 2, lines 22 - 23)], [i.e., reading, decompressing and rendering of audio data (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 14**, (Previously Presented) A computing system according to claim 11 wherein some **SLCSEs** are **modified** [i.e., filters needed to process audio data (col. 11, lines 39 - 40)] for a **particular one** of the plurality of **software applications** [i.e., multimedia (col. 11, line 40)].

O'Rourke et al. disclose;

Regarding **claim 15**, (Original) A computing system according to claim 14 wherein the **SLCSEs** [i.e., reader proxy 64…66…68…sound rendering proxy 70 (col. 10, lines 19 - 25), (see figure 3)] that are **application specific**, **reside** in **user mode** [i.e., software components in the user mode (col. 1, lines 8 - 10)], while **critical system elements** [i.e., software components (col. 1, line 10)], [i.e., drivers or filters (col. 1, line 52), (col. 4, line 52), (see figure 1 and 3)], which are **platform specific**, **reside** in the **operating system kernel** [i.e., software components in the kernel mode (col. 1, lines 10 - 11)].

O'Rourke et al. disclose;

Regarding **claim 16**, (Original) A computing system according to claim 5 wherein the **kernel module** is adapted to **enable data exchange** [i.e., interfaces (col. 1, lines 47 - 50)], [i.e., kernel mode/user mode boundary (col. 3, lines 2 - 3), (see figures 1 - 3)] **between** the **SLCSEs in user mode** [i.e., application program in user mode (col. 1,

lines 47 - 50)] and a **device driver in kernel mode** [i.e., driver or filter (col. 1, lines 50 - 52)], [i.e., reader driver 50, effect filter 54 (see figure 3)], and **wherein** the **exchange uses mapping of virtual memory** [i.e., filter graph (col. 11, line 8), (col. 11, line 12)] such that **data** is **transferred both** from the **SLCSEs in user mode** to the **device driver in kernel mode** and from the **device driver in kernel mode** to the **SLCSEs in user mode** [i.e., arrows are going from user mode to kernel mode and from kernel to use mode (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 17**, (Previously Presented) A computing system according to claim 1 wherein **SLCSEs** form a part of at least some of the plurality of software applications, by **being linked** thereto [i.e., controlling agent 44 (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 18**, (Original) A computing system according to claim 2 wherein the **SLCSEs** utilize **kernel services** supplied by the operating system kernel for **device access** [i.e., speaker 62 (see figure 3)], **interrupt delivery** [i.e., notifications (col. 11, line 2), (col. 9, lines 36 - 38)], and **virtual memory mapping** [i.e., filter graph (col. 11, line 8), (col. 11, line 12)], [i.e., disk driver 48 (see figure 2)] and **otherwise** execute **without interaction** from the **operating system kernel** [i.e., software components in the kernel mode (col. 1, lines 10 - 11)].

O'Rourke et al. disclose;

Regarding **claim 19**, (Original) A computer system as defined in claim 2 wherein **SLCSEs** are **not copies of OSLCEs** [i.e., user mode proxy of kernel mode operations

(col. 3, line 61)], [i.e., user...for kernel mode filter (col. 6, line 13), (col. 10, line 26), (see figure 3)].

O'Rourke et al. disclose;

Regarding **claim 20**, (Original) An **operating system** comprising the computing system of claim 2 [i.e., operating system (col. 1, lines 10 - 11)].

Claim 10 is rejected under 35 U.S.C. 103(a) as being unpatentable over O'Rourke et al. (US 6,212,574 B1) and Peek (5,481,706) as applied to claim 2 above, and further in view of Desoli et al. (US 2004/0025165 A1).

O'Rourke et al. disclose;

Regarding **claim 10**, (Previously Presented) A computing system according to claim 2, **plurality of software applications** [i.e., software components….mode (col. 1, line 9)] **access** [i.e., interaction (col. 1, lines 8 - 11)] to **operating system services** [i.e., software components…..operating system (col. 1, lines 10 -11)].

O'Rourke et al. and Peek do not disclose;

a function overlay is used to provide one of the plurality of software applications access to operating system services

However, Desoli et al. disclose;

a **function overlay** [i.e., operating system intercept module 304 (page 3, par 0026), (see figure 3)] is used to **provide one** of the **plurality of software applications**

[i.e., application 102 (page 3, par 0026), (see figure 3)] **access** to **operating system services** [i.e., operating system 104 (page 3, par 0026), (see figure 3), (page 1, par 0006)].

At the time of the invention, it would have been obvious to a person of ordinary skill in the art to modify the interaction between software components in the user mode and software components in the kernel mode of operating system of O'Rourke et al. and shared library of Peek to include operating system intercept module to intercept calls such system calls from application to an operating system as though by Desoli et al. to manage access of different applications to operating system services.

Response to Arguments

Applicant's arguments with respect to the pending claims have been considered but are moot in view of the new ground(s) of rejection.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to SYED RONI whose telephone number is (571)270-7806. The examiner can normally be reached on M - F (8:30 am - 5:00 pm).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Hyung Sub Sough (Sam) can be reached on (571) 272 - 6799. The fax

phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/SYED RONI/
Examiner, Art Unit 2194

/Hyung S. Sough/
Supervisory Patent Examiner, Art Unit 2194
09/17/09

Notice of References Cited	Application/Control No. 10/946,536	Applicant(s)/Patent Under Reexamination ROCHETTE ET AL.	
	Examiner SYED RONI	Art Unit 2194	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-6,212,574 B1	04-2001	O'Rourke et al.	719/321
*	B	US-5,481,706 A	01-1996	Peek, Jeffrey S.	710/200
*	C	US-2004/0025165 A1	02-2004	Desoli et al.	719/310
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Index of Claims	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10946536	ROCHETTE ET AL.
	Examiner	**Art Unit**
	SYED RONI	4113

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	11/06/2008	09/14/2009							
	1	✓	✓							
	2	✓	✓							
	3	✓	✓							
	4	✓	✓							
	5	✓	✓							
	6	✓	✓							
	7	✓	✓							
	8	✓	✓							
	9	✓	✓							
	10	✓	✓							
	11	✓	✓							
	12	✓	✓							
	13	✓	✓							
	14	✓	✓							
	15	✓	✓							
	16	✓	✓							
	17	✓	✓							
	18	✓	✓							
	19	✓	✓							
	20	✓	✓							

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10946536	ROCHETTE ET AL.
	Examiner	**Art Unit**
	SYED RONI	4113

SEARCHED

Class	Subclass	Date	Examiner
719	310	10/30/2008	SR

SEARCH NOTES

Search Notes	Date	Examiner
East Search notes attached	11/6/2008	SR

INTERFERENCE SEARCH

Class	Subclass	Date	Examiner

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIB DATA SHEET

CONFIRMATION NO. 7612

SERIAL NUMBER	FILING or 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
10/946,536	09/21/2004 **RULE**	718	2194	78803 (120-2 US)

APPLICANTS /SR/
Donn Rochette, Fenton, IA;
Paul O'Leary, Kanata, CANADA;
Dean Huffman, Kanata, CANADA;

**** CONTINUING DATA **************************** /SR/
This appln claims benefit of 60/504,213 09/22/2003

**** FOREIGN APPLICATIONS **************************** /SR/
none

**** IF REQUIRED, FOREIGN FILING LICENSE GRANTED **** ** SMALL ENTITY **
11/05/2004

Foreign Priority claimed	☐ Yes ☑ No		STATE OR COUNTRY	SHEETS DRAWINGS	TOTAL CLAIMS	INDEPENDENT CLAIMS
35 USC 119(a-d) conditions met	☐ Yes ☑ No	☐ Met after Allowance				
Verified and Acknowledged	/Syed Roni/ Examiner's Signature	Initials	IA	7	20	1

ADDRESS

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791
UNITED STATES

TITLE

Computing system having user mode critical system elements as shared libraries

FILING FEE RECEIVED		
385	FEES: Authority has been given in Paper No._____ to charge/credit DEPOSIT ACCOUNT No._____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit

EAST Search History

EAST Search History (Prior Art)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S75	1	("20050066303").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2009/09/10 11:39
S78	34	liberat$3 with kernel	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 12:08
S79	24	(@ad<"20030922") and S78	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 12:10
S80	235	(liberating with application)	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 12:12
S89	146005	(liberating encapsulating independence with application software) same (kernel with OS "operating system" underlying "user space")	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 12:55
S90	144986	(liberating encapsulating with application software) same (kernel with OS "operating system" underlying "user space")	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 12:56
S92	10980	(virtual virtualization) same S89	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:01
S93	4792	(@ad<"20030922") and S92	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:01
S95	10793	(virtual virtualization with application) same S89	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:02
S96	10749	(virtual virtualization near2 application) same S89	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:05
S97	11433	(virtual virtualization near2 application near independen $2 dependencies) same S89	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:06
S98	966	((virtual virtualization near2 application) near independen $2 dependencies) same S89	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:07
S99	461	(@ad<"20030922") and S98	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:07

S107	5456	(application near virtua$8) same (independen$4 dependencies seperate with underlying infrastructure OS operating application)	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:45
S108	5456	(application near virtua$8) with (independen$4 dependencies seperate with underlying infrastructure OS operating application)	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:45
S109	54	(application near virtua$8) with ((independen$4 dependencies seperate) with (underlying infrastructure OS operating application))	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:45
S110	24	(@ad<"20030922") and S109	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/10 13:45
S121	6	("5734903" | "5771383" | "5842226" | "6260075" | "6308247" | "6397262").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2009/09/10 16:06
S123	16	("5515538" | "5781550" | "5832513" | "5872963" | "6011803" | "6034963" | "6179489" | "6246683" | "6336140" | "6389479" | "6427173" | "6483840" | "6575746" | "6625650" | "6658480" | "6678746").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2009/09/10 16:15
S136	13	("5247678" | "5291601" | "5339422" | "5369766" | "5379431" | "5404529" | "5414854" | "5481719" | "5566346" | "5574915" | "5613120" | "5822787" | "5835743").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2009/09/10 16:55
S140	19	("4993017" | "5485579" | "5526521" | "5721922" | "5729710" | "5742825" | "5745759" | "5764984" | "5771383" | "5835764" | "5838968" | "5903752" | "5995745" | "6092095" | "6167425" | "6192514" | "6424988" | "6466962" | "6507861").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2009/09/10 17:01
S159	213762	(kernel near context switch transition pass) with user	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:35
S160	173	(kernel near (context switch transition pass)) with user	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:35

S161	72	(@ad<"20030922") and S160	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:35
S162	3742	"shared library"	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:46
S163	2093	"shared library" with application	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:46
S164	251	("shared library" with application) same (kernel user micro$1kernel microkernel)	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:47
S165	23	("shared library" with application) same ((kernel micro$1kernel microkernel) with user)	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:47
S166	16	(@ad<"20030922") and S165	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:48
S167	144	(@ad<"20030922") and S164	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2009/09/11 14:54
S168	11	("4663709" | "4847754" | "5047919" | "5109511" | "5226143" | "5230070" | "5305448" | "5339415" | "5339427" | "5353418" | "5375241").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2009/09/11 15:52

9/14/2009 11:22:22 AM
C:\ Documents and Settings\ sroni\ My Documents\ EAST\ Workspaces old\ 10946536.wsp

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: **ROCHETTE ET AL.**	)	Atty. Docket No.: **78803 (120-2 US)**
Serial No. **10/946,536**	)	Art Unit: **4113**
Filing Date: **SEPTEMBER 21, 2004**	)	Examiner: **SYED A. RONI**
Confirmation No. **7612**	)	
For: **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**	)	

RESPONSE

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Responsive to the Official Action of September 22, 2009, please consider the remarks set out below.

In the Claims:

1. (Previously Presented) A computing system for executing a plurality of software applications comprising:

a) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

b) a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, and

ii) wherein an instance of an SLCSE provided to one or more of the plurality of software applications from the shared library is run in a context of said one or more of the plurality of software applications without being shared with other of the plurality of software applications and where one or more other of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing essentially the same function.

2. (Original) A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored

in the shared library run simultaneously within the operating system.

3. (Original) A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing essentially the same function as SLCSEs remain in the operating system kernel.

4. (Previously Presented) A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. (Currently Amended) A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between ~~a~~ an SLCSE in the context of an application program and a device driver.

6. (Previously Presented) A computing system as defined in claim 1, wherein an SLCSE related to a predetermined function is provided to a first of the plurality of software applications for running first instance of the SLCSE, and wherein an SLCSE for performing essentially a same function is provided to a second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

7. (Original) A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

8. (Previously Presented) A computing system according to claim 7 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

9. (Original) A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

10. (Previously Presented) A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

11. (Previously Presented) A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

12. (Original) A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

13. (Original) A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

14. (Previously Presented) A computing system according to claim 11 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

15. (Original) A computing system according to claim 14 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

16. (Original) A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

17. (Previously Presented) A computing system according to claim 1 wherein SLCSEs form a part of at least some

of the plurality of software applications, by being linked thereto.

18. (Original) A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

19. (Original) A computer system as defined in claim 2 wherein SLCSEs are not copies of OSLCEs.

20. (Original) An operating system comprising the computing system of claim 2.

REMARKS

The Examiner is thanked for the thorough examination of the present application. The Examiner is also thanked for properly withdrawing his prior rejection. Dependent Claim 5 has been amended to correct a minor informality. The patentability of the claims is discussed below.

I. The Claimed Invention

The present invention, as recited in independent Claim 1, for example, is directed to a computing system for executing a plurality of software applications. The computing system includes an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode. The computing system also includes a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode. Some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications. When one of the SLCSEs is accessed by one or more of the plurality of software applications, it forms a part of the one or more of the plurality of software applications. An instance of an SLCSE provided to one or more of the plurality of software applications from the shared library is run in a context of the one or more of the plurality of software applications without being shared with other of the plurality of software applications. One or more other of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing essentially the same function.

II. The Claims Are Patentable

The Examiner rejected independent Claim 1 over a combination of O'Rourke et al. and Peek. O'Rourke et al. is directed to a system that proxies software components in a kernel mode via software components in a user mode. The Examiner correctly recognized that O'Rourke et al. fails to disclose a shared library having critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and wherein an instance of an SLCSE provided to one or more of the plurality of software applications from the shared library is run in a context of the one or more of the plurality of software applications without being shared with other of the plurality of software applications and where one or more other of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing essentially the same function.

The Examiner turned to Peek for these critical deficiencies. Peek is directed to libraries for use in a multithreaded computer environment that must be thread-safe and that cannot be recoded. Peek discloses identifying the libraries and repackaging them so that library functions are accessible by multiple threads without extensive library modifications.

Applicants submit the Examiner mischaracterized O'Rourke et al. as it fails to disclose some of the SLCSEs stored in the shared library being functional replicas of OSCSEs. The Examiner contended that O'Rourke et al., Col. 3,

line 61, Col. 6, line 13, and Col. 10, line 26, disclose some of the SLCSEs stored in the shared library are functional replicas of OSCSEs. Nowhere in Col. 3, line 61, Col. 6, line 13, and Col. 10, line 26, does it disclose some of the SLCSEs stored in the shared library being functional replicas of OSCSEs. Instead, Col. 3, line 61, Col. 6, line 13, and Col. 10, line 26, disclose a user mode proxy of kernel mode operations or kernel mode filters or drivers. More particularly, a software layer is formed on top of a kernel mode graph and allows a controller agent to manipulate a particular kernel mode filter by manipulating a user mode proxy of that particular kernel mode filter. (See O'Rourke et al., Col. 4, lines 5-9).

In other words, O'Rourke et al. discloses providing a generic proxy object that may be used for virtually all kernel mode filters either unchanged or through various extension mechanisms, or more simply providing a user mode <u>proxy</u> filter for a kernel mode filter. (See O'Rourke et al., Col. 6, lines 12-19, and Col. 10, lines 12-33). Indeed, the user mode proxy filter is not a functional replica, but merely acts as an intermediary to the kernel mode filters. Accordingly, independent Claim 1 is patentable for at least this reason.

Applicants further submit that the Examiner further mischaracterized O'Rourke et al. in that it fails to disclose that when one of the SLCSEs is accessed by one or more of the plurality of software applications, it forms a part of the one or more of the plurality of software applications. The Examiner contended that the O'Rourke et al. controlling agent **44** discloses that when one of the SLCSEs is accessed by one or more of the plurality of software applications, it forms a part of

the one or more of the plurality of software applications. The O'Rourke et al. controlling agent **44** queries "the drivers in order to identify data formats and connection formats in order to interconnect kernel mode filters to create a filter graph." "Controlling agent **44** will also receive notification of important events so that it may exercise control as necessary. Examples of such events would include end of processing, a data starvation situation, a data overrun situation, and so forth." (See O'Rourke et al., Col. 9, lines 28-38; See also O'Rourke et al., Col. 10, lines 33-40, for example). Indeed, the controlling agent **44** fails to disclose that when one of the SLCSEs is accessed by one or more of the plurality of software applications, it forms a part of the one or more of the plurality of software applications. In other words, the SLCSEs literally form part of the application. SLCSEs reside in the same address space as application code, in contrast to a proxy that is exclusive of the application.

Applicants further submit that the Examiner's proposed combination of references is improper in that a person having ordinary skill in the art would not turn to the repackaging of shared libraries of Peek in an attempt to combine with the proxying of software components in a kernel mode via a software component in a user mode. More particularly, O'Rourke discloses that adding proxy filters for kernel mode filters provides several key benefits, for example, "the controlling agent **44** may manipulate and communicate with a particular kernel mode filter simply by manipulating or communicating with its proxy. Thus, user mode proxies of kernel mode filters allow a robust, familiar interface to be presented to a particular controlling

agent while, simultaneously, allowing the controlling agent to take advantage of all the benefits of a kernel mode streaming architecture." (See O'Rourke et al., Col. 10, lines 33-41). In stark contrast, Peek is concerned with maintaining the integrity of a shared library that is accessible to multiple threads. Indeed, a person having ordinary skill in the art would not turn to the shared library teachings of Peek to combine with the kernel communication teachings of O'Rourke et al.

Additionally, storing some of the SLCSEs in the shared library as functional replicas of OSCSEs, as recited independent Claim 1, for example, is particularly advantageous in multiple operating system environments. This is in contrast to the teaching of both O'Rourke et al. and Peek. Still further, the present invention, as recited in independent Claim 1, for example, advantageously provides the ability to create unique environments for an application to execute within or by the SLCSEs, which is also in contrast to both O'Rourke et al. and Peek.

Still further, the Examiner contends that a person having ordinary skill in the art would modify the user mode drivers of O'Rourke et al. to be stored in a shared data structure such as a shared library as in Peek to improve the sharing of resources among different applications and use those kernel filters as the library functions in a user mode. Applicants submit that Peek fails to teach using a shared library to improve resource sharing, but rather teaches reducing resources required for preventing data corruption using a shared library. (See Peek, Col. 5, line 25 - Col. 6, line 5). Moreover, Applicants submit that it is not even possible to

place user mode drivers in the libraries, as suggested. Indeed, O'Rourke et al. fails to disclose a shared library, and thus, Applicants submit that any motivation to combine Peek with O'Rourke et al. comes from Applicants' own Specification, paragraphs 4-7, for example. Accordingly, the Examiner's combination of references is improper, and independent Claim 1 is patentable also for this reason.

It is submitted that independent Claim 1 is patentable over the prior art. Its respective dependent claims, which recite yet further distinguishing features, are also patentable over the prior art for at least the reasons set forth above. Notwithstanding the reasons set forth above, further arguments in support of the patentability of the dependent claims are provided below.

III. The Dependent Claims Are Patentable

A. Dependent Claim 2 Is Patentable

Dependent Claim 2 recites the multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system. Indeed, simultaneously running multiple instances of an SLCSE within the operating system is possible if replicas of OS critical system elements are provided. Neither O'Rourke and Peek disclose this. Allowing simultaneous operation advantageously allows multiple software applications to run at the same time or in parallel without deleterious results that would otherwise occur such as a first instance affecting a second instance or conflicts of that nature. Accordingly, dependent Claim 2 is patentable for these reasons also.

B. Dependent Claim 3 Is Patentable

Dependent Claim 3 recites the OSCSEs corresponding to and being capable of performing essentially the same function as SLCSEs remain in the operating system kernel. This advantageously provides multiple OS environments within a single OS. This is in contrast to a virtual machine (VM), for example, VmWare, where a VM enables multiple OSs to exist on the same hardware. However, they do it at a cost of duplicating the entire OS, including the kernel. OSCSEs advantageously allow for multiple OS environments to coexist using the same kernel, which increases efficiency.

The Examiner contended that O'Rourke et al. discloses the OSCSEs corresponding to and being capable of performing essentially the same function as SLCSEs remain in the operating system kernel and turned to Col. 10, lines 19-26, to support his contention. Applicants submit that the Examiner mischaracterized O'Rourke et al. as it fails to disclose the OSCSEs corresponding to and being capable of performing essentially the same function as SLCSEs remain in the operating system kernel. Instead, O'Rourke et al. discloses providing a user mode proxy filter for a kernel mode filter. (See O'Rourke et al., Col. 6, lines 12-19, and Col. 10, lines 12-33). Accordingly, dependent Claim 3 is patentable for these reasons also.

C. Dependent Claim 4 Is Patentable

Dependent Claim 4 recites one or more SLCSEs provided to one of the plurality of software applications having

exclusive use thereof, use system calls to access services in the operating system kernel. The Examiner contended that O'Rourke et al., Col. 11, lines 39-41, disclose the recited exclusive use. Applicants submit that O'Rourke et al., Col. 11, lines 39-41, fails to disclose the recited exclusive use and instead discloses that the filters needed to process audio data for multimedia may also be applicable to teleconferencing.

Indeed, each SLCSE represents a distinct OS environment. Multiple SLCSEs create multiple OS environments. Because each SLCSE uses system calls, they are able to use the same kernel. This advantageously allows multiple OS environments where, for example, multiple web servers may exist each having their own configuration (IP address, etc.), and multiple OS environments are created using a common kernel. Accordingly, dependent Claim 4 is patentable for these reasons also.

D. Dependent Claim 5 Is Patentable

Dependent Claim 5 recites the operating system kernel includes a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver. A kernel module advantageously enables conversions that may be necessary. For example, SLCSE 1 originally used kernel version 1.1 and is placed on a kernel with version 2.2. In this scenario there are conversions to allow the system calls intended for kernel v1.1 to work effectively with kernel v 2.2. Applicants submit that O'Rourke et al. fails to disclose a kernel module adapted to serve as an interface between an SLCSE in the context of an application

program and a device driver. Instead, O'Rourke et al. discloses kernel mode/user mode transition. Accordingly, dependent Claim 5 is patentable for these reasons also.

E. Dependent Claim 6 Is Patentable

Dependent Claim 6 recites an SLCSE related to a predetermined function is provided to a first of the plurality of software applications for running a first instance of the SLCSE, and an SLCSE for performing essentially a same function is provided to a second of the plurality of software applications for running a second instance of the SLCSE simultaneously. The Examiner contended that O'Rourke et al. Col. 11, lines 37-41, discloses the above-noted recitation. Applicants submit that O'Rourke et al., Col. 11, lines 37-41, fails to disclose the recited exclusive use and instead disclose that individual kernel mode filters will have wide applicability and utility in a variety of applications. For example, many filters needed to process audio data for multimedia may also be applicable to teleconferencing.

Indeed, multiple SLCSEs equate to multiple applications. Applications that may not execute effectively as multiple instances on the same OS can do so within multiple SLCSEs. Accordingly, dependent Claim 6 is patentable for these reasons also.

F. Dependent Claim 7 Is Patentable

Dependent Claim 7 recites the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, and the event is an

asynchronous event and requires information to be passed to the SLCSE from outside the application. This is part of the mechanism that enables an SLCSE, and any application that executes using the SLCSE, to execute on a kernel that it was not originally intended to execute with. The ability to send an event to an SLCSE provides increased efficiency and flexibility, and thus it allows an application to execute that would otherwise not be able to execute.

The Examiner contended that O'Rourke et al., Col. 9, lines 36-38, disclose the event being an asynchronous event and requiring information to be passed to the SLCSE from outside the application. Applicants submit that the Examiner mischaracterized O'Rourke et al. Instead, O'Rourke et al., Col. 9, lines 36-38, disclose a controlling agent that receives notification of important events so that it may exercise control as necessary, and examples of such events include end of processing, a data starvation situation, and a data overrun situation. Accordingly, dependent Claim 7 is patentable for these reasons also.

G. Dependent Claim 8 Is Patentable

Dependent Claim 8 recites a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism. In other words, a mechanism for creating an async event is defined. The Examiner cited to O'Rourke et al., Col. 3, line 61, Col. 6, line 13, and Col. 10, line 26, as disclosing the above-noted recitation. The Examiner's cited portions of O'Rourke et al. merely disclose a user mode proxy for kernel

mode filter. Accordingly, dependent Claim 8 is patentable for this reason also.

H. Dependent Claim 9 Is Patentable

Dependent Claim 9 recites the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode. This advantageously increases efficiency. Each transition from user mode to kernel is expensive with regard to processing requirements, as a context switch is performed. The claimed mechanism reduces the frequency of that transition.

The Examiner cited to O'Rourke et al., Col. 3, line 61, Col. 6, line 13, and Col. 10, line 26, as disclosing the above-noted recitation. The Examiner's cited portions of O'Rourke et al. merely disclose a user mode proxy for kernel mode filter. Accordingly, dependent Claim 9 is patentable for this reason also.

I. Dependent Claim 12 Is Patentable

Dependent Claim 12 recites the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping. In other words, multiple OS environments execute on a single kernel. In contrast to a VM, this provides a similar benefit, but increases efficiency and reduces complexity.

The Examiner contended that O'Rourke et al., Col. 11, lines 2, 8, and 12, Col. 9, lines 36-38, along with speaker 62, disk driver 48, disclose the SLCSEs utilize kernel services supplied by the operating system kernel for device access,

interrupt delivery, and virtual memory mapping. Applicants submit that the Examiner mischaracterized O'Rourke et al. in that Col. 11, lines 2, 8, and 12, disclose controlling agent **44** connecting proxy filters. O'Rourke et al., Col. 9, lines 36-38, discloses examples of notification events include end of processing, a data starvation situation, and a data overrun situation. Speaker 62 and disk driver 48 add nothing to the critical deficiencies of O'Rourke et al. Accordingly, dependent Claim 12 is patentable for these reasons also.

J. Dependent Claim 13 Is Patentable

Dependent Claim 13 recites SLCSEs include services related to at least one of network protocol processes, and the management of files. In other words, SLCSEs include services related to multiple concurrent network stacks or, multiple file systems.

The Examiner contended that O'Rourke et al., Figure 3 and Col. 2, lines 22-23, disclose the SLCSEs include services related to at least one of network protocol processes, and the management of files. Applicants submit that the Examiner mischaracterized the O'Rourke et al. cited portion as it fails to disclose network protocol processes. Instead, Col. 2, lines 22-23, of O'Rourke et al. disclose processing multimedia by processing a stream of data using a sequence of processing functions. Accordingly, dependent Claim 13 is patentable for this reason also.

K. Dependent Claim 14 Is Patentable

Dependent Claim 14 recites some SLCSEs are modified for a particular one of the plurality of software applications. An OS environment may be customized to a particular application without the need to customize the application. For example, a specific/customized file system that allows files to be accessed in an archive or compressed format may be provided by a custom SLCSE. In another example, a custom network protocol may be used to enable parallel processing over a custom memory interface.

The Examiner contended that O'Rourke et al., Col. 11, lines 39-40, disclose some SLCSEs are modified for a particular one of the plurality of software applications. Applicants submit that the Examiner mischaracterized O'Rourke et al. as it fails to disclose some SLCSEs are modified for a particular one of the plurality of software applications. Instead, O'Rourke et al., Col. 11, lines 39-40, disclose filters needed to process audio data for multimedia may also be applicable to teleconferencing. Accordingly, dependent Claim 14 is patentable for this reason also.

L. Dependent Claim 15 Is Patentable

Dependent Claim 15 recites the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel. The Examiner contended that O'Rourke et al. disclose the above-noted recitation and referred to Col. 1, lines 8-11, to support his contention. O'Rourke et al., Col. 1, lines 8-11, fail to disclose the SLCSEs that are application

specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel. Instead, O'Rourke et al., Col. 1, lines 8-11, disclose software components of a computer operating system and software components in a kernel mode of a computer system. Additionally, O'Rourke et al., Col. 1, line 52, discloses a layer of software, typically called a driver, existing on top of computer hardware in a system. Nowhere in the Examiner's cited portions of O'Rourke et al. does it disclose the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

Indeed, applications may use version 2 of a standard C library from an SLCSE (e.g. OS provides version 5 of the standard C library) while using device drivers for a disk subsystem provided by the kernel. Accordingly, dependent Claim 15 is patentable for this reason also.

M. Dependent Claim 16 Is Patentable

Dependent Claim 16 recites the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode. This advantageously reduces the need for transition from user mode to kernel mode.

The Examiner contended that O'Rourke et al., Col. 1, lines 47-52, and Col. 11, lines 8-12, somehow disclose the

kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode. Applicants submit that the Examiner mischaracterized the cited portions of O'Rourke et al. Instead, O'Rourke et al., Col. 1, lines 47-51, disclose the operating system providing interfaces through which an application program in user mode may access hardware or other services provided by the operating system, and thus a layer of software typically exists on top of computer hardware in the system. Col. 11, lines 8-11, disclose controlling agent **44** connecting corresponding proxy filter, and corresponding proxy filters handling the details of connecting the individual kernel mode filters into the desired filter graph. Indeed, O'Rourke et al. is silent as to the kernel module being adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and the data exchange using mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode. Accordingly, dependent Claim 16 is patentable for this reason also.

N. Dependent Claim 17 Is Patentable

Dependent Claim 17 recites SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto. In other words, applications do not change, and

they execute on an incompatible OS in the same manner as they would on a compatible OS.

Applicants submit that a person having ordinary skill in the art would not turn to Peek in an attempt to combine with O'Rourke et al. More particularly, Applicants submit that Peek fails to teach using a shared library to improve resource sharing, but rather teaches reducing resources required for preventing data corruption using a shared library. (See Peek, Col. 5, line 25 - Col. 6, line 5). Indeed, a person skilled in the art would not turn to the controlling agent **44** as in O'Rourke et al. in an attempt to arrive at the claimed invention, as recited in dependent Claim 17. O'Rourke et al. discloses using a controlling agent **44** to connect corresponding proxy filters, while Peek teaches reducing resources. In other words, Peek attempt to reduce resources, and O'Rourke et al. adds resources via the controlling agent **44**. Accordingly, dependent Claim 17 is patentable for this reason also.

O. Dependent Claim 18 Is Patentable

Dependent Claim 18 recites the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel. SLCSEs are not the same as the libraries provided by the OS. Indeed, SLCSEs are specific to the application and are independent of the OS.

The Examiner contended that O'Rourke et al., Col. 11, lines 2, 8, and 12, and Col. 9, lines 36-38, along with speaker 62, disk driver 48, disclose the SLCSEs utilize kernel services

supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping. The Examiner also contended that O'Rourke et al., Col. 1, lines 10-11, disclose executing without interaction from the operating system kernel.

Applicants submit that the Examiner mischaracterized O'Rourke et al. in that Col. 11, lines 2, 8, and 12, disclose controlling agent **44** connecting proxy filters. O'Rourke et al., Col. 9, lines 36-38, discloses examples of notification events include end of processing, a data starvation situation, and a data overrun situation. Speaker 62 and disk driver 48 add nothing to the critical deficiencies of O'Rourke et al. O'Rourke et al., Col. 1, lines 10-11, generally disclose a user mode and a kernel mode of a computer operating system, and fail to supply the above-noted deficiencies. Accordingly, dependent Claim 18 is patentable for this reason also.

III. Conclusion

In view of the arguments presented above, it is submitted that all of the claims are patentable. Accordingly, a Notice of Allowance is respectfully requested in due course. If the Examiner determines any remaining informalities exist, he is encouraged to contact the undersigned attorney at the telephone number listed below.

Respectfully submitted,

DAVID S. CARUS
Reg. No. 59,291
Allen, Dyer, Doppelt, Milbrath
& Gilchrist, P.A.
255 S. Orange Avenue, Suite 1401
Post Office Box 3791
Orlando, Florida 32802
407-841-2330
407-841-2343 fax
Attorney for Applicants

Electronic Patent Application Fee Transmittal

Application Number:	10946536
Filing Date:	21-Sep-2004
Title of Invention:	Computing system having user mode critical system elements as shared libraries
First Named Inventor/Applicant Name:	Donn Rochette
Filer:	David Scott Carus/Lisa Norberg
Attorney Docket Number:	78803 (120-2 US)

Filed as Small Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Extension - 1 month with $0 paid	2251	1	65	65

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
			Total in USD ($)	**65**

Electronic Acknowledgement Receipt	
EFS ID:	6824503
Application Number:	10946536
International Application Number:	
Confirmation Number:	7612
Title of Invention:	Computing system having user mode critical system elements as shared libraries
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	David Scott Carus/Lisa Norberg
Filer Authorized By:	David Scott Carus
Attorney Docket Number:	78803 (120-2 US)
Receipt Date:	15-JAN-2010
Filing Date:	21-SEP-2004
Time Stamp:	18:04:17
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Credit Card
Payment was successfully received in RAM	$65
RAM confirmation Number	4945
Deposit Account	010484
Authorized User	CARUS,DAVID S.

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		78803_EOTandResponse.pdf	883941 f25fedb4778aae8894f77902a9c40eb35a25fe49	yes	25

Multipart Description/PDF files in .zip description

Document Description	Start	End
Extension of Time	1	1
Amendment/Req. Reconsideration-After Non-Final Reject	2	2
Claims	3	7
Applicant Arguments/Remarks Made in an Amendment	8	25

Warnings:

Information:

2	Fee Worksheet (PTO-875)	fee-info.pdf	30116 f3e02ee386f030c4e8163b083e79feca1980c9ac	no	2

Warnings:

Information:

Total Files Size (in bytes):	914057

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: **ROCHETTE ET AL.**	Atty. Docket No.: **78803 (120-2 US)**
Serial No. **10/946,536**	Art Unit: **4113**
Filing Date: **SEPTEMBER 21, 2004**	Examiner: **SYED A. RONI**
Confirmation No. **7612**	
For: **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**	

REQUEST FOR EXTENSION OF TIME

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

The Applicants in the above-referenced application respectfully request a one-month time extension for filing the response to the Official Action mailed September 22, 2009. The extension of time fee in the amount of $65.00 is being paid by credit card on EFS-Web. The Commissioner is authorized to charge or credit any discrepancies in fee amounts to Deposit Account **01-0484.**

Respectfully submitted,

DAVID S. CARUS
Reg. No. 59,291
Allen, Dyer, Doppelt, Milbrath
& Gilchrist, P.A.
255 S. Orange Avenue, Suite 1401
Post Office Box 3791
Orlando, Florida 32802
407-841-2330
407-841-2343 fax
Attorney for Applicants

PTO/SB/06 (07-06)
Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number	Filing Date	
10/946,536	09/21/2004	☐ To be Mailed

APPLICATION AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	SMALL ENTITY ☒ RATE ($)	SMALL ENTITY FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =		OR	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =			X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT 01/15/2010

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	* 20	Minus	** 20	= 0	X $26 =	0	OR	X $ =	
Independent (37 CFR 1.16(h))	* 1	Minus	***3	= 0	X $110 =	0	OR	X $ =	
☐ Application Size Fee (37 CFR 1.16(s))									
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
					TOTAL ADD'L FEE	0	OR	TOTAL ADD'L FEE	

AMENDMENT

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)		RATE ($)	ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
☐ Application Size Fee (37 CFR 1.16(s))									
☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
					TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Legal Instrument Examiner:
/Katischa R. Wanzer/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**
If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

27975 7590 05/03/2010

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER	
RONI, SYED A	
ART UNIT	PAPER NUMBER
2194	

DATE MAILED: 05/03/2010

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	09/21/2004	Donn Rochette	78803 (120-2 US)	7612

TITLE OF INVENTION: COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	08/03/2010

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE DOES NOT REFLECT A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE IN THIS APPLICATION. IF AN ISSUE FEE HAS PREVIOUSLY BEEN PAID IN THIS APPLICATION (AS SHOWN ABOVE), THE RETURN OF PART B OF THIS FORM WILL BE CONSIDERED A REQUEST TO REAPPLY THE PREVIOUSLY PAID ISSUE FEE TOWARD THE ISSUE FEE NOW DUE.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status above is to be removed, check box 5b on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check box 5a on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and 1/2 the ISSUE FEE shown above.

II. PART B - FEE(S) TRANSMITTAL, or its equivalent, must be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted. If an equivalent of Part B is filed, a request to reapply a previously paid issue fee must be clearly made, and delays in processing may occur due to the difficulty in recognizing the paper as an equivalent of Part B.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PTOL-85 (Rev. 08/07) Approved for use through 08/31/2010.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

27975 7590 05/03/2010

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)

(Signature)

(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	09/21/2004	Donn Rochette	78803 (120-2 US)	7612

TITLE OF INVENTION: COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	08/03/2010

EXAMINER	ART UNIT	CLASS-SUBCLASS
RONI, SYED A	2194	719-310000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

❑ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

❑ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list

(1) the names of up to 3 registered patent attorneys or agents OR, alternatively,

(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1_____

2_____

3_____

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent) : ❑ Individual ❑ Corporation or other private group entity ❑ Government

4a. The following fee(s) are submitted:

❑ Issue Fee

❑ Publication Fee (No small entity discount permitted)

❑ Advance Order - # of Copies _____

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**

❑ A check is enclosed.

❑ Payment by credit card. Form PTO-2038 is attached.

❑ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number _____ (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)

❑ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27.

❑ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature _____ Date _____

Typed or printed name _____ Registration No. _____

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	09/21/2004	Donn Rochette	78803 (120-2 US)	7612

27975 7590 05/03/2010

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER
RONI, SYED A

ART UNIT	PAPER NUMBER
2194	

DATE MAILED: 05/03/2010

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 1069 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 1069 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at 1-(888)-786-0101 or (571)-272-4200.

PTOL-85 (Rev. 08/07) Approved for use through 08/31/2010.

Examiner-Initiated Interview Summary	Application No.	Applicant(s)	
	10/946,536	ROCHETTE ET AL.	
	Examiner	Art Unit	
	SYED RONI	2194	

All Participants:

(1) *SYED RONI*.

(2) *David S. Carus*.

Status of Application: _____

(3) *Sough Hyung Sub*.

(4) _____.

Date of Interview: *23 April 2010*

Time: *10 am (EST)*

Type of Interview:
☒ Telephonic
☐ Video Conference
☐ Personal (Copy given to: ☐ Applicant ☐ Applicant's representative)

Exhibit Shown or Demonstrated: ☐ Yes ☐ No
If Yes, provide a brief description: .

Part I.

Rejection(s) discussed:
101 statutory rejection, Prior Art rejection and objections.

Claims discussed:
1, 3, 6, 9, 19 and 20

Prior art documents discussed:
Elnozahy et al. (US 7,499,966 B2) and Wong et al. (US 2004/0216145 A1)

Part II.

SUBSTANCE OF INTERVIEW DESCRIBING THE GENERAL NATURE OF WHAT WAS DISCUSSED:
Applicant representative David S. Carus sent a proposed claim set and authorized EXR for Examiners's Amendment and also to replace "one or more of the plurality of software applications" with at least a first of the plurality of software applications in line 18 and "one or more other of the plurality of software applications" with at least a second of the plurality of software applications in line 23 of claim 1 respectively..Also authorized to delete the phrase "essentially" from claims 1 and 3, to cancel claims 6 and 20, to make claim 9 dependent upon claim 8, to spell out first occurrence of SLCSE in claim 1 and to replace OSLCESs with OSCSEs in claim 19.

Part III.

☐ It is not necessary for applicant to provide a separate record of the substance of the interview, since the interview directly resulted in the allowance of the application. The examiner will provide a written summary of the substance of the interview in the Notice of Allowability.
☐ It is not necessary for applicant to provide a separate record of the substance of the interview, since the interview did not result in resolution of all issues. A brief summary by the examiner appears in Part II above.

(Applicant/Applicant's Representative Signature – if appropriate)

Notice of Allowability	Application No.	Applicant(s)
	10/946,536	ROCHETTE ET AL.
	Examiner	Art Unit
	SYED RONI	2194

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *01/15/2010*.

2. ☒ The allowed claim(s) is/are *1 - 5 and 7 - 19*.

3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
      1. ☐ Certified copies of the priority documents have been received.
      2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
      3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
      1) ☐ hereto or 2) ☐ to Paper No./Mail Date _____.
   (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☐ Notice of References Cited (PTO-892)
2. ☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
3. ☐ Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date _____
4. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
5. ☐ Notice of Informal Patent Application
6. ☒ Interview Summary (PTO-413), Paper No./Mail Date *20100423* .
7. ☒ Examiner's Amendment/Comment
8. ☒ Examiner's Statement of Reasons for Allowance
9. ☐ Other _____.

DETAILED ACTION

EXAMINER'S AMENDMENT

An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview David S. Carus on 04/23/2010.

Claim 1, (Currently Amended) A computing system for executing a plurality of software applications comprising:

a) a processor;

~~a~~ b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,

~~b~~ c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, ~~and~~

ii) wherein an instance of a ~~an~~ SLCSE provided to at least a first ~~one or more~~ of the plurality of software applications from the shared library is run in a context of said at least first ~~one or more~~ of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second ~~one or more other~~ of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing ~~essentially~~ same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

Claim 3, (Currently Amended) A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing ~~essentially~~ the same function as SLCSEs remain in the operating system kernel.

Claim 6, (Canceled)

Claim 9, (Currently Amended) A computing system according to claim 8 ~~claim 7~~ wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

Claim 19, (Currently Amended) A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs ~~OSLCEs~~.

Claim 20, (Canceled)

Reasons for Allowance

Claims 1 – 5 and 7 – 19 are allowed.

The following is an Examiner's statement of reasons for allowance. None of Elnozahy et al. (hereinafter Elnozahy) (US 7,499,966 B2) and Wong et al. (hereinafter Wong) (US 2004/0216145 A1) on the record discloses "a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, and

ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of

a unique instance of a corresponding critical system element for performing same function.

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously" (Claim 1). Instead, Elnozahy discloses kernel extension device driver that are user space extensions of operating system code to minimize kernel calls by web server and Wong discloses user mode accessible copies of kernel-mode memory to facilitate a device driver to execute in user-mode while the graphics engine remains in kernel mode.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to SYED RONI whose telephone number is (571)270-7806. The examiner can normally be reached on M - F (8:30 am - 5:00 pm).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Hyung Sub Sough (Sam) can be reached on (571) 272 - 6799. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/SYED RONI/
Examiner, Art Unit 2194

/Hyung S. Sough/
Supervisory Patent Examiner, Art Unit 2194
04/26/10

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10946536	ROCHETTE ET AL.
	Examiner	**Art Unit**
	SYED RONI	4113

SEARCHED

Class	Subclass	Date	Examiner
719	310	10/30/2008	SR
719	319	4/23/2010	/SR/

SEARCH NOTES

Search Notes	Date	Examiner
East Search notes attached	11/6/2008	SR
East search attached	4/23/2010	/SR/
Inventor name search no -DP-	4/23/2010	/SR/

INTERFERENCE SEARCH

Class	Subclass	Date	Examiner
719	319	4/23/2010	/SR/

Issue Classification	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10946536	ROCHETTE ET AL.
	Examiner	**Art Unit**
	SYED RONI	2194

ORIGINAL		INTERNATIONAL CLASSIFICATION									
CLASS	SUBCLASS	CLAIMED					NON-CLAIMED				
719	310	G	0	6	F	9 / 22 (2006.01.01)					

CROSS REFERENCE(S)

CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)				
719	319				

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original
1	1	16	17												
2	2	17	18												
3	3	18	19												
4	4		20												
5	5														
	6														
6	7														
7	8														
8	9														
9	10														
10	11														
11	12														
12	13														
13	14														
14	15														
15	16														

/SYED RONI/ Examiner.Art Unit 2194 (Assistant Examiner)	(Date)	**Total Claims Allowed:** 18	
/Hyung S Sough/ Supervisory Patent Examiner.Art Unit 2194 (Primary Examiner)	04/26/2010 (Date)	O.G. Print Claim(s) 1	O.G. Print Figure 4

U.S. Patent and Trademark Office

Part of Paper No. 20100423

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIB DATA SHEET

CONFIRMATION NO. 7612

SERIAL NUMBER	FILING or 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
10/946,536	09/21/2004 **RULE**	~~718~~ 719	2194	78803 (120-2 US)

APPLICANTS
Donn Rochette, Fenton, IA; /SR/
Paul O'Leary, Kanata, CANADA;
Dean Huffman, Kanata, CANADA;

**** CONTINUING DATA ****************************
This appln claims benefit of 60/504,213 09/22/2003 /SR/

**** FOREIGN APPLICATIONS *****************************
none

**** IF REQUIRED, FOREIGN FILING LICENSE GRANTED **** ** SMALL ENTITY ** /SR/
11/05/2004

Foreign Priority claimed	☐ Yes ☑ No		STATE OR COUNTRY	SHEETS DRAWINGS	TOTAL CLAIMS	INDEPENDENT CLAIMS
35 USC 119(a-d) conditions met	☐ Yes ☑ No	☐ Met after Allowance	IA	7	20	1
Verified and Acknowledged	/Syed Roni/ Examiner's Signature	Initials				

ADDRESS
ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791
UNITED STATES

TITLE
Computing system having user mode critical system elements as shared libraries

FILING FEE RECEIVED		
385	FEES: Authority has been given in Paper No._____ to charge/credit DEPOSIT ACCOUNT No._____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit

EAST Search History

EAST Search History (Interference)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L5	44	kernel near extension with application	USPAT; UPAD	OR	ON	2010/04/23 14:44
L6	30	(@ad<"20030922") and 5	USPAT; UPAD	OR	ON	2010/04/23 14:45
L7	14	kernel near extension with (user application) adj space	USPAT; UPAD	OR	ON	2010/04/23 14:50
L8	6498	"719"/$.ccls.	USPAT; UPAD	OR	ON	2010/04/23 15:00
L9	11	kernel near extension with user adj space	USPAT; UPAD	OR	ON	2010/04/23 15:01
L10	4	8 and 9	USPAT; UPAD	OR	ON	2010/04/23 15:01
L11	354	719/319	USPAT; UPAD	OR	ON	2010/04/23 15:18
L12	625	kernel with extension	USPAT; UPAD	OR	ON	2010/04/23 15:18
L13	3	11 and 12	USPAT; UPAD	OR	ON	2010/04/23 15:18
L14	4923	kernel with user	USPAT; UPAD	OR	ON	2010/04/23 15:19
L15	42	11 and 14	USPAT; UPAD	OR	ON	2010/04/23 15:20
L16	1500	kernel with user and extension	USPAT; UPAD	OR	ON	2010/04/23 15:20
L17	16	11 and 16	USPAT; UPAD	OR	ON	2010/04/23 15:21

4/23/2010 3:37:44 PM
C:\Documents and Settings\sroni\My Documents\EAST\Workspaces old\10946536.wsp

EAST Search History

EAST Search History (Prior Art)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S198	1727	kernel with (user application) adj space	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 14:23
S199	760	kernel with (shared user) adj (library memory)	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 14:25
S200	69	S198 same S199	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 14:25
S201	30	(@ad<"20030922") and S200	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 14:25
S202	124	kernel with multi$3 near (operating adj system OS)	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 14:28
S214	33	(micro$1kernel microkernel) with (user application) near (space library dll)	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 15:40
S215	24	(@ad<"20030922") and S214	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 15:40
S216	23	(micro$1kernel microkernel) near task	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 16:00
S217	19	(@ad<"20030922") and S216	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/03/31 16:01
S218	3	("5452447" | "5481719" | "5485626").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2010/03/31 16:03
S245	199	copy near kernel	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/04/07 14:06
S251	105	(micro$1kernel microkernel) with task	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/04/07 14:34
S252	22	(micro$1kernel microkernel) with task same library	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/04/07 14:34
S253	16	(@ad<"20030922") and S252	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/04/07 14:35

S254	3	micro$1kernal with kernel	US-PGPUB; USPAT; JPO; IBM_TDB	OR	ON	2010/04/07 15:17
S289	8	((DONN) near2 (ROCHETTE)).INV.	US-PGPUB; USPAT	OR	ON	2010/04/23 13:05
S290	28	((PAUL) near2 (O''LEARY)).INV.	US-PGPUB; USPAT	OR	ON	2010/04/23 13:06
S291	5	((DEAN) near2 (HUFFMAN)).INV.	US-PGPUB; USPAT	OR	ON	2010/04/23 13:06
S292	6	(S289 S290 S291) and trigence	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2010/04/23 13:07

4/23/2010 3:40:05 PM
C:\Documents and Settings\sroni\My Documents\EAST\Workspaces old\10946536.wsp

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

27975 7590 05/03/2010

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)

EFILED (Signature)

(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	09/21/2004	Donn Rochette	78803 (120-2 US)	7612

TITLE OF INVENTION: COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	08/03/2010

EXAMINER	ART UNIT	CLASS-SUBCLASS
RONI, SYED A	2194	719-310000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

❑ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

❑ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list

(1) the names of up to 3 registered patent attorneys or agents OR, alternatively,

(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

ALLEN, DYER, DOPPELT,
1 MILBRATH & GILCHRIST, P.A.
Attorneys at Law
2 255 S. Orange Ave., Ste.1401
3 PO Box 3791
Orlando, FL 32802-3791

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: TRIGENCE CORP.

(B) RESIDENCE: (CITY and STATE OR COUNTRY): OTTAWA, ONTARIO, CANADA

Please check the appropriate assignee category or categories (will not be printed on the patent) : ❑ Individual ☒ Corporation or other private group entity ❑ Government

4a. The following fee(s) are submitted:

☒ Issue Fee

☒ Publication Fee (No small entity discount permitted)

❑ Advance Order - # of Copies _____

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**

❑ A check is enclosed.

☒ Payment by credit card. Form PTO-2038 is attached.

☒ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number 50-2810 (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)

❑ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27.

❑ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature /CHRISTOPHER F. REGAN/ Date July 13, 2010

Typed or printed name Christopher F. Regan Registration No. 34,906

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Electronic Patent Application Fee Transmittal

Application Number:	10946536
Filing Date:	21-Sep-2004
Title of Invention:	COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES
First Named Inventor/Applicant Name:	Donn Rochette
Filer:	Christopher F. Regan/joellen murphy
Attorney Docket Number:	78803 (120-2 US)

Filed as Small Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Utility Appl issue fee	2501	1	755	755
Publ. Fee- early, voluntary, or normal	1504	1	300	300

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Extension-of-Time:				
Miscellaneous:				
		Total in USD ($)		**1055**

Electronic Acknowledgement Receipt

EFS ID:	8006712
Application Number:	10946536
International Application Number:	
Confirmation Number:	7612
Title of Invention:	COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	Christopher F. Regan/joellen murphy
Filer Authorized By:	Christopher F. Regan
Attorney Docket Number:	78803 (120-2 US)
Receipt Date:	13-JUL-2010
Filing Date:	21-SEP-2004
Time Stamp:	16:45:17
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Credit Card
Payment was successfully received in RAM	$1055
RAM confirmation Number	3241
Deposit Account	
Authorized User	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)

1	Issue Fee Payment (PTO-85B)	78803_IF.pdf	117641 93e6b13155751a2f3e24fb44b95c662fe67f56a6	no	1
Warnings:					
Information:					
2	Fee Worksheet (PTO-875)	fee-info.pdf	32223 de93dc482533df69a9ba78cce47bdcad67680fd9	no	2
Warnings:					
Information:					
			Total Files Size (in bytes):	149864	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

<u>New Applications Under 35 U.S.C. 111</u>
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

<u>National Stage of an International Application under 35 U.S.C. 371</u>
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

<u>New International Application Filed with the USPTO as a Receiving Office</u>
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	ISSUE DATE	PATENT NO.	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/946,536	08/24/2010	7784058	78803 (120-2 US)	7612

27975 7590 08/04/2010
ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

ISSUE NOTIFICATION

The projected patent number and issue date are specified above.

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment is 1293 day(s). Any patent to issue from the above-identified application will include an indication of the adjustment on the front page.

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Application Assistance Unit (AAU) of the Office of Data Management (ODM) at (571)-272-4200.

APPLICANT(s) (Please see PAIR WEB site http://pair.uspto.gov for additional applicants):

Donn Rochette, Fenton, IA;
Paul O'Leary, Kanata, CANADA;
Dean Huffman, Kanata, CANADA;

IR103 (Rev. 10/09)