UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION

VIRTAMOVE, CORP., Plaintiff, v. GOOGLE LLC Defendant.	Case No. 7:24-CV-00033-ADA-DTG

PLAINTIFF'S RESPONSIVE CLAIM CONSTRUCTION BRIEF

TABLE OF CONTENTS

TABLE OF AUTHORITIES

VirtaMove and Google offer not just competing claim-construction proposals, but very different approaches to claim construction. In a case involving two patents and 15 asserted claims, Google demands construction of a matrix of 13 individual claim terms. In some cases, Google proposes inserting dozens of words taken from a popular dictionary, with no basis in the intrinsic record. In other cases, Google asserts indefiniteness, without providing clear and convincing evidence of invalidity as required by Federal Circuit law. In other cases, Google cherry-picks "lexicography" from the patent specification, distorting the patentee's good faith efforts to inform the scope and meaning of the invention. In each case, Google's proposal should be rejected.

I. Terms Primarily Appearing in U.S. Patent No. 7,519,814

A. "servers" ('814 claim 1)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
No construction necessary; plain and ordinary meaning.	physical servers

The parties dispute whether the term "server" extends to all computers that a POSITA would describe as a "server"—*i.e.*, the term's plain and ordinary meaning—or whether the term somehow excludes servers that incorporate virtual machine technology. *See* Dkt. 63 at 3-4 (arguing against an infringement theory where containers run on "virtual machines"). VirtaMove believes that no construction is necessary for two reasons.

First, the claim already makes clear that the claimed "servers" are hardware (because they include hardware components such as a "processor"). Specifically, the claim recites "a plurality of servers… wherein each server includes a processor…." '814 Patent at cl. 1. A "virtual" system cannot include hardware "a processor," such that construing "server" to mean "physical server" is simply redundant with other claim requirements.

Second, Google's attempt to exclude a system where a container is running on a virtual machine (even where that virtual machine is implemented on a physical server) is unsupported by either the claim language or any other evidence. Google's implied "no virtual machines" construction has no basis in the intrinsic or extrinsic record and should be rejected.

Google's key, if not only, evidence is the following passage from the patent specification:

> There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former ***an operating system, including files and a kernel, must be deployed for each application*** while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that ***an operating system must be licensed, managed and maintained for each application***. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow applications to be easily moved between platforms, ***without the requirement for a separate and distinct operating system for each application***.

'814 Patent at 1:51-2:3.[1] This passage does not support Google's conclusion. The specification describes the inability of conventional virtual machine technology, ***on its own***, to solve the problem of containerizing application sets. Specifically, putting each application set in its own virtual machine, as was conventional, has significant downsides, including the requirement to include an entire operating system for each individual application. The patented invention, on the other hand, allows the use of "one operating system regardless of the number of application containers deployed." *Id.* at 2:60-61.

These are the distinctions over conventional virtual machine technology that are claimed in the '814 Patent. In particular, claim 1 requires that each claimed server has an operating system

[1] All emphasis added unless otherwise noted.

with an operating system kernel, that each secure container of application software comprises application software for use with a local kernel residing permanently on one of the servers, and—critically—that the containers of application software ***cannot*** include a kernel. These limitations exclude the conventional virtual machine solution described at 1:51-2:3, because in the conventional solution each virtual machine contains its own operating system and its own kernel and thus cannot be a "secure container of application software" as claimed.

But it is not prohibited for the ***server*** to contain its own operating system, and indeed that is required. Nothing in the claim language or the specification precludes an embodiment where the claimed server corresponds to a computer using virtual machine technology, with a processor and operating system and kernel as claimed, and where a plurality of secure containers of application software as claimed (*without* their own operating system and kernel, also as claimed) are stored within the server's memory. Such an embodiment is entirely consistent with the claim context and the specification, because it continues to exclude the need for a separate and distinct operating system for each application, application set, or container. And even if this embodiment does not use a virtual machine, that incidental aspect of the embodiment cannot become a claim limitation without lexicography or disclaimer, which are absent here. *JVW Enters., Inc. v. Interact Accessories*, 424 F.3d 1324, 1335 (Fed. Cir. 2005) (Without clear and unambiguous disclaimer or lexicography, courts "do not import limitations into claims from examples or embodiments appearing only in a patent's written description, even when a specification describes very specific embodiments of the invention or even describes only a single embodiment."). "Mere criticism of a particular embodiment encompassed in the plain meaning of a claim term is not sufficient to rise to the level of clear disavowal" sufficient to define claim scope. *Thorner v. Sony Computer Ent. Am. LLC*, 669 F.3d 1362, 1366 (Fed. Cir. 2012).

Google also states that the specification teaches that virtual machine technology "provides virtual hardware." Dkt. 63 at 3. This statement has no textual support. The phrase "virtual hardware" does not appear in the patent, nor is there any plausible reference to "virtual hardware" in the cited passage or anywhere else in the specification. Again, the parties agree that a physical processor is required. In sum, Google does not and cannot explain why the inclusion of additional virtual machine technology in its servers negates infringement. Notably, Google does not allege that any of the intrinsic evidence it relies on constitutes disclaimer, which would be required for Google's attempt to deviate from plain and ordinary meaning in contending that if a container is implemented on a virtual machine which is itself implemented by a server, that container does not reside on any "server."

B. "operating system" ('814 claims 1, 10; '058 claim 1) "kernel"/"operating system kernel" ('814 claim 1; '058 claim 1)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
No construction necessary; plain and ordinary meaning.	**"operating system"**: "The software that controls the allocation and usage of hardware resources such as memory, central processing unit (CPU) time, disk space, and peripheral devices." **"kernel"/"operating system kernel"**: "The core of an operating system—the portion of the system that manages memory, files, and peripheral devices; maintains the time and date; launches applications; and allocates system resources."

Each of these terms has a plain and ordinary meaning, and the specification and claims of both the '814 and '058 Patents uses the terms in their plain and ordinary sense. Google ***admits*** that "the asserted patents use 'operating system' and 'kernel'/'operating system kernel' according to their conventional meanings." Dkt. 63 at 4. There is no reason to engage in redundant, unhelpful construction of these terms.

Google's basic argument is that construction is necessary because VirtaMove does not agree to Google's demand to insert additional, redundant, confusing language glossed from a Microsoft publication. That is not the law. The Court's obligation is to resolve actual disputes "regarding the proper ***scope*** of these claims." *O2 Micro Int'l Ltd. v. Beyond Innovation Tech. Co.*, 521 F.3d 1351, 1360 (Fed. Cir. 2008) (emphasis added). Google does not and cannot identify any dispute regarding the scope of either of these terms. VirtaMove opposes construction because these are poor definitions that will be confusing, not helpful, to a jury. For example, the extraneous non-limiting examples ("such as…"; "memory, files, and peripheral devices…") provide, at best, context rather than defining the metes and bounds of the claim scope. As another example, it is well known that software ***other*** than operating systems can "control the allocation and usage of hardware resources such as memory," as it is common for individual applications to have their own memory management capabilities. Because these terms have a plain meaning, and further because Google's proposals fail to accurately capture this plain meaning, Google's proposed constructions should be rejected.

C. "disparate computing environments" ('814 claim 1)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
Environments run by standalone or unrelated computers	indefinite

Google argues that this phrase is indefinite because the definition refers to "unrelated" computers while the claim ***requires***, in Google's own contention, the computers must be "related," creating a contradiction. This does not show indefiniteness.

As Google acknowledges, the claim context does not allow for two computers to be "unrelated" because they must be "part of a single 'system.'" In other words, Google acknowledges that the "unrelated" portion of the specification's description of "disparate computing environments" cannot fit into the broader context of the claim language. Because the claim as a whole ***undisputedly*** cannot extend to "unrelated" computers, only the "standalone" portion of that description could be relevant to the scope of the claims as a whole. *See Phillips v. AWH Corp.*, 415 F.3d 1303, 1313 (Fed. Cir. 2005) ("Importantly, the person of ordinary skill in the art is deemed to read the claim term … in the context of the particular claim in which the disputed term appears….").

Accordingly, the only relevant inquiry (in the context of the claim as a whole) is whether environments run by ***standalone*** computers is indefinite. Google presents no evidence that a POSITA would be unable to understand the boundaries of standalone computers, which is a common phrase used to indicate the ability of computers to operate independently of each other. Google's narrow focus on computers being "unrelated" (a scenario that Google acknowledges is simply inapplicable in the context of the asserted claims) ignores whether "standalone" computers can be understood to a POSITA, and Google presents no evidence at all that standalone computers would not be understood.

D. "service" ('814 claims 1, 14)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
No construction necessary; plain and ordinary meaning.	"specialized, software-based functionality provided by network servers and comprised of one or more applications"

As with "operating system," Google again simply demands to insert a redundant definition of the ordinary word "service," taken from a commercial publication from Microsoft, without identifying any reason for the construction.

"*specialized*": There is no basis to limit the scope of "service" to only "specialized" services. It is not clear what Google believes "specialized" means here, but the patent specification plainly discloses that the invention extends to all services, not merely specialized services. The patent specification states that "Examples of specific services include ***but are not limited to*** CRM (Customer Relation Management) tools, Accounting, and Inventory" ('814 Patent at 7:16-51); other examples in the specification include the remote login service "ssh" (*id.* at 10:49-50), and an "accounting/payroll service" (*id.* at 16:11-16). None of these are necessarily "specialized."

"*provided by network servers*": This phrase is confusing at best. "Servers" is a claim term, and the existing claim language recites a specific relationship between servers, containers, applications, and services.

"*comprised of one or more applications*": Setting aside the nonstandard usage "comprised of," this phrase confuses the claimed relationship between applications and services. The claim recites that each container comprises one or more executable applications, that the applications are "related to a service," and that the applications "each include an object executable… for performing a task related to the service." '814 Patent cl. 1. It is the container, not the service, that comprises applications.

E. "container" ('814 claims 1, 2, 4, 6, 9, 10, 13, 14)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
An aggregate of files required to successfully execute a set of software applications on a computing platform. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers.	An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. ***A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers.*** Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers; or above and 2:32-42

The '814 Patent specification includes a broad explanation of how a "container" fits within the context of the claimed invention. Although described as a "definition," in substance the patentee provided an encyclopedia entry, which cannot reasonably be interpreted as pure lexicography, and which would serve only to confuse the jury by substituting a single word in a claim with nearly 100 words of redundant examples of how containers may be implemented. Indeed, Google itself omits entire sentences from the supposed "definition" set forth in the specification, confirming that a POSITA would not understand the entirety of its discussion of "container" to be lexicography.

Nor could Google have shown that the "exacting" standard for lexicography is met. "To act as its own lexicographer, a patentee must 'clearly set forth a definition of the disputed claim term' other than its plain and ordinary meaning." *Thorner v. Sony Computer Ent. Am. LLC*, 669 F.3d 1362, 1365 (Fed. Cir. 2012). And "[t]he standard for disavowal of claim scope is similarly exacting. *Id.* The fact that high bar for lexicography is not met here is confirmed not only by the non-definitional nature of the specification's discussion of a "container," but also by the fact that two different defendants attempt to apply the alleged "lexicography" in completely different ways.

In particular, the Amazon defendants apply the alleged "lexicography" of the specification to provide a ***substantially different*** proposed "definition" of "container." *VirtaMove Corp. vs. Amazon.com, Inc, et al.*, Case No. 7:24-cv-00030-DC-DTG, Dkt. No. 71 at 5-6 (W.D. Tex. Oct. 22, 2024). The ***only*** overlap between Google's and Amazon's proposed constructions is the first sentence "An aggregate of files…" and the sentence "Each container for use on a server is mutually exclusive…." *Id.* These disagreements confirm that the entire specification's explanation of containers need not be part of the construction of "container" Regardless of how the Court construes "container," Plaintiff requests that the Court enter identical constructions in both the Google and Amazon actions. Plaintiff believes that the plain and ordinary meaning of "container" applies and is generally consistent with the only two sentences that both Google and Amazon have ***both*** proposed as being definitional.

Google also argues that, without construction, "container" could include "operating systems, kernels, or, by its terms, any files that are collectively needed to run any set of applications on a computer. Dkt. 63 at 10. That is textually false. Claim 1 expressly recites "the containers of application software ***excluding a kernel***," which specifically prevents identifying either a kernel or an operating system (which, by definition, includes a kernel) as the claimed "container." '814 Patent cl. 1.

F. "at least some of the different operating systems/at least some of the plurality of different operating systems" ('814 claim 1) "memory accessible to at least some of the servers" ('814 claim 1)

Term	Plaintiff's Proposed Construction	Defendant's Proposed Construction
At least some of the different operating systems/At least some of the plurality of different operating systems	No construction necessary; plain and ordinary meaning.	at least two or more of the different operating systems / at least two or more of the plurality of different operating systems
Memory accessible to at least some of the servers	memory that at least some of the servers can read from or write to	memory that at least ***two or more*** of the servers can read from or write to

Google's own dictionary definition of the pronoun "some" confirms its plain and ordinary meaning as "an ***indefinite quantity*** or ***indefinite number*** of people or things." Dkt. 63-7 at 6. "Indefinite quantity" does not mean "a quantity of two or more." Furthermore, the same dictionary entry, under the adjectival sense of the word, confirms that "some" modifies "a ***person or persons*** not specified" or "***one or several*** of a number of unspecified alternatives," expressly confirming the basic understanding that "some" means "one or more," not "two or more." *Id.*

Rejecting the plain and ordinary meaning of "some," Google seeks to limit the claim scope to "two or more." This narrowing is unsupported. Google seizes on a statement in the specification that the invention beneficially allows portability between platforms. Dkt. 63 at 11 (quoting '814

Patent at 1:65-2:3). But it is black-letter law that "not every benefit flowing from an invention is a claim limitation." *i4i Ltd. v. Microsoft Corp.*, 598 F.3d 831, 843 (Fed. Cir. 2010). And specifically, statements in the specification "touting the benefits of the invention" cannot limit the claim scope unless they "provide a definition or constitute a clear and unmistakable disclaimer." *Provisur Techs., Inc. v. Weber, Inc.*, No. 2021-1851, 2022 WL 17688071, at *3 (Fed. Cir. Dec. 15, 2022). Google does not contend, and cannot show, that the specification excerpt provides a definition of "some" or constitutes clear and unmistakable disclaimer.

G. "local kernel residing permanently on one of the servers" ('814 cl. 1)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
No construction necessary; plain and ordinary meaning.	local kernel in one of the server's memory that is not lost when power is removed from it

Again, here Google seeks to add extraneous words and concepts that are not present in the claim language or specification. Google's repeated reference to "the intrinsic evidence" merely highlights that Google does not and cannot identify any actual intrinsic support for its construction. *See* Dkt. 63 at 12-13 (no citations to the patent, file history, etc.). Google does not even use a dictionary definition of "permanent," instead making up an attorney-drafted pseudo-definition without either intrinsic or extrinsic support ("persistent or nonvolatile memory", *id.* at 13) and then looking to unrelated dictionary definitions to shore up its creativity. The concept of removing power from a server's memory appears nowhere in the patent claims or specification; nor do the terms or concepts "volatile" and "nonvolatile." The claims do not recite any "server's memory"; is Google referring to the claimed "memory accessible to at least some of the servers"? If not, Google is apparently inserting a new structural limitation. "Permanent" is a plain and ordinary word used in its plain and ordinary sense, and it does not need redefinition.

Moreover, Google's definition potentially requires some prediction about what will happen when "power is removed" from a system, with no guidance as to ***how*** the power is removed or under what circumstances. As Google's own evidence shows, a POSITA would generally know the difference between "permanent" and "temporary" storage, such that the plain meaning of "permanently residing" should be applied.

H. "secure containers of application software" ('814 claim 1)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
Containers where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets	***environments*** where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets

The phrase "secure containers of application software" provides the antecedent basis for all appearances of "container" throughout the claims. Google demands to replace the word "container" with "environment." But "container" is also itself a claim term that Google is asking the Court to construe. If a "secure container" is not a "container," then the Court should not construe "container" at all; and if the Court construes "container," it should not remove that word from the claim.

Google's appeal to lexicography does not require the Court to introduce a new textual inconsistency into the claim. The parties agree on the substance of the lexicography, *i.e.*, the patent's description about control of resources and insulation from the effects of other containers. And in context, a "secure application container" is certainly a type of "container." The patentee defined a "secure application container" as a particular type of environment, *i.e.*, an environment where application sets have certain relationships. This simply confirms that a secure application container is a ***type*** of environment, consistent with the rest of the specification and claim language; it does not mean that a secure application container is not a container. Consider a counterfactual definition of "secure operating system" as "software where security breaches are prevented." Clearly "software" is broader than "operating system," but that does not mean that a "secure operating system" can be satisfied by software that is not an operating system.

I. "an operating system's root file system" ('814 claim 1)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
No construction necessary; plain and ordinary meaning.	Indefinite

The claim recites "In a system having a plurality of servers with operating systems that differ… each of the containers ha[ving] a unique root file system that is different than an operating system's root file system." As VirtaMove explained during the meet and confer with Google, this term only has one possible meaning—it means that the root file system of each container must be different than ***each*** operating system's root file system.

This is the only plausible interpretation of the claim language because a POSITA would readily understand that each operating system has a ***different*** root file system. For example, five different operating systems might have root file systems "A," "B," C," "D," and "E," respectively. If we ask whether a given container has a root file system different from ***any*** of those root file systems A-E, the answer will ***always*** be "yes." For example, if the container had root file system "A," it would be different from root file systems B-E. Likewise, if the container had root file system "E," it would be different from root file systems A-D. And if the container had root file system "F," it would be different from root file systems A-E.

A POSITA would readily understand, in context, that if a container's unique root file system is the same as an operating system's root file system" (i.e., it is the same as ***any*** operating system's root file system), that container's root file system is ***not*** "different from an operating system's root file system." This is the only way to give meaning to this limitation. For example, Google does not allege that the claims identify a ***single*** operating system's root file system that must be examined. And Google cannot dispute that if a container's root file system had to be different from "at least one" of an operating system's root file system, then that limitation would ***always*** be satisfied. Accordingly, this claim term is not indefinite. If the Court believes a construction is necessary, it should be construed to mean "each of the containers has a unique root file system that is not the same as any operating system's root file system."

II. U.S. Patent No. 7,784,058

A. "critical system elements" (claim 1)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.	Indefinite

Contrasting with Google's other "lexicography" proposals, the '058 Patent does provide an unambiguous definition of the phrase "critical system element[s]," stating what a CSE is rather than providing examples or embodiments. There are two elements: that the CSE is "'normally' supplied by an operating system" and that it is "critical to the operation of a software application."

Regarding "normal," the patent specification provides further context, explaining: "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." '058 Patent at 1:22-27. This illustrates the conventional arrangement wherein CSEs are "normally" provided by an operating system (i.e., they are provided by the operating system if the structure of the operating system is not modified beyond its default operation). The specification also provides contrasting examples of the "invention," consistent with the claims, where "some system elements that are critical to the operation of a software application *are replicated from kernel mode, into user mode*…. These system elements are contained in a shared library." *Id.* at 9:15-19 (emphasis added). The specification parallels the claim requirements and confirms that the OSCSEs recited in limitation 1(b) generally correspond to the operation of a conventional system (where the operating system provides the critical system elements), whereas the SLCSEs of limitation 1(c) generally correspond to a non-conventional aspect of the claimed invention (where the critical system elements are stored in a shared library, outside of the operating system).

Second, Google's attack on the word "critical" fails. Google provides no evidence on this point. The Court should look to the text of the patent specification itself, the intrinsic evidence that is the best guide to the patent's meaning. *See, e.g.*, *Phillips v. AWH Corp.*, 415 F.3d 1303, 1319 (Fed. Cir. 2005) (extrinsic evidence, such as expert reports, "is unlikely to result in a reliable interpretation of patent claim scope unless considered in the context of the intrinsic evidence."); *OSRAM GmbH v. Int'l Trade Comm'n*, 505 F.3d 1351, 1356 (Fed.Cir.2007) ("The patent specification is the primary resource for determining how an invention would be understood by persons experienced in the field.").

The '058 Patent provides numerous examples of critical system elements, more than sufficient to illustrate what elements are "critical." First, the specification discusses "a TCP/IP stack," which a POSITA would readily recognize as the core network protocols used for Internet communication. '058 Patent at 5:41-53. The TCP/IP stack is plainly critical to any application that uses Internet communication. The next examples are additional network services, "including TCP/IP, Bluetooth, ATM; or message passing protocols." *Id.* at 6:11-13. The specification goes on to provide specific examples of CSEs that represent extensions or optimizations to file system or network functionalities, such as services to "[a]ccess files that reside in different locations" and network optimizations including "[m]odified protocol processing for custom hardware services." *Id.* at 6:14-28. In each case, software designed to rely on these services plainly would not function in its intended manner without them.

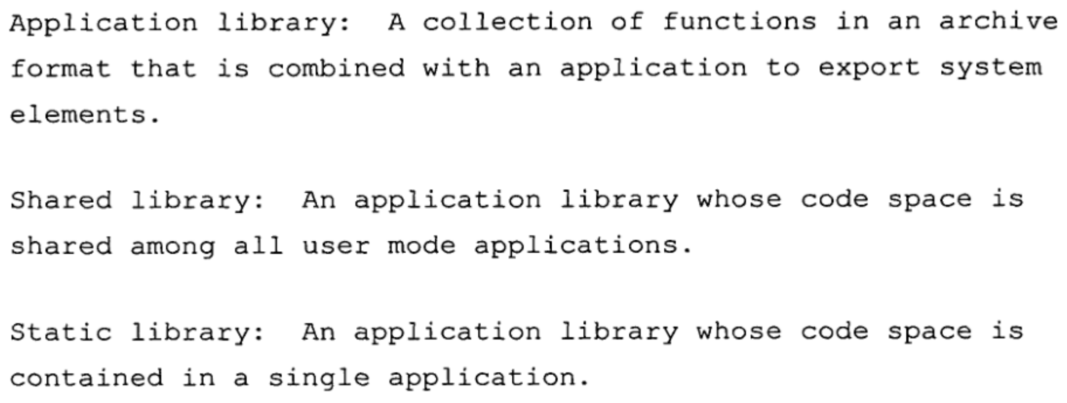
All of this intrinsic evidence guides a POSITA's understanding of what services are "critical" and confirms the definiteness of the claim scope. Google points to no evidence that any *other* understanding of "critical" would even be considered by a POSITA in the context of the '058 Patent and the above-cited intrinsic evidence. At the very least, Google's failure even to mention this evidence confirms that Google cannot prove indefiniteness by clear and convincing evidence, as required.

B. "shared library" (claim 1)

Plaintiff's Proposed Construction	Defendants' Proposed Construction
An application library occupying a code space shared among all user mode applications, which is different than the code space occupied by the kernel and its associated files and is accessible to multiple applications. In the alternative: An application library ***whose*** code space ***is*** shared among all user mode applications.	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.

The term "shared library" appears throughout the specification and claims of the '058 Patent. It has a plain and ordinary meaning that is confirmed by the claim context and by the specification. For example, the patent specification makes clear that "code space" refers to where a library is located, not to the library itself. *See, e.g.*, '058 Patent at 3:39-45 ("the same set of instructions in the same physical memory space, ***that is, shared code space***…"); *id.* at 6:54-55 ("Static library: An application ***whose*** code space is ***contained*** in a single application"); *id.* at 7:3-5 ("[W]hat is commonly done is to provide an application library ***in*** shared code space, which multiple applications can access."). This usage, which reflects the plain and ordinary meaning of "code space" to a POSITA, contradicts the notion that a shared library is *defined* as "an application library code space" as Google requests.

There is a simple explanation for the confusing construction, though: the patent applicant obviously introduced a pair of typographical errors into the definition of "Shared library." The original version of this definition, in the provisional application to which the '058 Patent claims priority, is shorter: "An application library ***whose*** code space ***is*** shared among all user mode applications." Ex. 1 (Provisional Patent Application No. 60/504,213) at 9. That definition cleanly flowed from the definition of "Application library" above it, and paralleled the definition of "Static library" below it, confirming that the key difference between a shared library and a static library is whether the code space is contained in a single application or shared among applications:



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

Shared library: An application library whose code space is shared among all user mode applications.

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

Id.

When the applicant revised the provisional specification to form the non-provisional application, additional detail was added to the definition, but the words "whose" and "is" were removed. Those words were not deleted from the definition of "Static library," which retains the same definition in the final specification. The new language includes "The code space is different than that occupied by the kernel," confirming that "code space" is a space occupied by code, not code itself. This confirms that the deletion of "whose" was unintentional, and that the correct interpretation should retain the original language of the provisional. A POSITA reading the specification would readily understand that this is the correct interpretation.

In the co-pending action against the Amazon defendants, VirtaMove has proposed "An application library ***whose*** code space ***is*** shared among all user mode applications" for this term. For consistency across the two actions, VirtaMove proposes the same construction here. This construction more accurately reflects the intended lexicography, as described above. Therefore, if the extent the Court believes construction is necessary, the correct definition without the typographical errors should be included: "An application library ***whose*** code space ***is*** shared among all user mode applications."

The other sentences requested by Google are both duplicative of existing claim limitations. Claim 1 already requires the shared library to be "in user mode," as distinguished from "kernel mode." And both the claim context and the first sentence of the lexicography already require the shared library to be accessible by multiple applications. Although these sentences are redundant, VirtaMove has proposed a compromise proposal that adapts them to the claim scope.

C. "some of the SLCSEs stored in the shared library….are accessible to some of the plurality of software applications / accessed by one or more of the plurality of software applications it" ('058 cl. 1)

Plaintiff's Proposed Construction	Defendants' Proposed Construction
Plain and ordinary meaning. In the alternative: wherein ***some*** of the plurality of the software applications can ***use*** SLCSEs stored in the shared library/***used*** by one or more of the plurality of software applications	wherein ***two or more*** of the plurality of the software applications can ***read*** SLCSEs stored in the shared library/***read*** by one or more of the plurality of software applications

As to "some" vs. "two or more," Google is wrong for the same reasons discussed above in section I.F above. Also, here Google does not even attempt to show intrinsic support for its "two or more" construction.

As to "read/read," Google again entirely ignores the intrinsic record, providing no citations to the '058 Patent and instead substituting attorney characterization and commercial dictionary definitions. Equating "access" with merely reading memory would render numerous portions of the '058 Patent specification, and even the claims, nonsensical. For example, claims 4 and 9 both recite different ways to "access" services provided by the operating system, either "using system calls" or using "a function overlay." The specification provides an exemplary embodiment of accessing services using system calls at 8:46-53. There is no suggestion that accessing services means reading the services from memory. Likewise, the specification provides an exemplary embodiment of accessing services using a function overlay at 8:62-9:13. This embodiment also uses function calls and operating system functionalities for substituting libraries, not just memory reads, for access. In either case, a narrow interpretation of "access" to exclusively mean reading memory is contrary to both plain meaning and the specification.

Furthermore, the description of a preferred embodiment expressly describes that, in the case where a SLCSEs is a "replica" or "substantial functional equivalent" of a kernel function, it "can be directly ***called*** by the applications 42 and as such can be ***run*** in the same context as the applications 42." *Id.* at 8:28-36. This directly corresponds to the disputed claim limitation 1(c)(i), which claims "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." This passage confirms that "accessing" can be performed not only by reading, but also by calling or by running.

VirtaMove's proposed construction "use" more accurately captures the plain and ordinary meaning of "access" as used in the specification. VirtaMove also believes that the original claim language, "are accessible to" / "accessed by," is readily understandable both to a POSITA and to a jury, and construing the term as "plain and ordinary meaning" would also resolve the parties' dispute by rejecting Google's request to narrow the scope of the claim to "read."

D. "functional replicas of OSCSEs" (claim 1)

Plaintiff's Proposed Construction	Defendant's Proposed Construction
Substantial functional equivalents or replacements of kernel functions	Indefinite

Google argues that "replica" has a lexicographic definition, *i.e.* "a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system (OS)," and that definition is indefinite as a term of degree because of the words "substantial" and "similar." Dkt. 63 at 18-19. This argument fails at both steps.

The Federal Circuit has explained that "[b]ecause language is limited, we have rejected the proposition that claims involving terms of degree are inherently indefinite." *Sonix Tech. Co. v. Publications Int'l, Ltd.*, 844 F.3d 1370, 1377 (Fed. Cir. 2017). "Thus, a patentee need not define his invention with mathematical precision in order to comply with the definiteness requirement." *Id.* (internal quotation marks omitted). "Claim language employing terms of degree has long been found definite where it provided enough certainty to one of skill in the art when read in the context of the invention." *Interval Licensing LLC v. AOL, Inc.*, 766 F.3d 1364, 1370 (Fed. Cir. 2014). In determining whether the patent has provided sufficient guidance for a term of degree, a reviewing court should "look to the written description for guidance." *Id.* at 1371.

First, Google's focus on a single sentence from the patent specification ignores the claim context and the full disclosure of the patent specification. In particular, the claim term is "functional replica," not "replica." Even if the generic description of "replica" were indefinite (it is not), the limitation to functional replicas provides important clarification.

The specification contains an additional description of the scope of "the term replica" specifically in the context of *functional* replicas: "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." '058 Patent at 8:27-32; *see also id.* at 9:52-56 ("The term replication means that like services are supplied [*i.e.*, that] essentially a same functionality is provided."). These sentences explicitly state what scope is "encompass[ed]" by "the term replica": (1) substantial functional equivalents of kernel functions; (2) replacements of kernel functions; and (3) copies of OSCSEs (*i.e.*, kernel functions). Of these three categories, "substantial functional equivalents" is logically the broadest, since either a replacement or a copy of a kernel function/OSCSE would necessarily also be functionally equivalent.

Accordingly, the phrase "functional replica" does not require mere similarity, but rather (at a minimum) "substantial functional equivalen[ce]." '058 Patent at 8:27-32. Google's suggestion that determining substantial functional equivalence of two CSEs is indefinite fails. Juries are regularly required to determine functional equivalence in the context of the Doctrine of Equivalents or in the context of 35 U.S.C. § 112, ¶ 6. As to "substantial," Google provides no evidence or explanation whatsoever why "substantial" is indefinite here, other than generic and inapposite case citations. It is Google's burden to prove indefiniteness by clear and convincing evidence, and Google has completely failed to meet that burden as to "substantial."

Dated: November 12, 2024

Respectfully submitted,

/s/ Reza Mirzaie
Reza Mirzaie
CA State Bar No. 246953
Marc A. Fenster

CA State Bar No. 181067
Neil A. Rubin
CA State Bar No. 250761
Amy E. Hayden
CA State Bar No. 287026
Jacob R. Buczko
CA State Bar No. 269408
James S. Tsuei
CA State Bar No. 285530
James A. Milkey
CA State Bar No. 281283
Christian W. Conkle
CA State Bar No. 306374
Jonathan Ma
CA State Bar No. 312773
Daniel Kolko
CA State Bar No. 341680
RUSS AUGUST & KABAT
12424 Wilshire Boulevard, 12th Floor
Los Angeles, CA 90025
Telephone: 310-826-7474
Email: rmirzaie@raklaw.com
Email: mfenster@raklaw.com
Email: nrubin@raklaw.com
Email: ahayden@raklaw.com
Email: jbuczko@raklaw.com
Email: jtsuei@raklaw.com
Email: jmilkey@raklaw.com
Email: cconkle@raklaw.com
Email: jma@raklaw.com
Email: dkolko@raklaw.com

Qi (Peter) Tong
4925 Greenville Ave., Suite 200
Dallas, TX 75206
Email: ptong@raklaw.com

ATTORNEYS FOR PLAINTIFF VIRTAMOVE, CORP.

CERTIFICATE OF SERVICE

I certify that this document is being served upon counsel of record for Defendants on November 12, 2024 via electronic service.

/s/ *Christian W. Conkle*

Exhibit 1

13408 U.S. PTO 09/22/03

Please type a plus sign (+) inside this box ⟶ +

PTO/SB/16 (8-00)
Approved for use through10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT und r 37 CFR 1.53(c).

PTO 60/504213 09/22/03

INVENTOR(S)

Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
Donn	Rochette	Fenton, Iowa, USA
Dean	Huffman	Kanata, Ontario, Canada
Paul	O'Leary	Kanata, Ontario, Canada

☐ *Additional inventors are being named on the____ separately numbered sheets attached hereto*

TITLE OF THE INVENTION (280 characters max)

USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

Direct all correspondence to: **CORRESPONDENCE ADDRESS**

☒ Customer Number 000293 ⟶ *Place Customer Number Bar Code Label here*

OR *Type Customer Number here*

☐ Firm or Individual Name					
Address					
Address					
City		State		ZIP	
Country		Telephone		Fax	

ENCLOSED APPLICATION PARTS ***(check all that apply)***

☒ Specification *Number of Pages* 23 ☐ CD(s), Number

☒ Drawing(s) *Number of Sheets* 6 ☐ Other (specify)

☐ Application Data Sheet. See 37 CFR 1.76

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT *(check one)*

☒ Applicant claims small entity status. See 37 CFR 1.27.

☒ A check or money order is enclosed to cover the filing fees

☒ The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number 04-1577

☐ Payment by credit card. Form PTO-2038 is attached.

FILING FEE AMOUNT ($) $80.00

The PTO did not receive the following listed items(s) Small Entity Status

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.

☐ Yes, the name of the U.S. Government agency and the Government contract number are: __________

Respectfully submitted,

SIGNATURE [illegible]

Date 09/17/03

TYPED or PRINTED NAME **Ralph A. Dowell**

REGISTRATION NO. *(if appropriate)* 26,868

Docket Number: 14451PRO

TELEPHONE (703) 415-2555

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

P19SMALL/REV05

USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

Field of the Invention

The invention relates to computer software, and more specifically to software that affects and extends services exported through application libraries.

Background of the Invention

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of the operating system to provide mechanisms to safely and effectively control access to shared resources.

In some cases the centralized control of elements critical to software applications creates a limitation caused by conflicts for shared resources. Two software applications that require the same file, yet each require a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. The common solution to these situations is to place software applications that may potentially conflict on separate compute platforms.

Current state of the art defines two architectural approaches to the migration of system elements from an operating system into an application context. 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. 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.

Summary of the Invention

In accordance with a first broad aspect, the invention provides a computing architecture that has an operating system kernel having critical system elements 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.

In some embodiments, the computing architecture has application libraries accessible by the software applications and augmented by the shared library.

In some embodiments, the critical system elements are left in the operating system kernel.

In some embodiments, the critical system elements use system calls to access services in the operating system kernel.

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.

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

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.

In some embodiments, the operating system kernel is adapted to provide interrupt handling capabilities to user mode CSEs.

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

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

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

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

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

In some embodiments, the operating system kernel is enabled when the software application is loaded into memory.

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

In some embodiments, in a native form the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements.

In some embodiments, the kernel services comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated.

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 architecture 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.

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.

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

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

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

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.

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.

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.

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

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.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in a 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.

According to a fifth broad aspect, the invention provides an operating system kernel having systems 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 of the system elements which are critical to the software applications.

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. 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.

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

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

Figure 1 is an architectural view of the traditional monolithic operating system;

Figure 2 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;

Figure 3 is an architectural view of an embodiment of the invention;

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

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

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

Detailed Description of the Preferred Embodiments

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. Replication implies that system elements are not replaced from an operating system, rather they become separate extensions accessed through shared libraries.

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

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.

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

User mode: The context in which applications execute.

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.

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.

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

Shared library: An application library whose code space is shared among all user mode applications.

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

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.

Up call mechanism: A means by which a service in kernel mode calls a function in a user mode application context. It is common to provide an up call mechanism within an operating system kernel.

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.

In order for an application of Figure 1 to make use of a critical system element in the kernel, the application makes a call to the application libraries. 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 makes a call to a critical system element through the application library, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware enters kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval 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.

Figure 2 shows a system architecture where critical system elements 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 as discussed in United States Provisional Patent Application entitled "Drag & Drop Application Management" which is incorporated herein by reference. 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 needs to make use of a critical system element which is being run in user mode, a rather convoluted 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 2 as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

The invention is contrasted with both of these architectures in that critical system elements are not isolated in the operating system kernel in the case of a monolithic architecture (Figure 1), also they are not removed from the context of an application as is the case with a multi-server architecture (Figure 2). Rather they are replicated and embodied in the context of an application.

Figure 3 shows an architectural view of the overall operation of the invention. Multiple user processes execute above a single instance of an operating system. Software applications utilize shared libraries as is done in United States Provisional Patent Application 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. Extended services are similar to those that appear in the context of the operating system kernel.

Figure 4 shows functionality above the operating system kernel all of which is run in user mode while the operating system kernel itself runs in kernel mode. The user mode functionality includes the user applications, the standard application libraries, and a new extended shared library provided by an embodiment of the invention. The extended shared library includes replicas of kernel functions. These functions can be directly called by the applications and as such can be run in the same context as the applications. In preferred embodiments, the kernel functions which are included in the extended shared library are also included in the operating system kernel. Furthermore, there might be different versions of a given critical system element forming part of the extended shared library with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel, for example certain system calls. Then the critical system elements which are included in the extended shared library may still make use of the operating system kernel to implement these platform specific functions.

Figure 5 represents the function of the kernel module (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.

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

Function Overlays

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 to intercept software application accesses to operating system services. 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 an 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 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.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, by the operating system, 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. 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.

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.

As discussed previously, different applications may be provided with their own versions of the critical system elements. Advantageously, this can make it appear that multiple applications running on a given platform have their own operating system. However in reality, there is only one operating system with application specific components in user mode, and with non-application specific components only residing in the kernel mode. It is noted that this allows different applications running on a given platform to operate in secure separation without clashing for resources and without versioning problems of libraries. Previous attempts to address this problem have included building hardware that is capable of running multiple versions of an operating system, and building a virtual machine in software which effectively allows each service to have its own operating system, but in software. The new solution provided by this embodiment of the invention yields the same benefits of these two other solutions, but without the requirement from multiple operating systems.

Kernel module

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:

- Initialization. Establish a channel between a CSE in user mode and a specific device. Informs the interrupt service that this CSE requires notification.

- Write data. Transfer data from a CSE to a device. User mode virtual addresses are converted to kernel mode virtual addresses.

- Read data. Transfer data from a device to a CSE. Kernel mode data is mapped into virtual addresses in user mode.

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.

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.

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. 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.

WE CLAIM:

1. A computing architecture comprising:

an operating system kernel having critical system elements 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 software applications in user mode;

wherein the critical system elements are run in a context of a software application.

2. A computing architecture according to claim 1 comprising application libraries accessible by the software applications and augmented by the shared library.

3. A computing architecture according to claim 1 wherein the critical system elements are left in the operating system kernel.

4. A computing architecture according to claim 1 wherein the critical system elements use system calls to access services in the operating system kernel.

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

6. A computing architecture according to claim 5 wherein the critical system elements in the context of an application program use system calls to access services in the kernel module.

7. A computing architecture according to claim 5 or 6 wherein the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

8. A computing architecture according to any one of claims 5 to 7 wherein the operating system kernel is adapted to provide interrupt handling capabilities to user mode CSEs.

9. A computing architecture according to claim 8 wherein the interrupt handling capabilities are initialized through a system call.

10. A computing architecture according to claim 9 wherein the kernel module comprises a handler which is installed for a specific device interrupt.

11. A computing architecture according to claim 10 wherein the handler is called when an interrupt request is generated by a hardware device.

12. A computing architecture according to claim 11 wherein the handler notifies the service in the context of an application through the use of an up call mechanism.

13. A computing architecture according to any one of claims 1 to 12 wherein function overlays are used to intercept software application accesses to operating system services.

14. A computing architecture according to any one of claims 1 to 13 wherein the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

15. A computing architecture according to any one of claims 1 to 14 wherein in a native form the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

16. A computing architecture according to claim 15 wherein the kernel services are replicated in user mode and contained in the shared library with the critical system elements.

17. A computing architecture according to claim 15 or 16 wherein the kernel services comprise memory allocation, synchronization and device access.

18. A computing architecture according to any one of claims 15 to 17 wherein the kernel services that are platform specific are not replicated.

19. A computing architecture according to claim 18 wherein the kernel services which are platform specific are called by a critical system element running in user mode.

20. A computing architecture according to claim 2 wherein a user process running under the computing architecture comprises 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.

21. A computing architecture according to any one of claims 1 to 20 wherein the software applications are provided with respective versions of the critical system elements.

22. A computing architecture according to claim 21 wherein 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.

23. A computing architecture according to any one of claims 1 to 22 wherein a control code is placed in kernel mode.

24. A computing architecture according to claim 5 wherein the kernel module is adapted to enable data exchange between the critical service elements in user mode and a device driver in kernel mode.

25. A computing architecture according to claim 24 wherein the data exchange uses mapping of virtual memory such that data is transferred both from the critical service elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical service elements in user mode.

26. A computing architecture according to claim 24 or 25 wherein the kernel module is adapted to export services for device interface.

27. A computing architecture according to claim 26 wherein 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.

28. A computing architecture according to claim 26 wherein 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.

29. A computing architecture according to claim 26 wherein the export services comprise transfer of data from a

device to a critical system element of the critical system elements in user mode.

30. A operating system comprising the computing architecture of any one of claims 1 to 29.

31. A computing platform comprising the operating system of claim 30 and computing hardware capable of running under the operating system.

32. A computing architecture according to any one of Figure 1 to 3.

33. A shared library accessible to software applications in a 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.

34. An operating system kernel having systems 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 of the system elements which are critical to the software applications.

35. A computing architecture according to claim 13 wherein the operating system kernel is enabled when the software application is loaded into memory.

36. 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 comprising:

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;

wherein the critical system elements are run in a context of a software application.

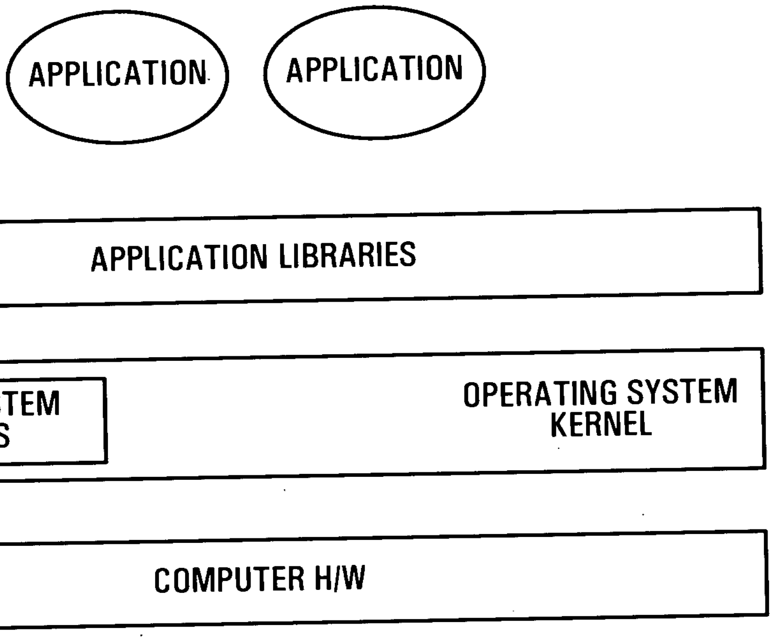

APPLICATION
APPLICATION
APPLICATION LIBRARIES
CRITICAL SYSTEM ELEMENTS
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 1

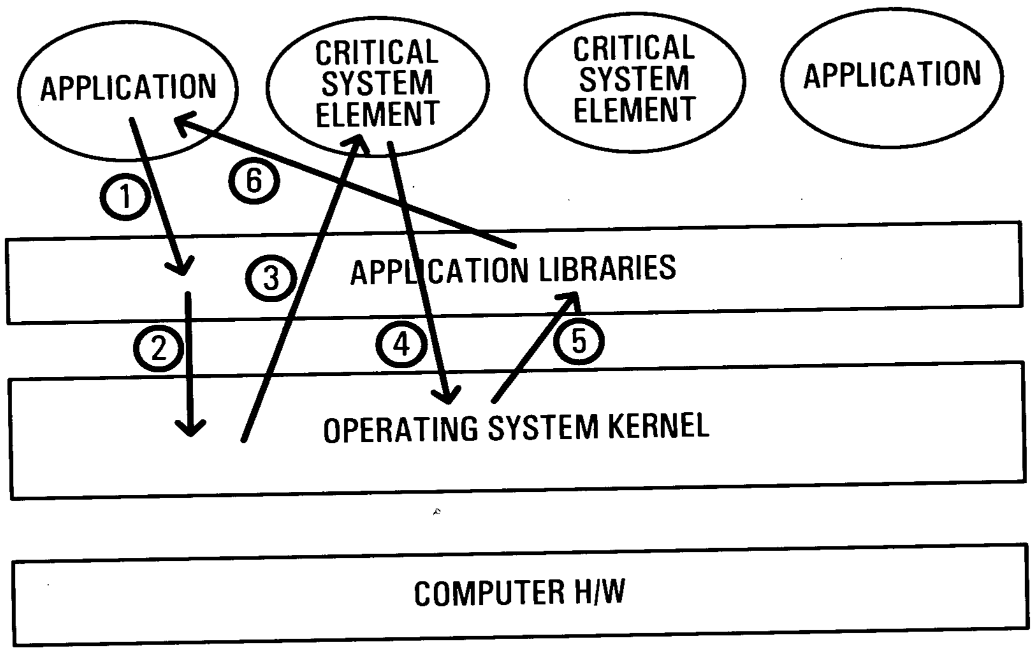

APPLICATION
CRITICAL SYSTEM ELEMENT
CRITICAL SYSTEM ELEMENT
APPLICATION
1
6
APPLICATION LIBRARIES
3
2
4
5
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 2

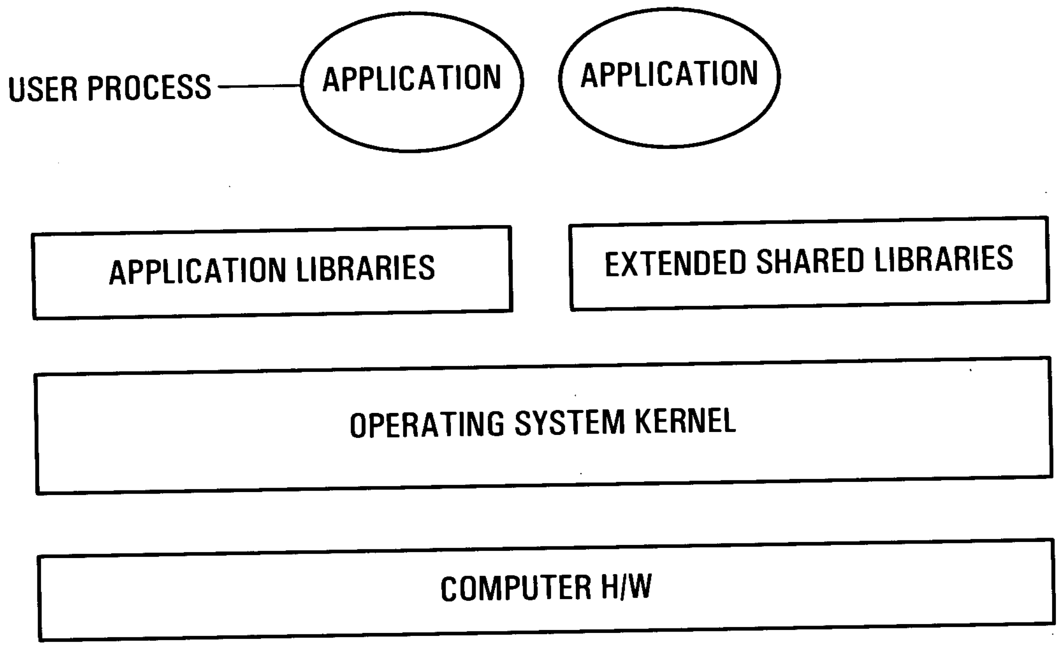

USER PROCESS
APPLICATION
APPLICATION
APPLICATION LIBRARIES
EXTENDED SHARED LIBRARIES
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 3

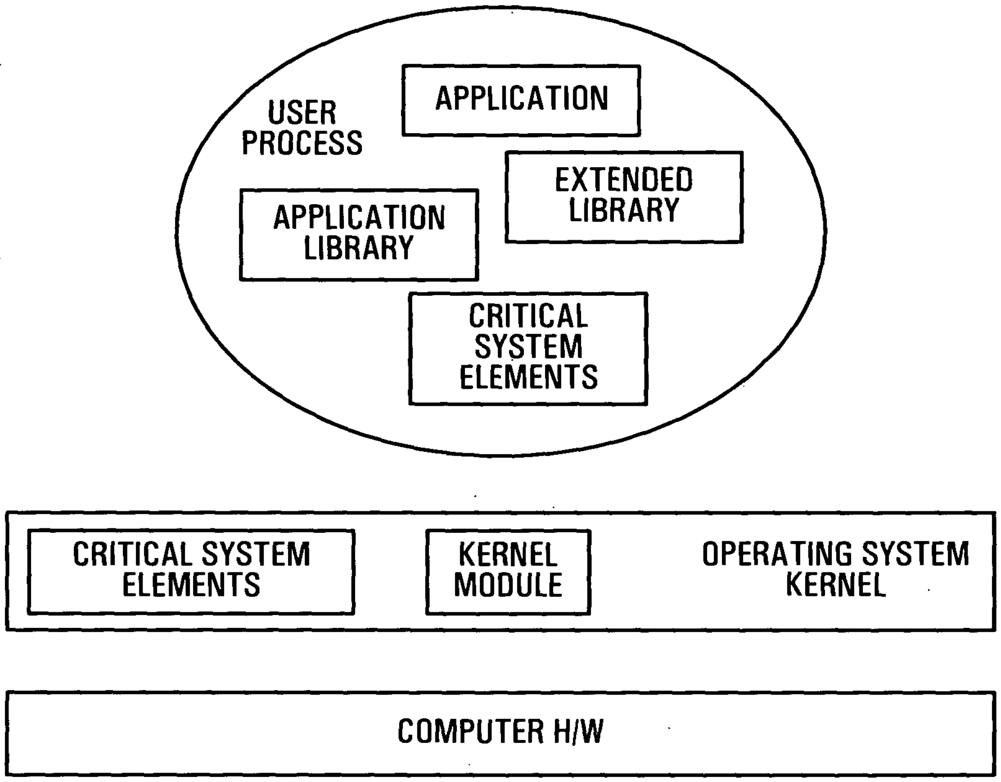

USER PROCESS
APPLICATION
EXTENDED LIBRARY
APPLICATION LIBRARY
CRITICAL SYSTEM ELEMENTS
CRITICAL SYSTEM ELEMENTS
KERNEL MODULE
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 4

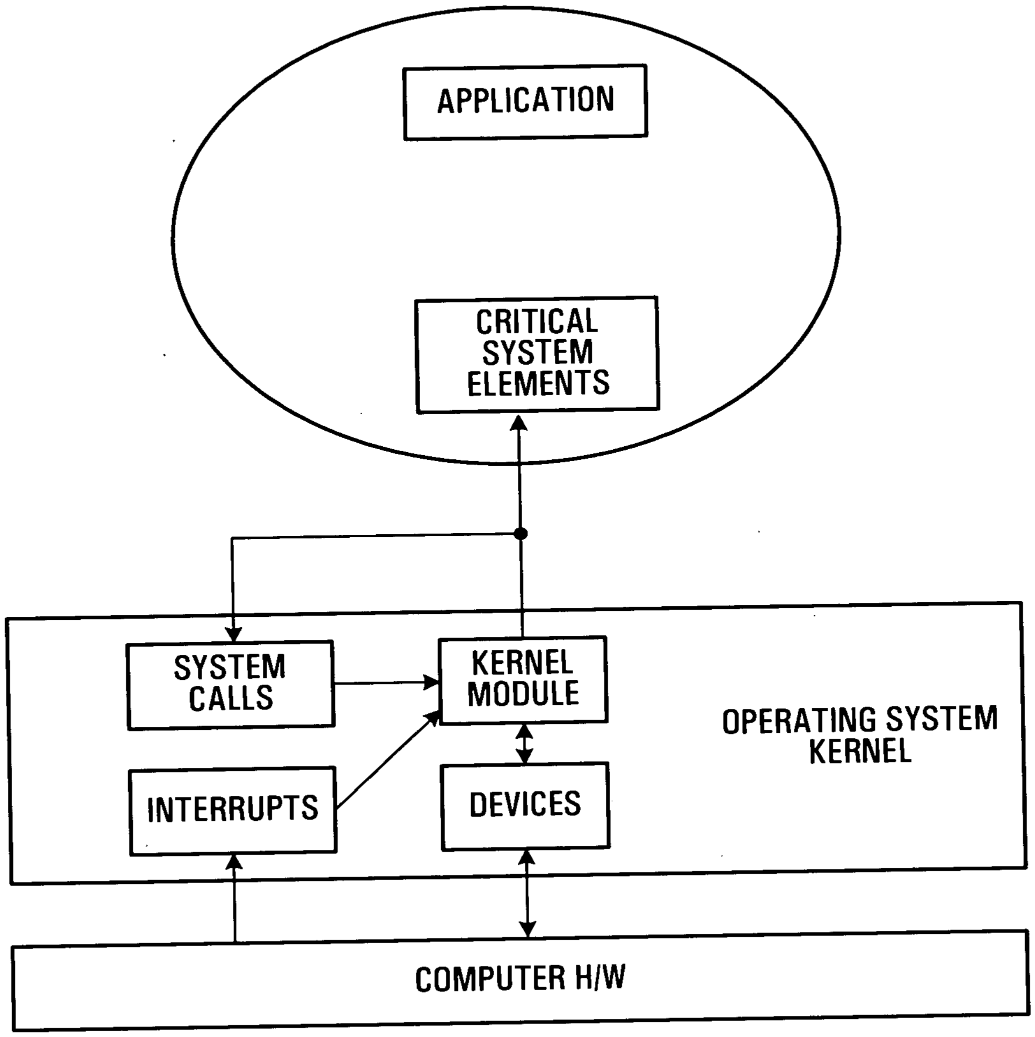

APPLICATION
CRITICAL SYSTEM ELEMENTS
SYSTEM CALLS
KERNEL MODULE
INTERRUPTS
DEVICES
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 5

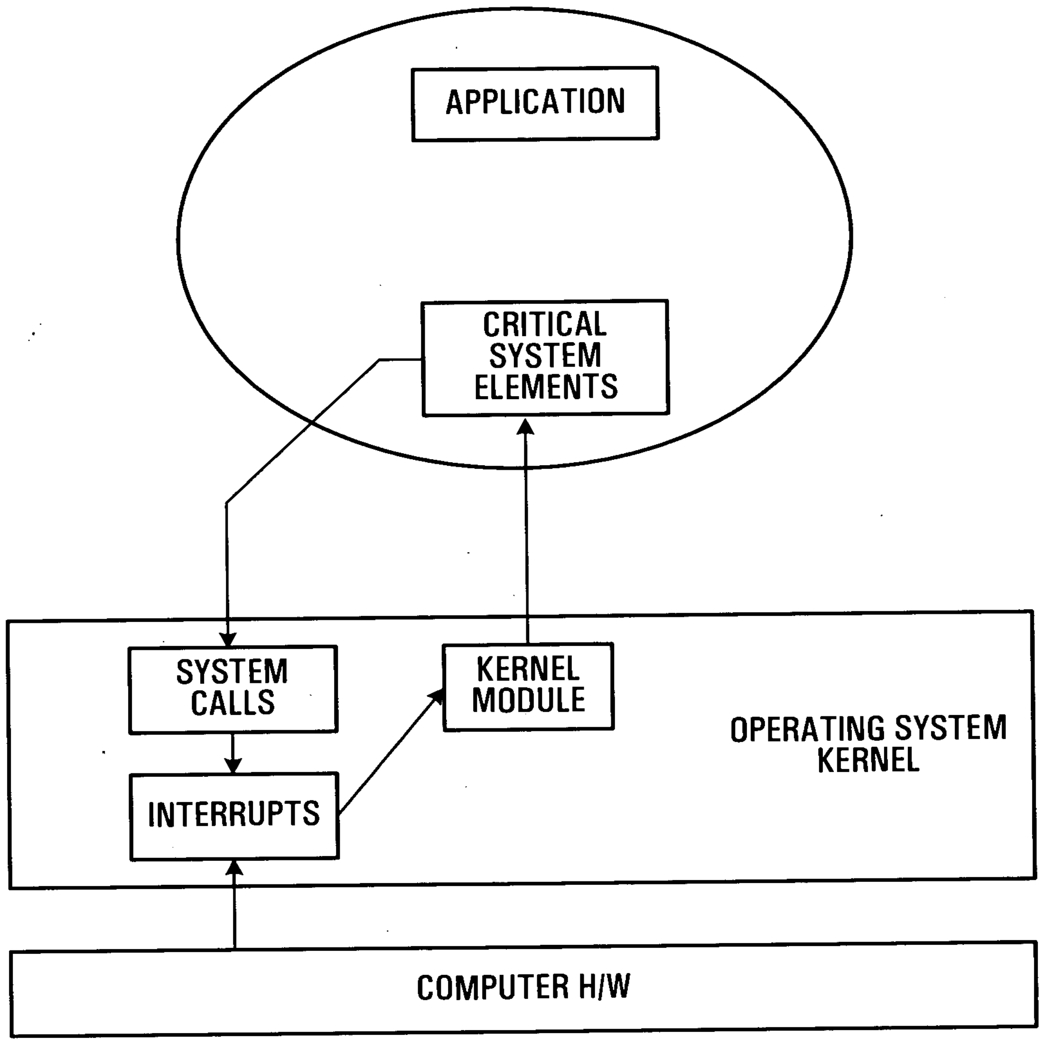

APPLICATION
CRITICAL SYSTEM ELEMENTS
SYSTEM CALLS
KERNEL MODULE
INTERRUPTS
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 6

PATENT APPLICATION SERIAL NO. ____________________

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

```
09/25/2003 EFLORES  00000027 60504213
01 FC:2005                        80.00 OP
```

PTO-1556
(5/87)