

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

GOOGLE LLC,
Petitioner,

v.

VIRTAMOVE, CORP.,
Patent Owner.

Case No. IPR2025-00489
Patent No. 7,784,058

DECLARATION OF SAMRAT BHATTACHARJEE, PH.D.

TABLE OF CONTENTS

I.	PERSONAL AND PROFESSIONAL BACKGROUND	1
II.	MATERIALS REVIEWED AND CONSIDERED	5
III.	MY UNDERSTANDING OF PATENT LAW	8
	A. Anticipation	10
	B. Obviousness.....	11
IV.	UNPATENTABILITY GROUNDS.....	13
V.	THE '058 PATENT.....	14
	A. Background and Specification.....	14
	B. Person of Ordinary Skill in the Art (“POSA”).....	25
	C. Prosecution History	26
	D. Challenged Claims.....	27
VI.	Claim Construction.....	28
VII.	Callender Renders Claims 1-18 Obvious	28
	A. Callender.....	28
	B. Claim-by-Claim Analysis.....	42
	1. Claim 1	42
	a. [1PRE]: “A computing system for executing a plurality of software applications comprising:”.....	42
	b. [1A] “a) a processor;”	46
	c. [1B]	47
	i. [1B.1] “b) an operating system having an operating system kernel...”.....	47
	ii. [1B.2] “[OS kernel] having OS critical system elements (OSCSEs)...”.....	49
	iii. [1B.3] “[OSCSEs] for running in kernel mode using said processor”.....	58
	d. [1C]	61
	i. [1C.1] “c) a shared library having shared library critical system elements (SLCSEs) stored therein...”.....	61

ii.	[1C.2] “for use by the plurality of software applications in user mode...”	71
e.	[1D]	79
i.	[1D.1] “i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and...”	79
ii.	[1D.2] “are accessible to some of the plurality of software applications and...”	84
iii.	[1D.3] “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”	88
f.	[1E]	90
i.	[1E.1] “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...”	90
ii.	[1E.2] “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, and...”	95
g.	[1F] “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.”	97
2.	Claim 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”	100

3. Claim 3: “A computing system as defined in claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel”	103
4. Claim 4	104
a. [4A] “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...”	104
b. [4B] “use system calls to access services in the operating system kernel”	106
i. Queue Setup	107
ii. Interrupt Setup and Event Notification.....	113
5. Claim 5	115
a. [5A]: “A computing system according to claim 1 wherein the operating system kernel comprises a kernel module...”	115
b. [5B]: “adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.”	117
6. Claim 6: “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.”.....	121
7. Claim 7: “A computing system according to claim 6, 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.”	124
8. Claim 8: “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.”.....	126
9. Claim 9: “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.”	127
10. Claim 10: “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.”	128
11. Claim 11: “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.”	130
a. Device access	130
b. Interrupt Delivery	131
c. Virtual Memory Mapping	131
12. Claim 12: “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.”	132
13. Claim 13: “A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.”	133
14. Claim 14: “A computing system according to claim 13 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.”	134
15. Claim 15: “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.”	135
16. Claim 16: “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.”	137
17. Claim 17: “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.”	137

18. Claim 18: “A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.”	138
VIII. APPENDIX: CLAIM LISTING	141

I, Samrat Bhattacharjee, Ph.D., declare:

1. I have been retained by Wolf, Greenfield & Sacks, P.C., counsel for Petitioner Google LLC, to assess claims 1-18 (the “challenged claims”) of U.S. Patent No. 7,784,058 (“the ’058 patent”) (EX1001). I am being compensated for my time at my standard rate of \$700.00 per hour, plus actual expenses. My compensation is not dependent in any way upon the outcome of the *inter partes* review of the ’058 patent.

I. PERSONAL AND PROFESSIONAL BACKGROUND

2. My qualifications for forming the opinions set forth in this declaration are summarized below and explained in more detail in my current curriculum vitae, provided as EX1004. EX1004 also includes a list of my publications and patents.

3. I am a tenured professor in the Computer Science Department at the University of Maryland. I am also an Affiliate Professor with the Department of Electrical and Computer Engineering at the University of Maryland. I have been an appointed faculty member at the University of Maryland since 1999.

4. I received a Ph.D. in computer science from the Georgia Institute of Technology in 1999. Before that I received a B.S. degree in Mathematics and Computer Science with Highest Distinction (Summa Cum Laude) and was recognized as the Outstanding Department Major at Georgia College and State

University, Milledgeville, Georgia in 1994.

5. I have been involved in the research and development of computing systems for over two decades. I perform research and I teach in the areas of computer networks, distributed systems, computer security, cryptography, and operating systems, among others. A major theme of my research has been the development of technology to improve the performance and security of data transfers and data storage on networks like the Internet. My research has examined problems ranging from network and operating system support audio and video streaming, large scale Internet video distribution, the design of novel network architectures for better and secure content distribution and storage, efficiency of content distribution over wireless networks, and so on.

6. For example, starting in the late 1990s, the focus of my research was the development of network and operating system technology to design a new network architecture that allowed for more efficient data transfer. Part of my Ph.D. research was to develop new and more efficient architectures for video delivery on the Internet, and I have published papers on this architecture during my graduate studies. I continued to work on video delivery and security and storage of digital content as a faculty member, and have published various papers in these areas. During 2007, I was a visiting researcher at AT&T Labs, and one of the projects I focused on was a video content delivery platform. This work resulted in both

publications and a granted US patent (US 8,752,100 B2). Among other research, I have authored papers and developed research software for public-key infrastructure KeyChains and a cryptographic library for “chit”-based security.

7. Along with working on research problems related to computer networking, computer security, cryptography, and video distribution, I routinely taught this material in both graduate and undergraduate courses. A number of my papers in these domains are with students at the University of Maryland, many of whom started in these areas after taking my classes.

8. By September 22, 2003, which is the earliest alleged priority date of the '058 patent, I had had extensive experience working with various Operating Systems, including at the kernel level. My experience included kernel modifications to both Linux and various flavors of SunOS (including Solaris). I had developed an operating system (“OS”) for networking applications, and also performed research on various performance improvements for networking applications within the kernel. After graduating, I taught the undergraduate OS course at Maryland, and also started a reading group to introduce students, including undergraduates to OS research and code. Inspired partly by this experience, a graduate student developed a teaching OS from scratch, which I and others than modified to teach undergraduate OS.

9. As a result of this experience, I co-authored a paper on how to teach low-level systems/OS to undergraduates. Evolutions of this teaching OS is still used to teach undergraduate OS at Maryland.

10. For over 25 years, I have developed professional and academic experience in the field of computer software and the management and deployment of server applications and other forms of networking-related applications. For example, I have authored or co-authored over 100 articles in peer-reviewed journals, conference proceedings, and workshops relating to computer networking and computer systems. Over the years I have received several fellowships, prizes, and awards for teaching excellence and technical papers, including an Alfred P. Sloan Jr. Fellowship in 2004 and the National Science Foundation CAREER award in 2001. I have also received an award from the SIGCOMM foundation for a “Test of Time” paper (that is given to an influential paper published ten years ago). My research work has been supported by multiple grants from the US National Science Foundation, and the Department of Defense. I have also started a Joint Ph.D. program with the University of Maryland and the Max Planck Society in Germany, and co-founded the annual Cornell, Maryland, Max Planck Research School that provides research exposure to about 80 students from across the world during a week-long school.

11. I have served on numerous paper and proposal review panels and participated on program committees for ACM/IEEE groups, technical journals, and conferences in Networking, Security and Systems in the aforementioned areas of computer science. I have created research software for all of my projects, much of which is available online for others to build on.

12. I am a named inventor on four U.S. Patents. These patents are generally related to computer software and the management and deployment of server applications.

13. I have served as an expert witness and technical consultant in litigation and Inter Partes Review matters concerning videoconferencing, data conferencing and screen sharing, voice over IP (VoIP) telephony, multimedia networking, distributed systems, operating systems, computer networks, and datacenter networking, among others. I have testified in several trials and depositions as an expert witness.

II. MATERIALS REVIEWED AND CONSIDERED

14. My findings, as explained below, are based on my years of education, research, experience, and background in the field of cryptographic techniques for computer systems, as well as my investigation and study of relevant materials for this declaration. When developing the opinions set forth in this declaration, I assumed the perspective of a person having ordinary skill in the art, as set forth in

Section V.B below. In forming my opinions, I have studied and considered the materials identified in the list below, as well as any other materials cited in this declaration.

Exhibit	Description
1001	U.S. Patent No. 7,784,058
1002	Prosecution History of U.S. Patent No. 7,784,058
1005	U.S. Patent No. 7,024,672 (“Callender”)
1007	U.S. Patent No. 6,263,376 (“Hatch”)
1008	U.S. Patent No. 6,260,075 (“Cabrerero”)
1009	U.S. Patent No. 6,212,574 (“O’Rourke”)
1010	U.S. Patent No. 5,481,706 (“Peek”)
1011	U.S. Patent No. 7,499,966 (“Elnozahy”)
1012	U.S. Patent App. Pub. No. 2004/0216145 (“Wong”)
1013	U.S. Patent App. Pub. No. 2003/0065856 (“Kagan”)
1014	U.S. Patent No. 7,583,681 (“Green”)
1015	Excerpts from Charles Petzold, <i>Programming Windows 95</i> (Microsoft Press 1996) (“Petzold”)
1016	U.S. Patent App. Pub. No. 2002/0066086 (“Linden”)
1017	U.S. Patent No. 6,349,355 (“Draves”)
1018	Excerpts from Silberschatz et. al, <i>Operating System Concepts</i> (Wiley 6 th ed. 2002) (“Silberschatz”)
1019	U.S. Patent No. 7,451,456 (“Andjelic”)
1020	Chart re: ’058 Patent accompanying Plaintiff VirtaMove Corp.’s Supplemental Preliminary Disclosure of Asserted Claims and Infringement Contentions, in <i>VirtaMove, Corp. v. Google LLC</i> , 7:24-cv-00033-DC-DTG (W.D. Tex.) (Sep. 26, 2024)
1021	U.S. Patent App. Pub. No. 2002/0095224 (“Braun”)
1022	U.S. Patent No. 6,173,336 (“Stoeckl”)
1023	U.S. Patent No. 6,016,515 (“Shaw”)
1024	U.S. Patent No. 7,080,172 (“Schmalz”)
1025	Excerpts from <i>Webster’s New World Computer Dictionary</i> (10 th ed. 2003)
1026	Excerpts from <i>Barron’s Dictionary of Computer and Internet Terms</i> (8 th ed. 2003)
1027	U.S. Patent No. 7,324,450 (“Oliver”)
1028	U.S. Patent App. Pub. No. 2003/0103455 (“Pinto”)

1029	U.S. Patent No. 6,526,567 (“Cobbett”)
1030	U.S. Patent App. Pub. No. 2004/0142563 (“Fontarensky”)
1031	U.S. Patent No. 7,284,246 (“Kemp”)
1032	U.S. Patent No. 5,375,241 (“Walsh”)
1033	U.S. Patent No. 6,698,015 (“Moberg”)
1034	Excerpts from Collin, <i>Dictionary of Computing</i> (Collin 4 th Ed. 2002) (“Collin”)
1035	Microsoft Computer Dictionary (Microsoft 5 th ed. 2002) (“Microsoft”)
1036	U.S. Patent No. 6,526,159 (“Nickerson”)
1037	U.S. Patent App. Pub. No. 2003/0154320 (“Calusinski”)
1038	U.S. Patent No. 7,437,483 (“Goossen”)
1039	U.S. Patent No. 7,100,162 (“Green-162”)
1040	U.S. Patent No. 7,453,852 (“Buddhikot”)
1041	U.S. Patent No. 5,584,023 (“Hsu”)
1042	U.S. Patent No. 7,144,031 (“Lin”)
1043	U.S. Patent No. 6,792,492 (“Griffin”)
1044	U.S. Patent App. Pub. No. 2002/0116563 (“Lever”)
1045	U.S. Patent No. 6,594,698 (“Chow”)
1046	U.S. Patent No. 7,216,164 (“Whitmore”)
1047	U.S. Patent No. 6,988,271 (“Hunt”)
1048	Liss et. al, <i>Efficient Exploitation of Kernel Access to Infiniband: a Software DSM Example</i> , 11th Symposium on High Performance Interconnects, 2003 (“Liss”)
1049	Patel et. al., <i>A Model of Completion Queue Mechanisms Using the Virtual Interface API</i> , Proc. IEEE Int’l Conf. on Cluster Computing, 281-282 (IEEE 2002) (“Patel”)
1052	Google LLC’s Proposed Claim Terms for Construction, <i>VirtaMove, Corp. v. Google LLC</i> , 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 1, 2024)
1053	Plaintiff’s Disclosure of Proposed Claim Constructions, <i>VirtaMove, Corp. v. Google LLC</i> , 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 1, 2024)
1054	U.S. Patent No. 7,124,260 (“LaBerge”)
1055	U.S. Patent No. 7,406,677 (“Colling”)
1056	U.S. Patent No. 5,278,969 (“Pashan”)
1057	Excerpts of <i>Webster’s New World Dictionary</i> (4 th Ed. 2003)
1058	Joint Claim Construction Statement, <i>VirtaMove, Corp. v. Google LLC</i> , 7:24-cv-00033-DC-DTG (W.D. Tex.) (Dec. 18, 2024)

1059	Google’s Opening Claim Construction Brief, <i>VirtaMove, Corp. v. Google LLC</i> , 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 22, 2024)
1061	U.S. Patent No. 7,210,124 (“Chan”)
1062	U.S. Patent No. 7,055,152 (“Ganapathy”)
1063	U.S. Patent No. 6,419,502 (“Trammel”)
1064	U.S. Patent App. Pub. No. 2003/0140051 (“Fujiwara”)
1065	U.S. Patent No. 6,442,752 (“Jennings”)
1066	U.S. Patent App. Pub. No. 2003/0225864 (“Gardiner”)
1067	VirtaMove’s Responsive Claim Construction Brief, <i>VirtaMove, Corp. v. Google LLC</i> , 7:24-cv-00033-DC-DTG (W.D. Tex.) (Nov. 12, 2024)
1068	VirtaMove’s Surreply Claim Construction Brief, <i>VirtaMove, Corp. v. Google LLC</i> , 7:24-cv-00033-DC-DTG (W.D. Tex.) (Dec. 13, 2024)
1069	U.S. Patent No. 7,213,247 (“Wilner”)
1070	U.S. Patent App. Pub. No. 2004/0078600 (“Nilsen”)
1071	U.S. Patent App. Pub. No. 2002/0138675 (“Mann”)
1097	U.S. Patent No. 7,424,710 (“Nelson”)

III. MY UNDERSTANDING OF PATENT LAW

15. In developing my opinions, I discussed various relevant legal principles with Petitioner’s attorneys. I understood these principles when they were explained to me and have relied upon such legal principles, as explained to me, in the course of forming the opinions set forth in this declaration. My understanding in this respect is as follows:

16. I understand that “*inter partes* review” (IPR) is a proceeding before the United States Patent & Trademark Office for evaluating the patentability of an issued patent’s claims based on prior-art patents and printed publications.

17. I understand that, in this proceeding, Petitioner has the burden of proving that the challenged claims of the ’058 patent are unpatentable by a

preponderance of the evidence. I understand that “preponderance of the evidence” means that a fact or conclusion is more likely true than not true.

18. I understand that, in IPR proceedings, claim terms in a patent are given their ordinary and customary meaning as understood by a person of ordinary skill in the art (“POSA”) in the context of the entire patent and the prosecution history pertaining to the patent. If the specification or prosecution history provides a special definition for a claim term that differs from the meaning the term would otherwise possess, that special definition controls. If a claim element is expressed as a “means” for performing a specified function, I understand that it covers the corresponding structure described in the specification and equivalents of the described structure. I have applied these standards in preparing the opinions in this declaration.

19. I understand that determining whether a particular patent or printed publication constitutes prior art to a challenged patent claim can require determining the effective filing date (also known as the priority date) to which the challenged claim is entitled. I understand that for a patent claim to be entitled to the benefit of the filing date of an earlier application to which the patent claims priority, the earlier application must have described the claimed invention in sufficient detail to convey with reasonable clarity to the POSA that the inventor had possession of the claimed invention as of the earlier application’s filing date. I

understand that a disclosure that merely renders the claimed invention obvious is not sufficient written description for the claim to be entitled to the benefit of the filing date of the application containing that disclosure.

20. I understand that for an invention claimed in a patent to be patentable, it must be, among other things, new (novel—*i.e.*, not anticipated) and not obvious from the prior art. My understanding of these two legal standards is set forth below.

A. Anticipation

21. I understand that, for a patent claim to be “anticipated” by the prior art (and therefore not novel), each and every limitation of the claim must be found, expressly or inherently, in a single prior-art reference. I understand that a claim limitation is disclosed for the purpose of anticipation if a POSA would have understood the reference to disclose the limitation based on inferences that a POSA would reasonably be expected to draw from the explicit teachings in the reference when read in light of the POSA’s knowledge and experience.

22. I understand that a claim limitation is inherent in a prior art reference if that limitation is necessarily present when practicing the teachings of the reference, regardless of whether a person of ordinary skill recognized the presence of that limitation in the prior art.

B. Obviousness

23. I understand that a patent claim may be unpatentable if it would have been obvious in view of a single prior-art reference or a combination of prior-art references.

24. I understand that a patent claim is obvious if the differences between the subject matter of the claim and the prior art are such that the subject matter as a whole would have been obvious to a person of ordinary skill in the relevant field at the time the invention was made. Specifically, I understand that the obviousness question involves a consideration of:

- the scope and content of the prior art;
- the differences between the prior art and the claims at issue;
- the knowledge of a person of ordinary skill in the pertinent art; and
- if present, objective factors indicative of non-obviousness, sometimes referred to as “secondary considerations.” To my knowledge, the Patent Owner has not asserted any such secondary considerations with respect to the '058 patent.

25. I understand that in order for a claimed invention to be considered obvious, a POSA must have had a reason for combining teachings from multiple prior-art references (or for altering a single prior-art reference, in the case of obviousness in view of a single reference) in the fashion proposed.

26. I further understand that in determining whether a prior-art reference would have been combined with other prior art or with other information within the knowledge of a POSA, the following are examples of approaches and rationales that may be considered:

- combining prior-art elements according to known methods to yield predictable results;
- simple substitution of one known element for another to obtain predictable results;
- use of a known technique to improve similar devices in the same way;
- applying a known technique to a known device ready for improvement to yield predictable results;
- applying a technique or approach that would have been “obvious to try,” *i.e.*, choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success.
- known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations would have been predictable to one of ordinary skill in the art;
- some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior-art reference or

to combine prior-art reference teachings to arrive at the claimed invention. I understand that this teaching, suggestion or motivation may come from a prior-art reference or from the knowledge or common sense of one of ordinary skill in the art.

27. I understand that a universal motivation known in a particular field to improve technology can provide a motivation to combine prior art references even absent any hint or suggestion in the references themselves. I also understand that obviousness is determined in light of all the facts and that a given course of action often has simultaneous advantages and disadvantages, and that this does not necessarily obvious motivation to combine teachings from multiple references.

28. I understand that for a single reference or a combination of references to render the claimed invention obvious, a POSA must have been able to arrive at the claimed invention by modifying, implementing, or combining the teachings of the applied references.

IV. UNPATENTABILITY GROUNDS

29. As I explain in Section VII below, it is my opinion that Callender renders claims 1-18 obvious.

30. The '058 patent's earliest-alleged priority date on the patent itself is September 22, 2003. *See* EX1001, code (60). I understand that Callender, which

was filed on June 26, 2002, qualifies as prior art to the '058 patent.¹ Callender was not relied on by the examiner during prosecution of the '058 patent. *See* EX1002.

V. THE '058 PATENT

A. Background and Specification

31. A typical computer system may run multiple “application[s]” (which the '058 patent also refers to as “application programs”), such as email or word-processing, that must make shared use of the computer system’s resources.

EX1001, 1:21-28 (“Computer systems are designed in such a way that software application programs share common 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.”). POSAs understood that such “resources” including things like the

¹ I am informed and understand that VirtaMove may claim that the alleged invention of the '058 patent had a conception date sometime earlier than the priority date listed on the face of the patent. For purposes of this declaration, I have been asked to assume that the earliest priority date for the '058 patent is the earliest priority date alleged on the face of the patent, which is September 22, 2003 – the filing date of provisional application No. 60/504,213. *See* EX1001, code (60).

computer's hardware devices, or certain key files. EX1001, 1:28-31; Microsoft (EX1035), 378 (“operating system” defined as “[t]he software that controls the allocation and usage of hardware resources such as memory, central processing unit (CPU) time, disk space, and peripheral devices”).

32. The computer's “operating system (OS)” (EX1001, 2:1) “traditionally...provides mechanisms to...control [the applications'] access to shared resources.” EX1001, 1:20-24. Additionally, the '058 patent states that the OS “‘normally’ supplie[s]” “service[s] or part[s] of a service” that are “critical to the operation of a software application”; the patent calls such services, or parts thereof, “critical system elements” (“CSEs”). EX1001, 6:6-9. Examples of CSEs include services such as networking, access to files, “memory allocation” and “device access.” EX1001, 5:38-45, 6:12-28, 9:33-35.

33. CSEs are “normally” found in the OS's “kernel.” EX1001, 5:21-23. The “kernel” is the “core” of an OS and performs tasks like “manag[ing] memory, files, and peripheral devices” and “allocat[ing] system resources.” Microsoft (EX1035), 300; *see also* Callender, 1:40-43 (“Because the operating system kernel acts as a gatekeeper to computer resources, direct access to resources is generally limited to kernel mode processes.”). POSAs understood that peripheral devices include things like disks or network interfaces. Microsoft (EX1035), 398 (“peripheral device,” or “peripheral,” is “[i]n computing, a device, such as a disk

drive, printer, modem, or joystick, that is connected to a computer and is controlled by the computer's microprocessor.”).

34. POSAs understood that, to enable the kernel to perform the tasks I mentioned in the paragraph above, conventional computer processors run in a special “kernel mode,” which has unrestricted access to the computer's resources, when executing the kernel. EX1001, 6:32-36 (“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.”); Callender, 1:37-48 (“Processes running in kernel mode are privileged and have access to all computer resources (such as all available memory), without the restrictions that apply to user mode processes. Because the operating system kernel acts as a gatekeeper to computer resources, direct access to resources is generally limited to kernel mode processes. Distinctions between user mode processes and kernel mode processes also may be supported by computer hardware. For example, many microprocessors have processing modes to support the distinctions between user mode processes and kernel mode processes.”).

35. In contrast, when running applications, processors typically run in “user mode,” where access to certain resources is restricted to prevent applications from interfering with each other or damaging the system—application code cannot

run in kernel mode. EX1001, 6:31 (“User mode: The context in which applications execute.”), 6:32-36 (“In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.”); Callender, 1:32-48 (“Generally, application processes run within user mode so that the processes are isolated and cannot interfere with each other’s resources. ... Processes running in kernel mode are privileged and have access to all computer resources (such as all available memory), without the restrictions that apply to user mode processes. Because the operating system kernel acts as a gatekeeper to computer resources, direct access to resources is generally limited to kernel mode processes. Distinctions between user mode processes and kernel mode processes also may be supported by computer hardware. For example, many microprocessors have processing modes to support the distinctions between user mode processes and kernel mode processes.”).

36. Applications often need to use CSEs that are typically provided by the kernel. *See* EX1001, 5:38-58 (describing an application using “TCP/IP” services that are normally supplied by the “kernel” of the “Linux” operating system).

37. POSAs understood that to access kernel services, applications typically make “system calls” to request that the OS kernel perform a needed service (like a CSE) for the application, using the processor executing in kernel

mode. For example, when discussion prior art ways to use CSEs, the '058 patent explains:

In order for an application of FIG. 1 *to make use of a critical system element 17 in the kernel 16, the application 12 a or 12 b 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.*

EX1001, 6:66-7:16.²

38. POSAs understood that the scheme I discussed in the previous paragraph, in which applications use CSEs by making system calls, has some known disadvantages. One disadvantage is that switching processor modes

² All emphases added unless otherwise indicated.

between user and kernel mode takes time, and thus could negatively impact performance. My testimony is corroborated by, *e.g.*, Callender, 1:50-56 (referring to “the performance degradation associated with switching process modes” between user and kernel mode). Another disadvantage is that, if all applications are invoking a CSE by making a system call to the kernel, then all applications have to use whatever version of the CSE that the computer system’s OS kernel provides. As the ’058 patent explains, this could be disadvantageous because different applications may require different versions of a CSE for optimal performance. *See* EX1001, 5:54-6:3 (explaining potential performance and security issues arising from two different applications using the same version of “TCP/IP services”).

39. Because of the disadvantages I discussed in the previous paragraph, it was also known to implement some CSEs in user mode instead of in kernel mode, as the ’058 patent admits. *See* EX1001, 7:23-25 (discussing prior-art “system architecture where critical system elements 27a, 27b execute in user mode”).

40. In some known user-mode CSE implementations, the CSE was provided by a “process[]” (which the ’058 patent also refers to as a “server”) that executed in user mode but was separate from any application that used the CSE. *See* EX1001, 7:23-30 (“FIG.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 FIG.2a is the GNU Hurd operating system.”). In other words, when a user-mode application needed a CSE, it would request for a different user-mode process, rather than the kernel, to perform the CSE.

41. In the prior-art user-mode CSE implementations that the '058 patent discusses, either different user-mode processes provided different user-mode CSEs, or a single user-mode process provided all user-mode CSEs. *See* EX1001, 7:23-61 (discussing the prior art scheme involving user-mode processes with respect to Figures 2a and 2b, and noting that “[t]he essential difference between the architecture described in FIG.2a and FIG. 2b is the use of a single process context to contain all user mode critical system elements”), Fig. 2a (below, showing separate “critical system element” 27a and “critical system element” 27b), Fig. 2b (below, showing just one circle labeled “critical system *elements*” plural). As illustrated in Fig. 2a (below), however, communication between a user-mode application needing a CSE and a user-mode process providing that CSE still passed through the kernel—that is, the application issued a request to the kernel, which then invoked the user-mode CSE process. EX1001, 7:31-52.

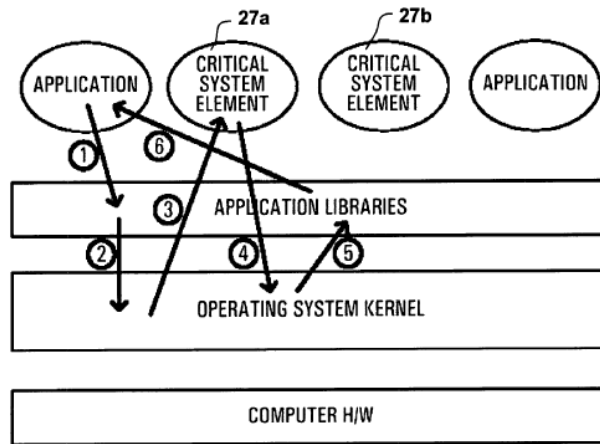


FIG. 2a
Prior Art

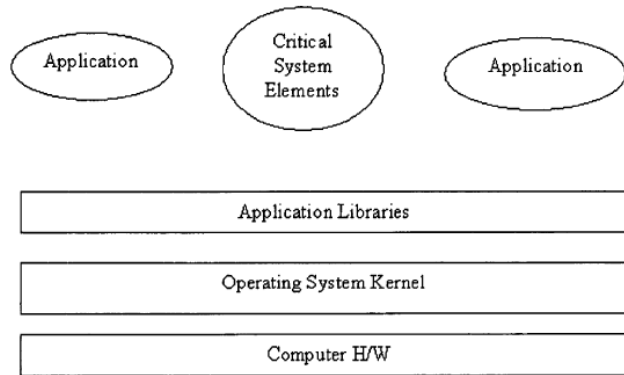


FIG. 2b
Prior Art

42. In contrast to the above-described scheme requiring communication with a separate user-mode CSE process via the kernel, the '058 patent discloses “replicat[ing] [CSEs] in user mode” by “placing CSEs similar to those in the OS in shared libraries.” EX1001, 5:22-34.

43. “The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions.” EX1001, 8:27-28. POSAs

understood that a “function” is a predefined set of instructions that carry out a specific action (*e.g.*, a CSE). My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, Hatch (EX1007), 1:22-33 (“The software that instructs a computer or microprocessor to perform tasks is called a program. One type of program is a function, which usually performs a particular service or a series of discrete tasks. For instance, a function named DISPLAY could instruct a computer to display a value on a monitor. Functions are typically compiled within a larger program or are compiled within a library or collection of functions. Generally, a function is invoked by calling the function. When a function is called, parameter values are typically passed to the function to tailor the task that the function will perform.”).

44. The ’058 patent says that placing a CSE in a shared library “provides a means of attaching or linking a CSE service to an application having access to the shared library.” EX1001, 5:29-31. *See also* EX1001, 9:15-27 (describing how “according to the invention, some system elements that are critical to the operation of a software application” are “contained in a shared library”).

45. The ’058 patent says the “functions” in the shared CSE library “can be *directly* called by the applications...and as such can be run in the same context as the applications.” EX1001, 8:31-33; *see also id.*, 9:41-42 (“FIG. 4 shows that the invention allows for [CSEs] to exist in the same context as an application.”), FIG.

4. In other words, POSAs understood that the user-mode application itself performs the CSE (by calling a function from the library), rather than asking another user-mode process to perform the CSE.

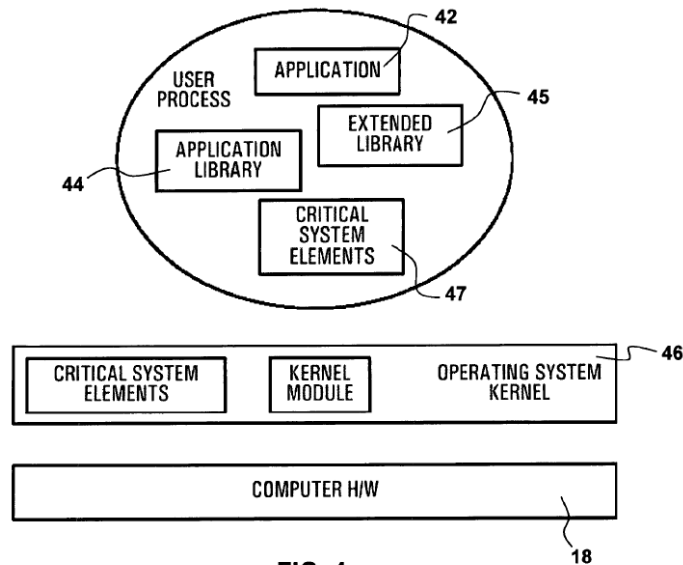


FIG. 4

46. The patent refers to user-mode CSEs in a shared library as “SLCSEs,” and to CSEs in the kernel as “OS [operating system] critical system elements (OSCSEs).” EX1001, 2:7-12. The patent mentions the well-known “dynamic linked library (DLL)” as an example of a “shared library.” EX1001, 2:45-51 (“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.”).

47. The patent contends that the ability to “execute” CSEs “in the same context as an application” “contrast[s]” with prior-art user-mode CSEs that were allegedly only implemented as a “shared service.” EX1001, 1:46-54. However, the patent admits, consistent with a POSA’s understanding, that the “sharing of code” between applications via a shared library in user mode was “common practice.” EX1001, 3:30-33. *See also* EX1001, 7:3-5 (“As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access.”). The patent also admits, again consistent with a POSA’s understanding, that it was “typical” to provide “functions” in “libraries which applications link with.” EX1001, 6:37-45 (“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....”).

48. The ’058 patent is silent with respect to the alleged novelty of using linking and shared libraries to make CSEs available to applications. As I demonstrate in my analysis in Section VII below, Callender disclosed using linking and shared libraries to make CSEs available to applications before the ’058 patent was filed.

B. Person of Ordinary Skill in the Art (“POSA”)

49. I have been informed and understand that for purposes of assessing whether prior-art references disclose every element of a patent claim (thus “anticipating” the claim) and/or would have rendered the claim obvious, the patent and the prior-art references must be assessed from the perspective of a person having ordinary skill in the art (“POSA”) to which the patent is related, based on the understanding of that person at the time of the patent claim’s priority date. I have been informed and understand that a POSA is presumed to be aware of all pertinent prior art and the conventional wisdom in the art, and is a person of ordinary creativity. I have applied this standard throughout my declaration.

50. In my opinion, a POSA as of the ’058 patent’s September 22, 2003 earliest claimed priority date would have had at least a bachelor’s degree in computer science, computer engineering, or a related field, with three years of academic and/or industry experience in the areas of “computing system[s]” and “application libraries.” EX1001, 1:15-17. More education may substitute for less experience.

51. By 2003, I held a Ph.D. in computer science, and I had over 10 years of experience with computing systems and application libraries. Thus, I was a person of more than ordinary skill in the art during the relevant timeframe. However, I worked with many people who fit the characteristics of the POSA, and

I am familiar with their level of skill. When developing the opinions set forth in this declaration, I assumed the perspective of a person having ordinary skill in the art, as set forth above.

C. Prosecution History

52. I have reviewed the prosecution history of the '058 patent (EX1002).

53. The examiner rejected the originally-filed claims over Cabrero (EX008). EX1002, 216-222. In response, the applicants amended claim 1 to recite that SLCSEs are “replicas of OSCSEs.” EX1002, 235. The examiner then again rejected the claims over Cabrero. EX1002, 265-266. An interview between the applicants and the examiner then occurred, and after this interview, the applicants amended claim 1 to require SLCSEs to be “functional” replicas of OSCSE. EX1002, 273, 278.

54. The examiner then rejected the claims over O'Rourke (EX1009) in combination with Peek (EX1010). EX1002, 296-297. In response, the applicants argued that O'Rourke did not meet the “functional replica” requirement because O'Rourke disclosed a user-mode “proxy” that “merely acts as an intermediary” to the kernel. EX1002, 323. The applicants also argued that a POSA would not have combined O'Rourke with Peek. EX1002, 324-325.

55. After this rejection, another interview between the applicants and examiner was conducted. EX1002, 348-351. At this interview, O'Rourke and

Peek were not discussed; the reason why they were not discussed is not apparent from the file history. Instead, the references discussed at the interview were Elnozahy (EX1011) and Wong (EX1012).

56. After this interview, the examiner allowed the claims with an amendment to Element [1E] (*see* claim listing in Section VIII below) to recite “at least a first” and “at least a second” of the plurality of software applications. EX1002, 350-351. The examiner also incorporated into claim 1 originally-filed dependent claim 6, which is essentially limitation [1F]’s “wherein” clause. EX1002, 350-352.

57. The examiner stated that neither Elnozahy nor Wong disclosed the amended language of claim 1. I have reviewed Elnozahy and Wong, and neither reference explicitly discloses shared libraries or an SLCSE in a shared library, and there is no indication that the examiner considered whether these features would have been obvious. Furthermore, I see no record in the file history of Callender having been before the examiner during prosecution.

D. Challenged Claims

58. I have been asked to determine whether claims 1-18 would have been obvious over Callender. Claims 1-18 are reproduced in the Claim Listing (§VIII below).

VI. CLAIM CONSTRUCTION

59. I am informed and understand that in litigation filed by the Patent Owner (“VirtaMove”) against Google, both sides have taken positions on the construction of certain claim terms. I am also informed and understand that there has been no ruling in the litigation on claim construction yet.

60. In my analysis in Section VII below, for those claim elements where the parties have proposed alternative constructions in litigation, I present alternative mappings of the prior art under both parties’ proposed constructions of the disputed terms, thus demonstrating unpatentability regarding of which party’s construction is correct. In addition, for claim elements where the parties in litigation agreed to a construction, I have shown how the prior art meets the agreed construction.

VII. CALLENDER RENDERS CLAIMS 1-18 OBVIOUS

61. As I explain below, in my opinion, claims 1-18 would have been obvious to a POSA in view of Callender and the background knowledge of a POSA.

A. Callender

62. Callender discloses techniques for “a single implementation of operations that are common to both kernel mode processing and user mode processing, relative to a hardware adapter.” Callender, 2:28-31.

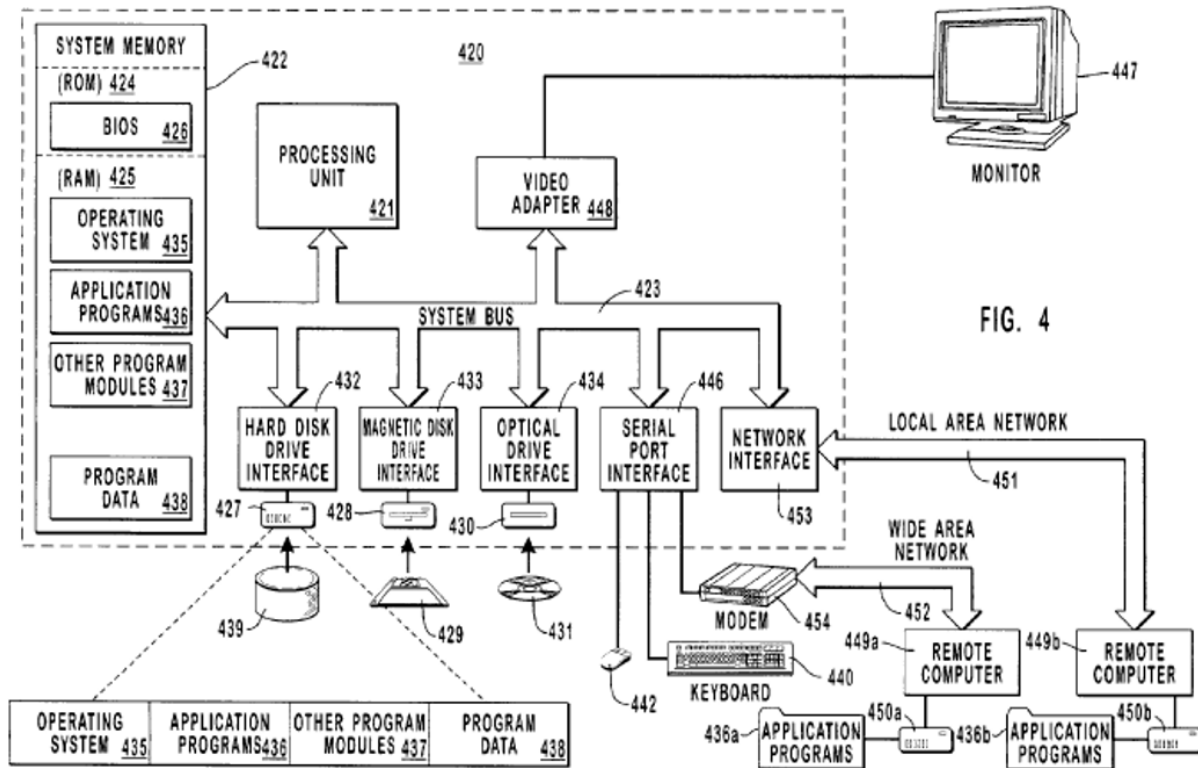
63. An example of an “adapter” that Callender’s techniques can be used with is a “host channel adapter.” Callender, 4:51-57 (“FIGS. 2A-2C illustrate an example hardware driver model that allows for a single implementation of common user mode and kernel mode operations *in accordance with the present invention*. InfiniBand application 210A accesses *host channel adapter* (‘HCA adapter’) 280A through user mode library 230A and through kernel mode miniport driver 240A.”). POSAs understood that an HCA adapter enables communication using the well-known “InfiniBand” networking protocol. My testimony regarding a POSA’s background knowledge of InfiniBand and HCAs is corroborated by, for example: Kagan (EX1013), [0007] (“Packet network communication adapters are a central element in new high-speed, serial input/output (I/O) bus architectures that are gaining acceptance in the computer industry. In these systems, computing hosts and peripherals are linked together by a switching network, commonly referred to as a switching fabric, taking the place of parallel buses that are used in traditional systems. A number of architectures of this type have been proposed, culminating in the ‘InfiniBand™’ (IB) architecture...”), [0008] (“The host network adapter in IB systems is known as a host channel adapter (HCA).”); Green (EX1014), 1:50-60 (“Another, newer, protocol known as Infiniband can carry data at rates exceeding 2.5 Gbps in each direction.”).

64. Callender says that examples of “operations” that are “common” to both user- and kernel-mode processing include “operations” for “sending and receiving” information through the adapter. Callender, Abstract (“Methods, systems, and computer program products that, by defining a common interface, allow for a single implementation of operations common to both kernel mode and user mode processing, relative to a hardware adapter...The common operation may provide a user mode process direct access to a hardware adapter, such as for sending and receiving information, without switching to kernel mode.”), 2:41-50 (“The common operation may provide a user mode process direct access to a hardware adapter without switching to kernel mode...operations for sending and receiving information through the adapter may be common to both user mode processing and kernel mode processing.”), 3:40-4:16 (“FIG. 1 shows a high-level block diagram of a hardware driver model that allows for a single implementation of common user mode and kernel mode operations in accordance with the present invention.... Having corresponding operations implemented in both user mode implementation 130 and kernel mode implementation 140 is desirable for at least a couple of reasons. First, kernel mode implementation 130 generally includes all operations possible for adapter 150 so that applications without user mode interface 120 are able to interact with adapter 150.... frequent operations like sending and receiving information are likely candidates to be included in user

mode implementation 130...”), 4:51-5:5 (“FIGS. 2A-2C illustrate an example hardware driver model that allows for a single implementation of common user mode and kernel mode operations in accordance with the present invention.... Initiating and terminating access are controlled through kernel mode operations to assure an appropriate level of security. After initiation, however, security for frequent communication operations, such as those related to sending and receiving data, is enforced at the HCA hardware, allowing for extremely fast user mode access.”), claim 4 (reciting “the at least one operation that is common to kernel mode processing and user mode processing corresponds to a communication operation for sending and receiving information through the hardware adapter”), claim 12 (similar), claim 22 (similar).

65. Callender techniques are performed in a “computer” with a conventional “operating system.” Callender, 8:19-9:5 (“With reference to FIG. 4, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional computer 420.... The computer 420 may also include a magnetic hard disk drive 427 for reading from and writing to a magnetic hard disk 439, a magnetic disk drive 428 for reading from or writing to a removable magnetic disk 429... Program code means comprising one or more program modules may be stored on the hard disk 439, magnetic disk 429, optical

disc 431, ROM 424 or RAM 425, including an operating system 435....”), Fig. 4 (below).



66. Callender’s computer also has various hardware components, including “adapter[s],” which POSAs understood are network adapters, since Callender describes these adapters as being used for networking. Callender, 9:22-24 (“When used in a LAN networking environment, the computer 420 is connected to the local network 451 through a network interface or adapter 453.”). As I discussed in paragraph 63 above, an HCA adapter is used for the InfiniBand networking protocol. Thus, POSAs understood Callender to disclose a “computer” that includes an “adapter” such as an HCA adapter.

67. Callender’s Figure 1 illustrates “a hardware driver model that allows for a single implementation of common user mode and kernel mode operations.” Callender, 3:39-44. POSAs understood that a “driver” is software by which a computer interacts with a hardware device such as an adapter. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, Microsoft (EX1035), 177 (“driver” defined as “hardware device or a program that controls or regulates another device.... A software driver is a device-specific control program that enables a computer to work with a particular device, such as a printer or a disk drive.”).

68. In Callender’s model, a “user mode implementation” of an operation provides applications with “direct access to [the] adapter...without switching to kernel mode.” Callender, 3:45-49. Callender’s model also comprises a “kernel mode implementation” of “all operations possible for [the] adapter,” including “operations unique to kernel mode” and operations common to both modes, and “frequent operations like sending and receiving information” are operations for the adapter that are “likely candidates to be included” in both user- and kernel-mode implementations. Callender, 3:52-4:12 (“Having corresponding operations implemented in *both user mode implementation 130 and kernel mode implementation 140* is desirable for at least a couple of reasons. First, kernel mode implementation 130 generally includes *all* operations possible for adapter 150...

kernel mode implementation 140 generally includes operations that are unique to kernel mode... as indicated above, frequent *operations like sending and receiving* information are likely candidates to be included in user mode implementation”).

69. Callender’s Figure 2A (annotated below) shows a more specific use case for the general model of Figure 1—an “example hardware driver model that allows for a single implementation of common user mode and kernel mode operations” specifically for an HCA adapter. Callender, 4:51-57, Fig. 2A.

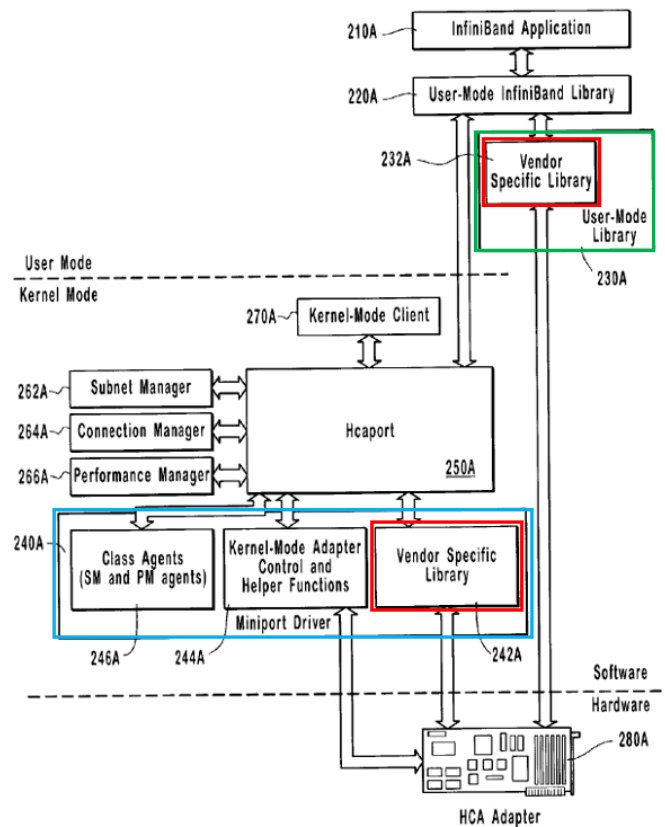


Fig. 2A

70. As Figure 2A shows, the example model includes a “kernel mode miniport driver” in kernel mode and a “user mode library” in user mode.

Callender, 4:55-59.

71. Callender says that an “InfiniBand application 210A... accesses [HCA adapter] 280A through user mode library 230A and through kernel mode miniport driver 240A.” Callender, 4:54-57. POSAs understood from the name “InfiniBand application” that this application uses the InfiniBand networking protocol. POSAs also understood that this application is a user-mode application, at least because Figure 2A shows this application on the “User Mode” side of the “User Mode” v. “Kernel Mode” dividing line.

72. The “user mode library” is “implemented as a dynamic link library (‘DLL’).” Callender, 5:13-15. POSAs understood that the prior art used the term “dynamic link library” to refer to the same thing the patent refers to as a “dynamic linked library,” as corroborated by the use of the same abbreviation “DLL.” *See* Callender, 5:13-15; EX1001, 2:48 (“dynamic linked library (DLL)”).

73. As I explain in paragraphs 74-77 below, POSAs understood that Callender’s “user mode library” includes “operations” for “sending and receiving” data via the HCA adapter that are also implemented in the “kernel mode miniport driver.” Callender, 4:4-12, 4:55-59. In other words, just as the ’058 patent describes (as I discussed in Section V.A above), the same service is provided by

both a user-mode DLL and the kernel, and those applications written to call the DLL can use the user-mode service, while others are able to use the kernel-mode service.

74. Callender explains that “hardware-specific operations” for the HCA adapter are implemented in a “**vendor specific library**,” also called a “**process mode independent (‘PMI’) HCA IO [input/output] access library**.” Callender, 5:20-28 (“The vendor specific library 232A of user mode library 230A maps the abstraction of InfiniBand library 220A to the hardware-specific operations of HCA adapter 280A. The same vendor specific library appears as vendor specific library 242A of miniport driver 240A...Accordingly, vendor specific library 232A and vendor specific library 242A represent, at least in part, a process mode independent (PMI) HCAIO access library.”), Fig. 2A (annotated below).

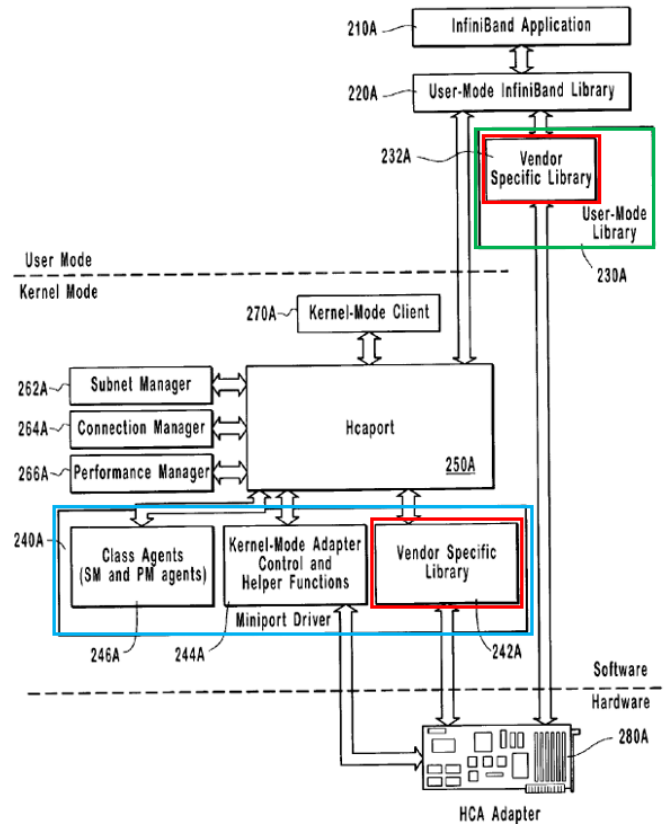


Fig. 2A

75. The “PMI HCA IO access library” (also called the “vendor specific library,” as I discussed in the previous paragraph) is incorporated into both the user mode library and the kernel-mode miniport driver. Callender, 5:19-42 (“The vendor specific library 232A of user mode library 230A maps the abstraction of InfiniBand library 220A to the hardware-specific operations of HCA adapter 280A. The same vendor specific library appears as vendor specific library 242A of miniport driver 240A.”), 6:45-56 (“The process mode independent library (e.g., vendor specific library 232C and vendor specific library 242B) are statically linked into both miniport 240C and user mode library 230C.”), Fig. 2A (annotated above).

76. POSAs understood that “operations” for “sending and receiving” information through the adapter (Callender, Abstract, 2:41-52, 3:53-4:16, 4:51-5:5) are examples of “IO [input/output] access” (Callender, 5:26-28), because those operations require sending data into and reading data out of, respectively, the adapter. Thus, POSAs understood that the “IO access” features of the “vendor specific library”/“PMI HCA IO access library” (*see* Callender, 5:19-27) are used by both the user-mode and kernel-mode versions of the “operations for sending and receiving” information (*e.g.*, Callender, 2:40-52, 3:40-4:16, 4:51-66), through the HCA adapter. For example, at the beginning of the discussion of Figures 2A-2C, in which the discussion of the “vendor specific library”/“PMI HCA IO access library” (*see* Callender, 5:19-27) appears, Callender says

FIGS. 2A-2C illustrate an example hardware driver model that allows for a **single implementation of common user mode and kernel mode operations in accordance with the present invention....** Initiating and terminating access are controlled through kernel mode operations.... After initiation, however, security for frequent communication operations, such as those related to sending and receiving data, is enforced at the HCA hardware.... For example...kernel mode miniport driver 240A is responsible for creating and destroying **communication queues**, whereas user mode library 230A permits user mode data transfer to the communication queues at HCA adapter 280A.”

Callender, 4:51-5:5. Thus, POSAs understood that, after the kernel initiates access to the adapter, both the user-mode and kernel-mode operations can use the “vendor specific library”/“PMI HCA IO access library” (*see* Callender, 5:19-27) to access “communication queues at [the] HCA adapter” (Callender, 5:4-5) that are used for transferring data to and from the adapter, since the introductory passage at 4:51-5:5 refers to sending and receiving as an example of implementation of “common **user mode and kernel mode** operations.”

77. Additionally, Callender refers to “kernel-bypass IO [input/output] features offered by the user mode library 230A.” Callender, 5:15-18, Fig. 2A (annotated below, showing arrow from “User-Mode Library 230” to “HCA Adapter 280A” bypassing the “Kernel Mode”). As I noted in the previous paragraph, POSAs understood that Callender’s “operations” for “sending and receiving” information through the adapter are examples of “IO access” operations. Additionally, Callender notes that “some software drivers bypass kernel mode for certain operations.” Callender, 1:59-61. Therefore, POSAs understood that the user mode implementations of the “sending” and “receiving” “operations” are contained in the “user mode library 230A” and utilize the “kernel-bypass IO features.”

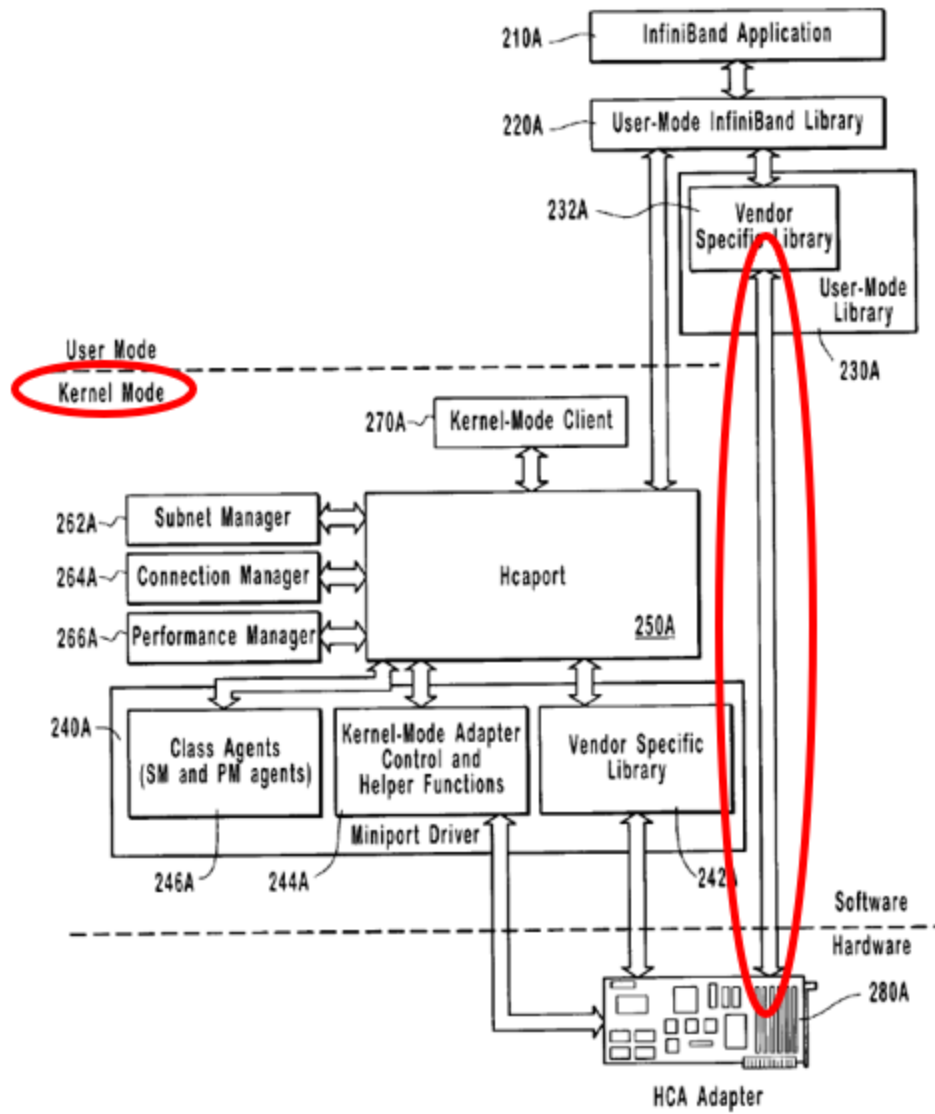


Fig. 2A

78. Finally, POSAs also understood that the sending and receiving “operations” in the “user mode library” are implemented as functions. As I noted in Section V.A above, POSAs understood that a function is a predefined set of instructions that carry out a specific action. Furthermore, Callender discloses “functions” in a library. Callender, 5:19-28 (referring to “InfiniBand library

220A” and “vendor specific library” / “process mode independent (‘PMI’) HCA IO access library”), 6:45-56 (describing “functions” in the “PMI library” that is part of the user mode library). To the extent Callender does not disclose implementing the sending and receiving operations as functions, this would have been obvious to a POSA. The use of functions was a well-known and customary software technique that was within a POSA’s skill. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Hatch (EX1007), 1:22-29 (“The software that instructs a computer or microprocessor to perform tasks is called a program. One type of program is a function, which usually performs a particular service or a series of discrete tasks....Functions are typically compiled within a larger program or are compiled within a library or collection of functions”); Petzold (EX1015), 959 (“The term dynamic linking refers to the process that Windows uses to link a function call in one module to the actual function in the library module.”), 960 (“Thus, one purpose of dynamic link libraries is to provide functions and resources that can be used by many different programs.”). Furthermore, POSAs understood that implementing the sending and receiving operations as functions would beneficially provide a predefined set of instructions for performing a repeated operation like sending or receiving.

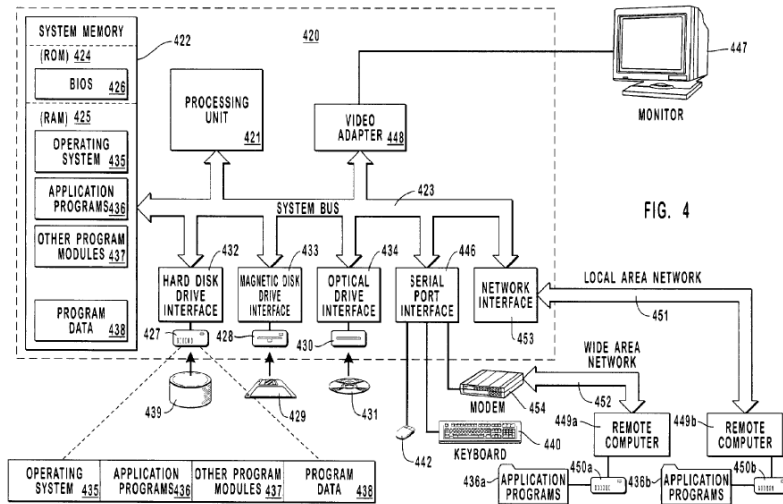
B. Claim-by-Claim Analysis

1. Claim 1

- a. [1PRE]: “A computing system for executing a plurality of software applications comprising:”

79. Callender’s independent claims 1 and 9 recite a “*computer system*.”

Additionally, Callender’s Figure 4 depicts an “exemplary *system* for implementing [its] invention.” Callender, 8:19-20. This “system” includes a “*computer*” that has a “*system memory*” and a “*system bus*.” Callender, 8:19-24, Fig. 4. POSAs would understand that Callender’s “computer” is a *computing system*.



80. Callender’s computer includes a “processing unit” connected to “system memory” storing “application *programs*”:

With reference to FIG. 4, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional *computer 420*, including a processing unit 421, a system memory 422, and a system bus 423 that couples various system

components including the system memory 422 to the processing unit 421....

The computer 420 may also include a magnetic hard disk drive 427 for reading from and writing to *a magnetic hard disk 439*, a magnetic disk drive 428 for reading from or writing to a removable magnetic disk 429...

Program code means comprising one or more program modules may be stored on the hard disk 439, magnetic disk 429, optical disc 431, ROM 424 or RAM 425, *including* an operating system 435, *one or more application programs* 436, other program modules 437, and program data 438.

Callender, 8:19-58, Fig. 4.

81. Additionally, when discussing Fig. 1, which is a “block diagram of [Callender’s] hardware driver model” (3:10-13), Callender says the “model” lets “*applications*” (plural) “interact with adapter 150.” Callender, 3:40-59 (“FIG. 1 shows a high-level block diagram of a hardware driver model that allows for a single implementation of common user mode and kernel mode operations in accordance with the present invention.... Having corresponding operations implemented in both user mode implementation 130 and kernel mode implementation 140 is desirable for at least a couple of reasons. First, kernel mode implementation 130 generally includes all operations possible for adapter 150 so

that *applications* without user mode interface 120 are able to interact with adapter 150.”).

82. Thus, Callender’s system includes a *plurality* of “*applications*” (3:57), which Callender also calls “application programs” (8:57), that interact with the adapter.

83. POSAs understood that these applications are *software applications* at least because Callender uses “application” to refer to a “*software*” application. Callender, 1:18-29 (“With the increasing performance of computer hardware, the operation of computer *software* is becoming a more significant factor in overall system performance....One typical communication bottleneck relates to various *software layers* and the corresponding transitions between process modes that often occur between an application and a hardware adapter.”), Fig. 1 (below, showing “Application” 110 running in “user mode” “layer” of “*Software*”)

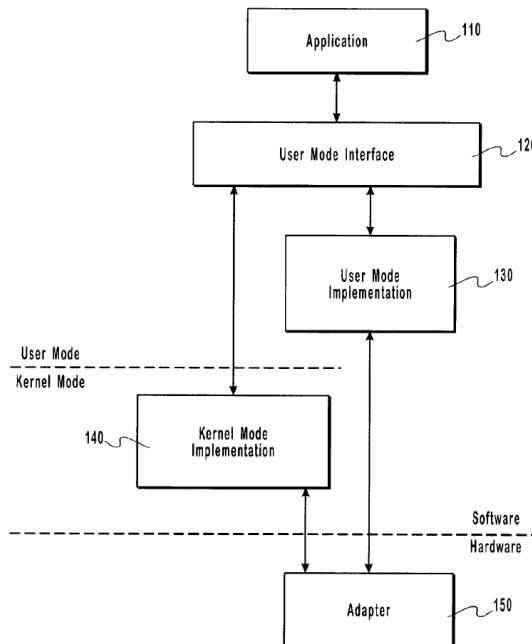


Fig. 1

84. POSAs further understood that the “applications” comprise “computer-*executable* instructions.” Callender, 9:59-62 (Callender’s invention is “described in the general context of computer-*executable* instructions”).

85. Finally, POSAs understood that the “processing unit” in Callender’s “system” *executes* the application programs’ instructions (*see* my discussion of Callender’s “processing unit” and “system” in paragraph 80 above, citing Callender, 8:19-58, Fig. 4). POSAs understood that executing instructions is the well-known purpose of a processing unit. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, Linden (EX1016), [0005] (“Information processing system or computing systems comes in many configurations. They include without limitation the generally known computers....

For example, computers have general-purpose central processing units (CPU) which are designed to execute a specific set of instructions.”).

86. Thus, Callender’s “computer” (8:21) is a *computing system for executing a plurality of software applications*.

b. [1A] “a) a processor;”

87. Callender’s “computer” includes a “*processing* unit.” Callender, 8:19-24, Fig. 4. POSAs understood that a “processing unit” is a “*processor*.” I understand that in litigation regarding the ’058 patent, VirtaMove has also pointed to a “CPU,” *i.e.*, “central processing unit,” in the accused products as being the claimed *processor*. See EX1020, 3-4; see also Microsoft (EX1035), 92 (“central processing unit” is “CPU”).

88. I understand that in litigation regarding the ’058 patent, the parties have agreed to construe the claim term *processor* as a physical computer processor. POSAs understood that Callender’s “processing unit” is a physical computer processor. See Callender, 8:19-9:5 & Fig. 4 (describing hardware in the “computer” including the “processing unit”). Thus, Callender’s “processing unit” meets the agreed litigation construction of *processor*. See EX1058, 6.

c. [1B]

i. [1B.1] “b) an operating system having an operating system kernel...”

89. Callender’s “computer” *comprises* an “*operating system*.” Callender, 8:54-58 (“Program code means” in the Figure 4 computer “comprising one or more program modules may be stored on the hard disk 439, magnetic disk 429, optical disc 431, ROM 424 or RAM 425, including an operating system 435, one or more application programs 436, other program modules 437, and program data 438.”), Fig. 4; *see also* Callender, 5:55-65 (“Hcaport 250B establishes an operating environment that provides miniport 240B with operating system functionality, such as memory allocation, registry access, and port/register access.”).

90. Callender does not describe the “operating system” in any way that would suggest to a POSA that it is anything other than a conventional operating system. Thus, POSAs understood that Callender’s “operating system” is a conventional OS, which is “[t]he software that controls the allocation and usage of hardware resources such as memory, central processing unit (CPU) time, disk space, and peripheral devices.” Microsoft (EX1035), 378. This is consistent with Callender’s “Background” discussion of “operating systems” having an “operating system kernel” that “acts as a gatekeeper to computer resources.” Callender, 1:41-43.

91. Thus, Callender’s “operating system” meets Google’s litigation construction. *See* EX1059, 4. VirtaMove proposed in litigation that no construction was necessary. *Id.*

92. Callender references an “*operating system kernel*” in the “Background.” Callender, 1:39-42. Callender also describes its inventive “hardware driver model” (3:40-43) as having “kernel mode” components. *E.g.*, Callender, Abstract, 4:41-5:5.

93. Additionally, Callender says the “user mode library” provides “kernel-bypass IO features.” Callender, 5:15-18. POSAs thus understood that the “operating system” in Callender’s computer *has an operating system kernel*.

94. At minimum, POSAs would have found it obvious to include a kernel in the OS. POSAs understood that it was customary for OSs to have kernels. My testimony regarding a POSA’s background knowledge is corroborated by the ’058 patent, which refers to the “operating system kernel” in the “conventional architecture” illustrated in Figure 1. EX1001, 6:62-7:16 My testimony is also corroborated by, *e.g.*, Draves (EX1017), 1:38-47 (explaining in “BACKGROUND” section that “Critical operating system components are implemented in kernel mode.”). In my opinion, POSAs would reasonably have expected success with such an implementation, since having a kernel in the OS was conventional.

95. Callender does not describe the “kernel” in any way that would suggest to a POSA that it is anything other than a conventional OS kernel. Thus, POSAs understood that Callender’s “kernel” is a conventional kernel, which POSAs understood is the “core...portion” of an operating system “that manages memory, files, and peripheral devices; maintains the time and date; launches applications; and allocates system resources.” Microsoft (EX1035), 257. This is consistent with Callender’s “Background” discussion of the “operating system kernel” as a “gatekeeper to computer resources.” Callender, 1:41-43.

96. Thus, Callender’s “kernel” meets Google’s litigation construction. EX1059, 4. VirtaMove has proposed that no construction is necessary. *Id.*

97. Thus, Callender’s computer *comprises an operating system having an operating system kernel.*

ii. [1B.2] “[OS kernel] having OS critical system elements (OSCSEs)...”

98. The ’058 patent says a *critical system element* (CSE) is “[a]ny service or part of a service, ‘normally’ supplied by an operating system, that is critical to the operation of a software application.” EX1001, 6:6-8. In my opinion, the outer boundaries of what is “normally supplied” and “critical” in the ’058 patent’s definition of *CSE* are not well-defined to a POSA. However, the patent says examples of CSEs include “[n]etwork services” including “message passing protocols.” EX1001, 6:12-13 (providing “EXAMPLE CSEs”).

99. The patent also notes that a “kernel may provide a TCP/IP service” and describes this as an example of a CSE. EX1001, 3:22-30. POSAs understood that a TCP/IP service was a type of network service. My testimony regarding a POSA’s understanding is corroborated by, *e.g.*: Gardiner (EX1066), [0004] (referring to “TCP/IP” as a “network protocol”).

100. Callender discloses “operations” for “sending” information to and “receiving” information from the HCA adapter. Callender, Abstract, 2:40-52, 4:51-5:5; *see also* my discussion of Callender in Section VII.A above.

101. HCA adapters are used for communication over a network using the InfiniBand protocol, as I also discussed in Section VII.A above (discussing Kagan (EX1013), [0007]-[0008]; Green (EX1014), 1:50-60).

102. Callender’s “operations for sending and receiving information from [an HCA] adapter” (Callender, 4:4-7) are therefore “[n]etwork services” (EX1001, 6:12-13).

103. Thus, while the outer boundaries of what is “normally supplied” and “critical” in the ’058 patent’s definition of *CSE* (EX1001, 6:6-8) are not well-defined, POSAs understood that at least Callender’s “operations” for “sending” information to and “receiving” information from the HCA adapter are within the scope of *CSEs*.

104. I understand that in litigation, VirtaMove has argued that the term *critical system element* should be construed as “[a]ny service or part of a service, ‘normally’ supplied by an operating system, that is critical to the operation of a software application.” EX1059, 14. In my opinion, Callender’s operations for sending and receiving are *CSEs* even if VirtaMove’s litigation construction were accepted, since these operations match examples of *CSEs* provided in the ’058 patent. I also note that VirtaMove’s claim-construction brief cites ““TCP/IP”” and “network services” as examples of *CSEs* in patent. *See* EX1067, 17.

105. The ’058 patent also says “[a] *CSE* is a dynamic object providing some function that is executing instructions used by the applications.” EX1001, 6:8-10. I understand that in litigation, neither party has asserted that this language imposes a claim requirement. *See also* EX1052, 2 (VirtaMove’s infringement contentions, which do not address the “dynamic object” language). However, if this is a requirement, then in my opinion, this would have been an obvious implementation of Callender, as I explain below.

106. POSAs understood that an “object” is a set of instructions and/or data. My testimony is corroborated by, *e.g.*, Braun (EX1021), [0053] (“Objects are a set of instructions and/or data which can be called by a pointer and/or an identifier and parameters to implement a specified function.”).

107. Callender's "operations" for "sending and receiving information" (4:8-9) are part of a "driver." Callender, 1:11-18 ("The present invention relates to the field of *driver* software for computer hardware. Specifically, the present invention relates to...a single implementation of one or more *operations* that are *common to both kernel mode processing and user mode processing* relative to the hardware adapter."), 3:10-15 ("FIG. 1 shows a high-level block diagram of a *hardware driver model* that allows for a single implementation of *common user mode and kernel mode operations* in accordance with the present invention."). POSAs also understood that "driver software" (Callender, 1:11-12) is customarily implemented as a set of instructions. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Saulpaugh (EX1103), 1:29-33 ("Drivers are typically software instructions that can be loaded into the computer system's memory and when executed will communicate with the device to properly configure the device for operation.").

108. It was known to POSAs to load drivers into memory as needed, *i.e.* dynamically. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*, Schmalz, (EX1024) 1:7-11 ("A system may have device driver software, which provides instructions on how to control or interface with a device within the system. The driver software may be allocated to different types of memories and subsequently de-allocated. Driver memory allocation may be

dynamic.”). It was also known that dynamic loading was one of two known options for loading of drivers or function libraries, *i.e.*, static and dynamic. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Walsh (EX1032), 1:36-59 (comparing static v. dynamic linking of libraries); Mitrovic (EX1104), 1:11-22 (“When a programmer develops a computer program, the source code of the program typically accesses many functions and variables within the program. These accesses are expressed in the source code as mere references to the names of the functions or variables. However, some time before the functions and variables can be accessed, the name must be bound to the memory location where either the entry point of the function or the data for the variable resides. This binding can be performed in two ways: statically or dynamically”). Therefore, POSAs would have implemented Callender’s drivers as dynamic objects, and reasonably expected success doing so.

109. Callender discloses “kernel mode” implementations of the above-discussed sending and receiving “operations.” Callender, 2:40-52; *see also* my discussion of Callender in Section V.A above. POSAs understood, or would have found it obvious, that operations implemented in “kernel mode” were part of the *operating system*, because POSAs understood that “kernel mode” operations were customarily implemented in the *OS*. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, Draves (EX1017), 1:38-47 (“The

other level of execution is referred to as privileged mode, system mode, or kernel mode. Critical operating system components are implemented in kernel mode-- kernel-mode components are responsible for things like virtual memory management, responding to interrupts and exceptions, scheduling execution threads, synchronizing the activities of multiple processors, and other critical or sensitive functions. Such components, which execute from system mode, are generally referred to collectively as ‘the kernel.’”).

110. Thus, Callender’s “kernel mode implementation[s]” (2:40-45) of the “operations for sending and receiving information from adapter 150” (4:5-7) are *OS critical system elements (OSCSEs)*.

111. Furthermore, as explain in paragraphs 112-115 below, POSAs understood that the *OS kernel has these OSCSEs*.

112. As I noted previously (*e.g.* in paragraph 110), Callender’s OSCSEs are part of Callender’s “kernel mode implementation.” Callender, 4:4. As I explained in Section VII.B.1.c.i above, POSAs understood that Callender’s “kernel mode implementation” is part of Callender’s “kernel,” *i.e.*, the *[OS] kernel*.

113. Callender discusses “kernel-bypass IO features offered by the user mode library 230A,” and Figure 2A (annotated below) shows an arrow going from “User-Mode Library 230A” directly to “HCA Adapter” 280A, bypassing the components in “Kernel Mode.” Callender, 5:15-18.

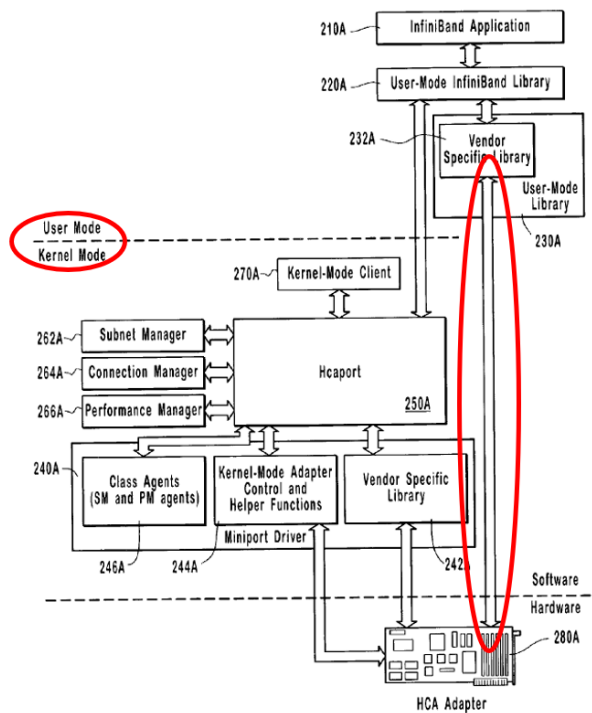


Fig. 2A

114. Furthermore, referencing Figure 2B, Callender states that “[k]ernel specific routines are implemented in kernel mode adapter control and helper functions 244B.” Callender, 5:43-54.

115. Thus, POSAs understood that in Callender, “kernel mode” operations or functions reside in the “kernel,” which is “bypassed” by certain “IO features” in the user-mode library.

116. To the extent Callender is not considered to disclose that the “kernel mode” “operations for sending and receiving” (4:4-7) are in the OS kernel, this would have been obvious. POSAs understood that it was customary to implement critical functions, including communicating with hardware, in the kernel. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*:

Microsoft (EX1035), 300 (“kernel” is “[t]he core of an operating system—the portion of the system that,” among other things, “manages memory, files, and peripheral devices”); Draves (EX1017), 1:38-47 (“Critical operating system components are implemented in kernel mode ... Such components, which execute from system mode, are generally referred to collectively as ‘the kernel.’”); Webster’s (EX1025), 205 (“kernel” is “[i]n an operating system, the core portions of the program that reside in memory and perform most essential operating system tasks, such as handling disk input and output operations and managing the internal memory”); Barron’s (EX1026), 275 (“kernel” is “the central part of an OPERATING SYSTEM. In many operating systems, only the kernel can access hardware directly.”).

117. Furthermore, Callender explains that the kernel-mode “operations for sending and receiving information” are part of a “driver.” Callender, 1:11-18 (“The present invention relates to the field of *driver* software for computer hardware. Specifically, the present invention relates to...a single implementation of one or more *operations* that are *common to both kernel mode processing and user mode processing* relative to the hardware adapter.”), 3:10-15 (“FIG. 1 shows a high-level block diagram of a *hardware driver model* that allows for a single implementation of *common user mode and kernel mode operations* in accordance with the present invention.”).

118. POSAs understood that it was customary for drivers to be dynamically integrated into an OS kernel. My testimony regarding a POSA's background knowledge is corroborated by, e.g.: Oliver (EX1027), 8:64-9:5 ("Device drivers 502 may reside in a kernel space of the OS...[t]he OS may be a Windows operating system[.]"); Pinto (EX1028), [0031] ("Such a transport engine 400 may be hardware which resides in a host memory 430 separately from the host channel adapter (HCA) 120, or alternatively, may be software provided as part of kernel-level device drivers (not shown) of a host operating system (OS)."); Cobbett (EX1029), 5:45-59 ("The device driver resides within the kernel 20 and is the only piece of code able to communicate directly with the hardware. It interfaces with the kernel via a kernel to device driver interface 40."); Nilsen (EX1070), [0065] ("The current art of client software, including both the Mobile IP part and the VPN part, has impact on how a combined architecture can be realized....These principles apply for today's state-of-art terminals like laptops and PDAs, running a state-of-art operating system like Microsoft Windows....These terminals and operating systems make a distinction between user space 119 and kernel space 120. User applications 121 run in user space. This is in contrast to key network components like a TCP/IP stack 118, *drivers* for LAN/WLAN adapters 110 or *drivers* for PPP dial-connection 111 that come as a native part of the operating system in kernel space.").

119. POSAs would reasonably have expected success with such an implementation because it was a customary kernel implementation, as I mentioned in the previous paragraph.

120. Thus, Callender's OS has an OS kernel *having OSCSEs*.

iii. [1B.3] “[OSCSEs] for running in kernel mode using said processor”

121. Callender's “kernel mode” versions of the “operations for sending and receiving” (4:4-7) are *OSCSEs*, as I explained in Section VII.B.1.c.ii above.

122. Callender says the “operations for sending and receiving may be common to both user mode processing and *kernel mode processing*.” Callender, 2:47-50.

123. Callender also discloses that “processes” “*run[]*” in either user or kernel mode. Callender, 1:33-39 (“application processes run within user mode...Processes *running in kernel mode* are privileged” and are “without the restrictions that apply to user mode processes.”).

124. Thus, POSAs understood, or at minimum would have found obvious, that Callender's kernel-mode operations for sending and receiving are used by “processes” that “run” in “kernel mode,” *i.e.*, these operations are *for running in* Callender's “kernel mode.”

125. POSAs also understood that Callender's “kernel mode” is *run using* Callender's “processing unit” (8:22), *i.e.*, the *processor* (as I discussed in Section

VII.B.1.b above for Element [1A]), since POSAs understood that a computer's processor runs kernel-mode processes. *See, e.g.*, Callender, 1:44-48 (“many microprocessors have processing modes to support the distinctions between user mode processes and kernel mode processes.”).

126. The '058 patent's specification defines “kernel mode” as “[t]he context in which the kernel portion of an operating system executes.” EX1001, 6:32-33. I understand that the parties' agreed litigation construction adopts this definition and adds the statement from the specification that “[a]pplicaton code cannot run in kernel mode.” EX1001, 6:34-36; EX1058, 6.

127. Callender describes “kernel mode” as a “processing *context*” that the “vendor specific library” (or “PMI library”) runs in. *See* Callender, 6:45-56.

Callender says:

The process mode independent library (e.g., vendor specific library 232C and vendor specific library 242B) are statically linked into both miniport 240C and user mode library 230C.... Generally, *the PMI library's functions have one or more context specific extension areas passed to them*. The extensions are PMI work areas, used by the PMI function to establish the processing context. *Based on the processing context*, the corresponding user mode or *kernel mode* implementations are called or mapped for a commonly named function.

Callender, 6:45-56. POSAs reading this would have understood that since the “processing context” determines whether the “kernel mode” or “user mode”

implementation of a function in the PMI library is used, then the “kernel mode” and “user mode” are both examples of different processing contexts that the PMI library runs in.

128. Furthermore, POSAs understood that the “kernel” portion of Callender’s OS executes in Callender’s “kernel mode.” Callender, 1:40-42 (“Because the operating system kernel acts as a gatekeeper to computer resources, direct access to resources is generally limited to kernel mode processes.”). Callender also notes that “microprocessors” have “processing modes to support the distinctions between user mode processes and kernel mode processes.” Callender, 1:44-47. Callender further notes that “application processes run within user mode.” Callender, 1:32-34. Thus, POSAs understood Callender’s applications (which I discussed in Section VII.B.1.a above) run in user mode, not kernel mode.

129. Alternatively, implementing Callender such that applications did not run in kernel mode would have been obvious. POSAs understood that it was conventional for applications not to run in kernel mode, as the ’058 patent itself admits. *See* Callender, 1:32-35 (explaining in “Background” section that “application processes run within user mode so that the processes are isolated and cannot interfere with each other’s resources”); EX1001, 6:33-36 (“In *conventional* systems, there is a physical separation enforced by hardware between user mode and kernel mode. *Application code cannot run in kernel mode.*”).

130. Thus, Callender’s “kernel mode” meets the parties’ litigation construction of the term *kernel mode*, which is “[t]he context in which the kernel portion of an operating system executes. Application code cannot run in kernel mode.” EX1058, 6.

131. Therefore, Callender’s OSCSEs are *for running in kernel mode using said processor* as claimed.

d. [1C]

i. [1C.1] “c) a shared library having shared library critical system elements (SLCSEs) stored therein...”

132. The ’058 patent states that a dynamic linked library (DLL) (also referred to in the art as a dynamic link library, *see* Callender, 5:13-15) is an example of a shared library. *See* EX1001, 2:45-50 (“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.”). This is consistent with a POSA’s background knowledge that a DLL is a shared library. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Draves (EX1017), 3:53-61 (in “BACKGROUND” section, stating: “There is, however, an important shortcoming of this [prior-art] virtual memory scheme: it

does not allow sharing of memory-resident and position-dependent code between user-mode and kernel-mode threads. To illustrate this, FIG. 5 shows a virtual memory space 23 in the context of a user process A. A potentially is [*sic*] ***shareable program module such as a DLL (dynamic link library)*** is loaded in a range of virtual memory addresses within the address space of the user process.”); Nelson (EX1097), 18:31-46 (“As is well known, DLL is a library (for example, the Winsock and WSPI libraries 1546, 1560 shown in FIG.3) that is dynamically linked to application programs when they are loaded or run rather than as the final phase of compilation. This means that the same block of library code can be shared between several tasks rather than each task having to containing copies of the routines it uses. At run time, ether the system loader or the tasks entry code arranges for library calls to be patched with the addresses of the real shared library routines.”)³; Petzold (EX1015), 959 (DLLs are “separate files containing functions that can be called by programs and other DLLs to perform certain jobs”), 960 (“one purpose of dynamic link libraries is to provide functions and resources that can be used by many different programs”), 965-966 (“Multiple applications can

³ Although Nelson was filed in December of 2003, the language I have cited from Nelson regarding DLLs describes what was “well known” regarding DLLs prior to Nelson’s filing date. Nelson (EX1097), 18:31-32.

use the same DLL simultaneously, but under Windows 95 these applications are shielded from interfering with each other.”).

133. Callender’s “example hardware driver model” includes a “user mode library” “implemented as a dynamic link library (‘DLL’).” Callender, 4:51-5:18 (“FIGS. 2A-2C illustrate an example hardware driver model that allows for a single implementation of common user mode and kernel mode operations in accordance with the present invention. InfiniBand application 210A accesses host channel adapter (‘HCA adapter’) 280A through user mode library 230A... The user mode library 230A exports a well know application program interface (‘API’) (in the form of a dispatch table) for the routines it supports. In one embodiment, user mode library 230A is implemented as a dynamic link library (‘DLL’).”), Fig. 2A (below).

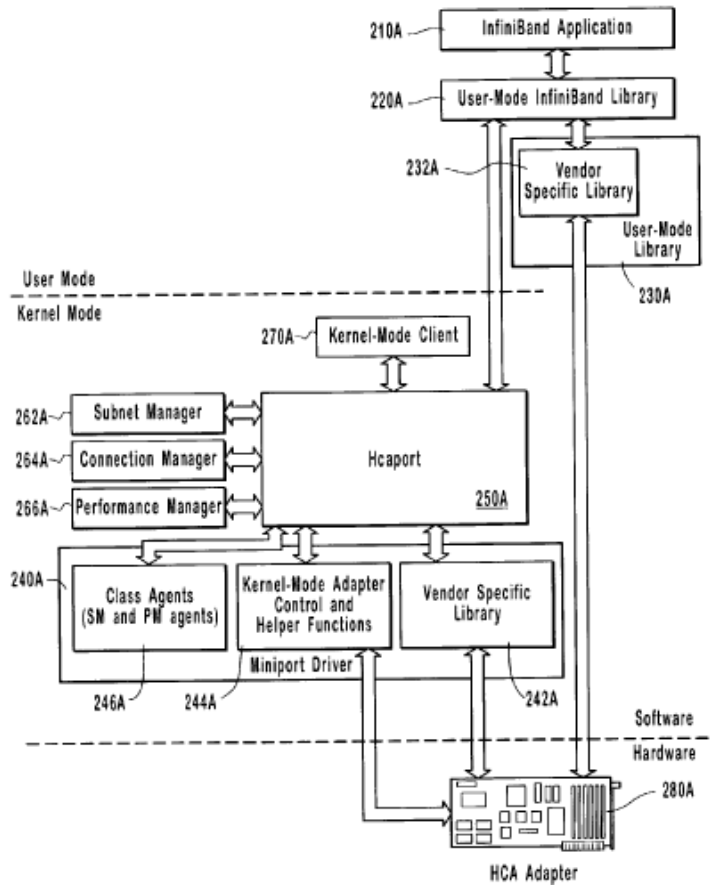


Fig. 2A

134. Thus, Callender’s “user mode library” is a *shared library*.

135. Furthermore, the ’058 patent’s specification defines a “*shared library*” as

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.

EX1001, 6:49-53. “Code space” refers to physical memory space. EX1001, 3:39-43 (“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.”).

136. As I explain in paragraphs 137-138 below, POSAs understood that a DLL satisfies the '058 patent's specification's definition of a “shared library,” which I quoted in the previous paragraph, and which Google's litigation construction also adopts. *See* EX1059, 16.

137. POSAs understood that “[a] typical approach to managing computer system memory is to allocate some portion of the *memory* to the operating system or kernel and another portion for application or user processes.” Draves (EX1017), 2:28-31. Draves describes how a prior-art “DLL (dynamic link library) is loaded in a range of virtual memory addresses within the address space of [a] user process.” Draves, 3:52-60, Fig. 5. As shown in Figure 5, in the prior art, a “virtual *code portion 25 of the DLL is mapped to* a specific portion 28 of *physical memory*.” Draves, 3:66-4:1. Additionally, the “[k]ernel 24 is also mapped to portions 30 of physical memory”; as Figure 5 shows, these portions are *different* from the physical memory portions where the DLL is found. Draves also explains that in the prior art, if “another user process needs to use the same DLL that the first process has already loaded,” “[t]his can be accomplished as shown in FIG. 5,” which shows that the “second process” has its own “virtual address space,” and which shows that “[t]o implement sharing, the virtual memory portion at which

code portion 32 has been positioned maps to the same physical memory 28 that is being used by the code portion 25 of application program A.” Draves, 4:16-28. However, because “[u]ser process B needs its own copy of data...the virtual memory portion at which data portion 33 [the data portion for user process B] has been positioned maps to a different place 34 in physical memory 27 than that used by data portion 26 of application program A.” Thus, Draves illustrates a POSA’s understanding that in the prior art, a DLL, which is shareable, is located in a physical memory space, i.e., code space, that is separate from that occupied by the kernel and that is mapped to a “code portion” in the “virtual address space” of two different applications.

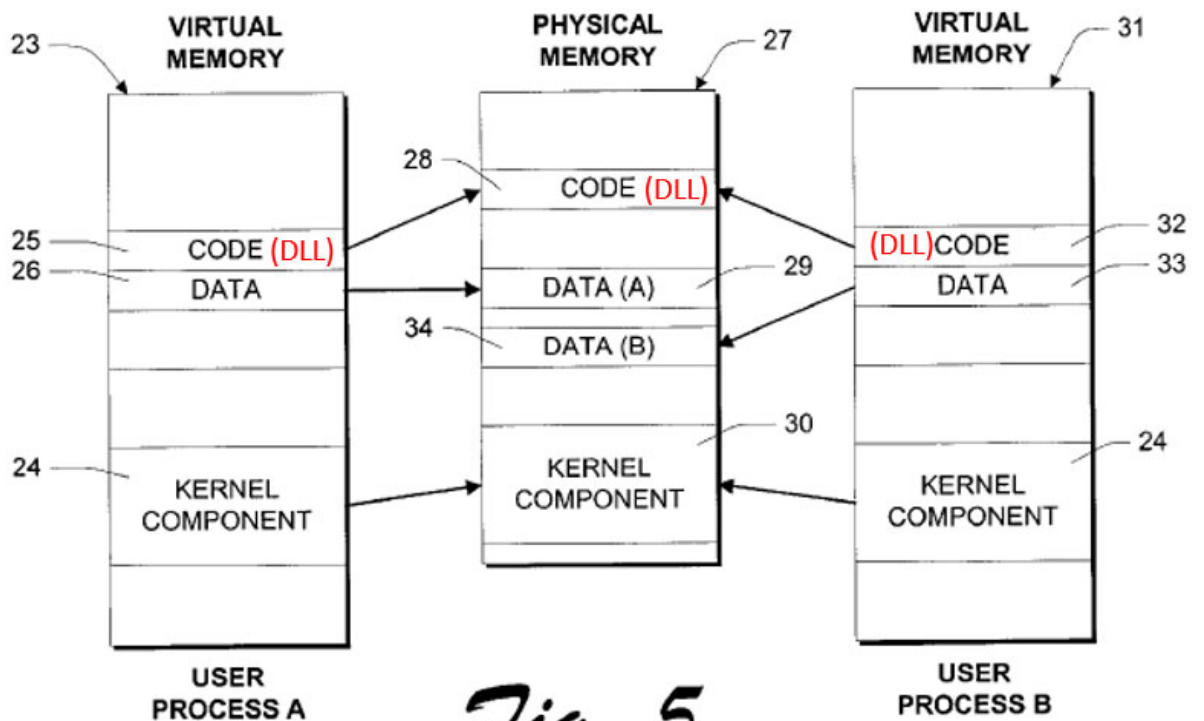


Fig. 5
Prior Art

Draves, Fig. 5

138. Nelson (EX1097) also corroborates my testimony that a POSA would have understood that a DLL satisfies the '058 patent's specification's definition of "shared library." Nelson (EX1097), 18:31-46 ("As is well known, DLL is a library ... that is dynamically linked to application programs when they are loaded or run rather than as the final phase of compilation. This means that *the same block of library code can be shared* between several tasks rather than each task having to containing copies of the routines it uses. At run time, ether the system loader or the tasks entry code *arranges for library calls to be patched with the*

addresses of the real shared library routines.”). My testimony is also corroborated by Petzold (EX1015). Petzold, 959 (Microsoft Press 1996) (DLLs are “separate files containing functions that can be called by programs and other DLLs to perform certain jobs”), 960 (“one purpose of [DLLs] is to provide functions and resources that can be used by many different programs”), 965-966 (“Everything the DLL does is done on behalf of the application. For example, all memory it allocates is owned by the application. Any windows it creates are owned by the application. Any files it opens are owned by the application...

Multiple applications can use the same DLL simultaneously, but under Windows 95 these applications are shielded from interfering with each other.... *The data maintained by a DLL, however, is different for each process. Each process has its own address space for any data the DLL uses.”*

139. Callender’s “user mode library” also meets VirtaMove’s litigation construction, which requires “an application library whose code space is shared among all user mode applications” (EX1068, 11), for the reasons I explain in paragraphs 140-141 below.

140. As I noted in paragraph 137 above (discussing, *e.g.*, Draves, 2:28-31, 3:52-4:32), POSAs understood that a DLL is loaded into a shared physical memory space. POSAs understood that this shared physical memory space into which a DLL is loaded is the *code space* of that DLL. My testimony is corroborated by the

'058 patent itself, which refers to a “physical memory space” as a “code space.”

See EX1001, 3:39-43 (“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.”).

Furthermore, I note that in its claim construction briefing, VirtaMove argued that “‘code space’ is a space occupied by code.” EX1067, 19.

141. Second, POSAs understood that this code space is shared among all user mode applications in the same manner described in the '058 patent, which is also the well-known and conventional way of sharing code between applications. The '058 patent explains that the “manner” in which “the code” for an SLCSE “is shared by all applications on the same compute platform” is that “all applications may physically execute the same set of instructions” representing an SLCSE that reside in “the same physical memory space, that is, shared code space.” EX1001, 3:30-44. The '058 patent describes this technique of sharing code as “common practice” that “all applications” use. EX1001, 3:30-35. Likewise, as I discussed in paragraphs 137-138 above (discussing Draves, 2:28-31, 3:52-4:32, Fig. 5; Nelson, 18:31-46; Petzold, 959-966), POSAs understood that the code representing a DLL is stored in a location in memory that all user-mode applications may access in order to execute the code.

142. As I discussed in Section V.A above, Callender discloses both user mode and kernel mode implementations of “operations” for “sending” and “receiving” information to/from the HCA adapter. These “operations” are *CSEs* for the reasons I explained in Section VII.B.1.c.ii for Element [1B.2].

143. The user mode “operations” for sending and receiving are also *CSEs* for an additional, independent reason. As I noted in Section VII.A above, the “sending” and “receiving” “operations” in Callender’s “user mode library” utilize the “kernel-bypass IO [input/output] features.” Callender, 5:15-18, Fig. 2A. The ’058 patent identifies a “kernel bypass” “network optimization[.]” feature as an example of a CSE. EX1001, 6:11-26.

144. The user mode implementations of the “sending” and “receiving” “operations,” which are *CSEs* as discussed above, are implemented as functions contained in Callender’s “user mode library,” as I discussed in Section VII.A above. POSAs understood that a library “*store[s]*” its contents. Microsoft (EX1035), 309 (“library” is “[i]n programming, a collection of routines *stored* in a file”); *see also* Jennings (EX1065), 1:46-49 (“Libraries allow a computer programmer to keep certain types of procedures and functions that are commonly used by numerous applications in a single library code file that all applications can use. This eliminates the need to duplicate code for common functions in each application.”). Because the functions implementing the sending and receiving

operations are in a shared library, *i.e.*, the “user mode library,” these functions are *SLCSEs*. EX1001, 2:45-47 (“In accordance with this invention, *SLCSEs* are placed in shared libraries[.]”).

145. Thus, Callender’s “user mode library” is a *shared library having SLCSEs stored therein*.

ii. [1C.2] “for use by the plurality of software applications in user mode...”

146. Callender’s “applications” (also called “application programs”) that interact with the HCA adapter are *the plurality of software applications*, as I explained in Section VII.B.1.a for Element [1Pre]. POSAs understood that Callender’s “user mode library” is *for use by* these *applications*, for two independent reasons, as I explain below.

147. First, as I noted in Section VII.B.1.d.i above, Callender’s “user mode library” is a DLL. POSAs understood that DLLs are shared by multiple applications. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Nelson (EX1097), 18:31-46 (“As is well known, DLL is a library...that is dynamically linked to application programs.... This means that the same block of library code can be shared between several tasks.”); Petzold (EX1015), 959-960 (“...one purpose of [DLLs] is to provide functions and resources that can be used by many different programs...”), 965-966 (“Everything the DLL does is done on behalf of the application. For example, all memory it

allocates is owned by the application. Any windows it creates are owned by the application. Any files it opens are owned by the application... **Multiple applications can use the same DLL simultaneously**, but under Windows 95 these applications are shielded from interfering with each other... The data maintained by a DLL, however, is different for each process. Each process has its own address space for any data the DLL uses.”).

148. Second, POSAs understood that Callender discloses multiple “applications” using the user mode library. Callender’s “user mode library” provides “access[]” to the “HCA Adapter.” Callender, 4:54-63. Callender’s “Background” section notes that “some hardware adapters support enforcement of security measures...so that user mode **applications** may access the hardware directly.” Callender, 1:53-61. Additionally, Callender discloses that the user mode library enables user-mode access to the HCA adapter by bypassing the kernel. *See* Callender, 5:15-18 (referring to “kernel-bypass IO features offered by the user mode library”), FIG. 2A (annotated below, showing arrow going directly from “Vendor Specific Library” in “User-Mode Library” to “HCA Adapter”). Thus, POSAs understood that the HCA adapter is a “hardware adapter” that “**applications**” (plural) “may access...directly” using the “user mode library.” Callender, 1:53-61. *See also* Callender, 4:54-63 (“InfiniBand application 210A accesses host channel adapter (‘HCA adapter’) 280A through user mode library

230A and through kernel mode miniport driver 240A.”). This is consistent with a POSA’s background knowledge that multiple applications on a single computer can access an InfiniBand network via an HCA. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, Ganapathy (EX1062), 1:11-39 (noting in “BACKGROUND” that “Infini[B]and...allows *applications* running on *a* computer...to send messages...by directly accessing the hardware without going through the operating system layer.”).

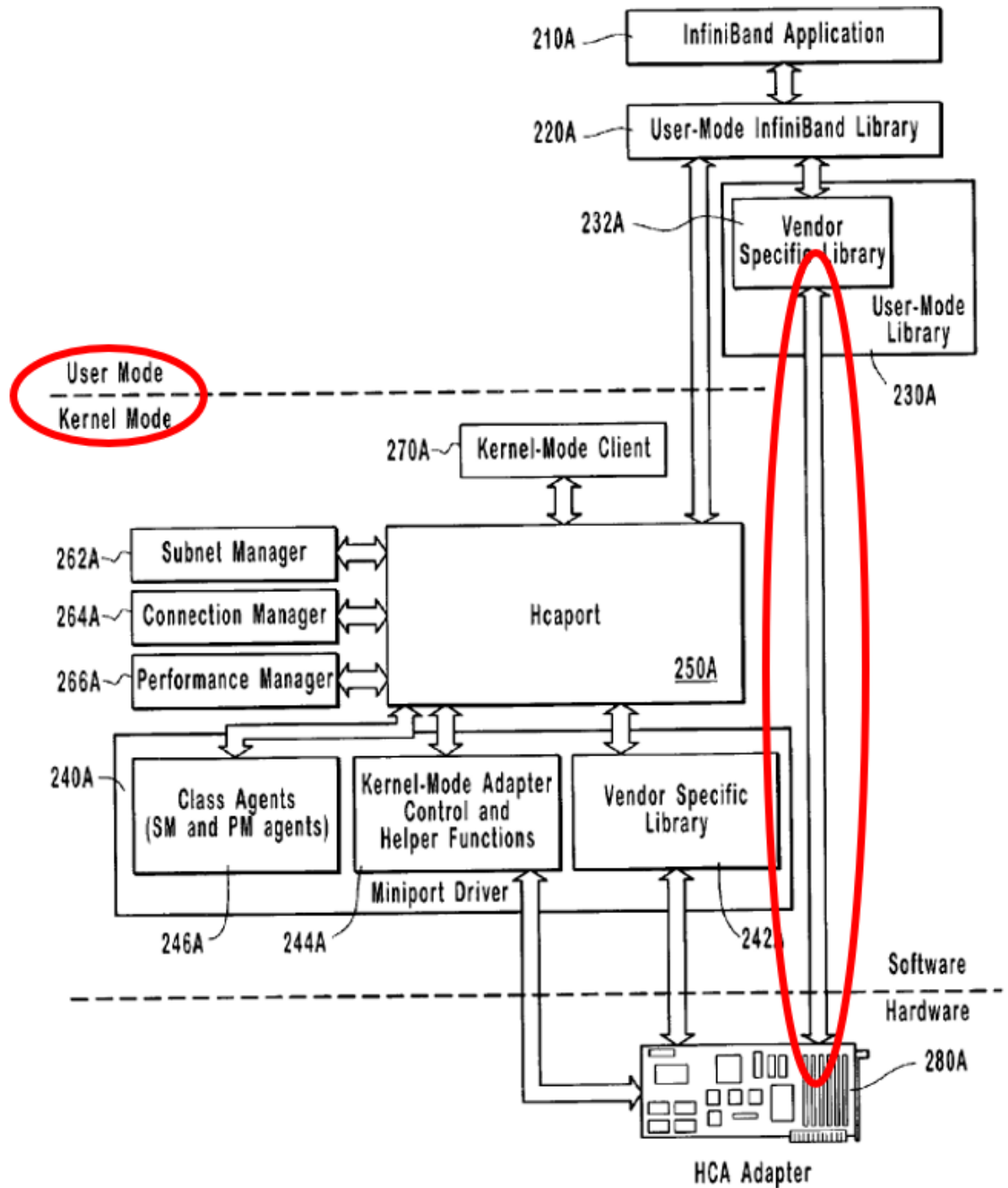


Fig. 2A

149. The functions in the “user mode library” that implement the “sending” and “receiving” “operations” are the claimed *SLCSEs*, as I explained in Section

VII.B.1.d.i above. Because the “user mode library” is *for use by the plurality of software applications* as I discussed above, POSAs understood that the sending and receiving functions in the library are likewise *for use by the plurality of software applications* that interact with the HCA adapter, because sending and receiving data are operations that multiple applications that interact with the HCA adapter perform.

150. Callender’s Figures 2A and 2C show an arrow going between the “InfiniBand Application” and the “User-Mode InfiniBand Library,” and an arrow going between the “User-Mode InfiniBand Library” and the “User-Mode Library.” POSAs understood that this visually depicts an example “InfiniBand application 210A access[ing] host channel adapter (‘HCA adapter’) 280A through user mode library 230A” (Callender, 4:54-57). Thus, the “sending” and “receiving” functions in the “user-mode library” are for use by the plurality of applications.

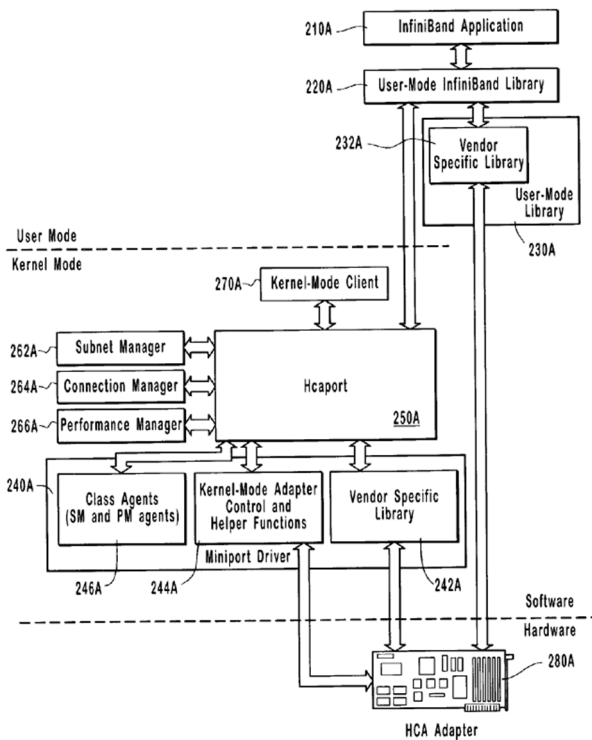


Fig. 2A

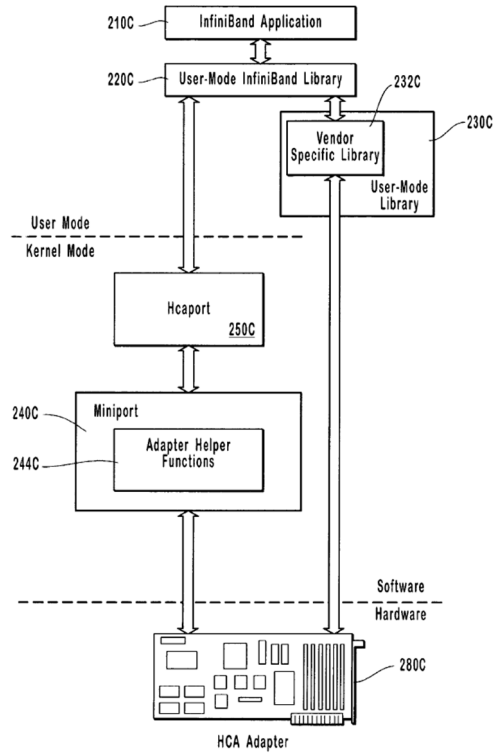


Fig. 2C

151. In the example depicted in Figures 2A and 2C that I discussed above, POSAs understood that the application uses an SLCSE in the “user-mode library” by calling through the “User-Mode InfiniBand Library” to access the user-mode library and the SLCSE. This understanding is further supported by Callender’s description of the “user-mode library” as an “extension” of the “user-mode InfiniBand library.” Callender, 5:9-11. Moreover, it was well-known for applications to use functionality in a library via another library. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Fontarensky (EX1030), [0034] (“Message system 20 can include a dynamic link library (‘DLL’). A DLL is a module including executable functions and/or data. The

DLL can be called by a software application or another DLL to perform a particular function or access data.”); Kemp (EX1031), 1:53-60 (“Each .dll file may itself contain calls to other.dll files. For example, a user interface .dll file may provide a user interface comprised of several tabbed property sheets for the user to choose from depending on the type of device functionality that the user wishes to access and/or modify. In such a case, the user interface .dll file may contain fixed calls to other predetermined.dll files representing each of the tabbed property sheets of the user interface.”); Chan (EX1061), 3:27-35 (“Each programming team develops their respective library 108-112, and exposes entry points to application programs 102–106, so application programmers can use the respective functionality. This delegation of human resources, based on functionality, allows application programmers to rely on services provided by experts in respective library groupings 108-112. Further, library developers are able to rely on functionality provided by other libraries 120–124.”), 22:27-29 (“Many DLLs have spaghetti dependencies requiring many other DLLs to be loaded even when only small portions are utilized.”). The ’058 patent too describes an embodiment where user-mode CSE functionality is implemented via different libraries that work together. EX1001, 8:4-13, 8:16-26, 9:43-46, Figs. 3-4

152. As I discussed above, the SLCSEs are in Callender’s “*user mode* library” and are used by applications that “run within *user mode*.” Callender,

4:54-57 (“InfiniBand application 210A accesses host channel adapter (‘HCA adapter’) 280A through user mode library 230A and through kernel mode miniport driver 240A.”), 1:33-35 (“Generally, application processes run within user mode so that the processes are isolated and cannot interfere with each other’s resources.”).

153. The ’058 patent defines “user mode” as “[t]he context in which applications execute.” EX1001, 6:31. I understand that in litigation, the parties’ agreed construction of *user mode* adopts this definition. EX1058, 6. As I explain below, Callender’s “user mode” meets this definition, for two reasons.

154. First, Callender describes “user mode” as a “processing *context*” that the “vendor specific library” (or “PMI library”) runs in:

The process mode independent library (e.g., vendor specific library 232C and vendor specific library 242B) are statically linked into both miniport 240C and user mode library 230C. As noted above, by making the library a common component, the development, maintenance, and testing of an HCA miniport may be reduced. Generally, **the PMI library’s functions have one or more context specific extension areas passed to them.** The **extensions** are PMI work areas, used by the PMI function to **establish the processing context.** Based on the **processing context**, the corresponding **user mode or kernel mode implementations are called** or mapped for a commonly named function.

Callender, 6:45-56.

155. Second, POSAs understood that Callender’s applications in “user mode.” Callender, 1:33-35 (“Generally, application processes run within user mode...”).

156. Therefore, POSAs understood that the SLCSEs are *for use by the plurality of software applications in user mode.*

e. [1D]

i. [1D.1] “i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and...”

157. The SLCSEs are *stored in the shared library*, i.e., the “user mode library,” as I discussed in Section VII.B.1.d.i above for [1C.1].

158. In my opinion, the outer boundaries of the scope of the claim term “*functional replicas*” is unclear to POSAs. However, the ’058 patent says, “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 [OS].” EX1001, 1:65-2:1. Additionally, the patent says, “[t]he 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.” EX1001, 8:27-32. Thus, in my opinion, at a minimum, a *functional replica* of an OSCSE encompasses a copy of an OSCSE.

159. In Callender, the *OSCSEs* are the kernel-mode sending and receiving operations (as I discussed in Section VII.B.1.c.ii for [1B.2]), while the *SLCSEs* are the functions in the user-mode library implementing the sending and receiving operations (as I discussed in Section VII.B.1.d.i for [1C.1]). Callender says, “*the same source code* may be written for a kernel mode communication operation and a user mode communication operation.” Callender, 4:30-32. Furthermore, Callender says this source code can be in a “static library derived from a common source.” Callender, 4:33-34. POSAs understood that if the source code for a given operation is in a static library, then the executable code for the operation is copied into both the user-mode and kernel-mode implementations of the operation. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, EX1001, 6:54-55 (saying that “static library” is “[a]n application library whose code space is contained in a single application.”); Petzold (EX1015), 959 (“*Static linking* occurs during program development when you link various object (.OBJ) modules, run-time library (.LIB) files, and usually a compiled resource (.RES) file to create a Windows .EXE file”), 961 (“An object library is a file with the extension .LIB containing code that is added to your program's .EXE file when you run the linker during static linking.”).

160. Thus, Callender discloses that each *SLCSE* includes a copy of the executable code for the operation that is the corresponding *OSCSE*, and therefore is a *functional replica* of the corresponding *OSCSE*.

161. Furthermore, even if the *SLCSEs* were not copies of the corresponding *OSCSEs* (for example, in the circumstance I discuss in Section VII.B.18 below for claim 18), the relationship between Callender's user-mode InfiniBand sending/receiving operations and the corresponding kernel-mode operations parallels the example of "replication" of *OSCSEs* by *SLCSEs* provided in the '058 patent. *See* EX1001, 6:22-55. Specifically, the specification says "[e]mbodiments of the invention enable the replication of critical system elements normally found in an operating system kernel," and describes an "example" in which the "CSE that is part of a shared library" is a "TCP/IP stack," and where the "TCP/IP services in the CSE are the same as those provided in the Linux [operating system] kernel." EX1001, 6:22-55. POSAs understood that InfiniBand, like TCP/IP, is a networking protocol. *See, e.g.,* Kagan (EX1013), [0007], which I discuss in Section VII.A above). Additionally, the user-mode InfiniBand sending/receiving operations provide "services" to the user-mode applications that are the "same as those provided in the [operating system] kernel" (EX1001, 6:22-55) by the kernel-mode operations; indeed, Callender discloses that both the user-mode and kernel-mode InfiniBand sending/receiving operations are invoked via a

“common interface.” Callender, Abstract, 2:32-39, 3:29-34, 4:23-40, 7:5-26.

Thus, although the full scope of “similar attributes” in the ’058 patent’s definition of “replica” is not well-defined to POSAs, POSAs would have understood that Callender’s user-mode functions for performing the InfiniBand sending/receiving operations, which are the claimed SLCSEs (as I discussed in Section VII.B.1.d.i for [1C.1]), have the same relevant attributes (i.e., “characteristic or quality,” EX1057, 41-42), namely the services provided, as their kernel-mode counterpart operations, which are the claimed OSCSEs (as I discussed in Section VII.B.1.c.ii for [1B.2]), in the same manner described in the ’058 patent, and are thus *functional replicas of OSCSEs* even though the outer boundaries of this claim language are not well-defined to POSAs. Furthermore, unlike in the prior art that the ’058 patent distinguishes, the services provided by the SCLSEs are *not* provided by a separate process with which an application needs to communicate by going through the kernel (*see* my discussion of the ’058 patent’s specification distinguishing of this technique in Section V.A); rather, the “services” are provided by SLCSEs that form a part of the application, as I discuss in Section VII.B.1.e.iii below for [1D.3].

162. The user-mode sending/receiving functions are also *functional replicas of OSCSEs* under VirtaMove’s litigation construction, “substantial functional equivalents or replacements of kernel functions.” EX1067, 22. As I

noted above, Callender discloses that the user-mode functions include copies of the executable code for the corresponding kernel-mode operations. I am informed and understand that VirtaMove has argued in litigation “‘the term replica’ specifically in the context of *functional* replicas” includes “copies of OSCSEs.” EX1067, 23 (emphasis original). Furthermore, in Callender, Callender SLCSE replicates the function of the corresponding OSCSE: the user-mode sending function performs the same “operation” (sending data via the HCA adapter) as the kernel-mode counterpart operation, and the user-mode receiving function likewise performs the same “operation” (receiving data via the HCA adapter) as the kernel-mode counterpart operation. Additionally, the user-mode sending and receiving functions replace the corresponding kernel-mode operations for applications using the user-mode library, as I explained in Section VII.B.1.d.ii for [1C.2].

163. During prosecution, the ’058 patent’s applicants distinguished O’Rourke (EX1009) by arguing that a “functional replica” cannot be a mere “intermediary” to the kernel. EX1002, 323 (“Indeed, the user mode proxy filter is not a functional replica, but merely acts as an intermediary to the kernel mode filters.”).

164. POSAs understood that Callender’s user-mode sending and receiving functions (the *SLCSEs*) are not mere intermediaries. Callender’s user-mode sending and receiving functions implement complete operations by executing their

own source code, which may be the same as the source code of their kernel-mode counterpart operations. Callender, 3:53-55 (“Having *corresponding operations implemented in both* user mode implementation 130 and kernel mode implementation 140 is desirable”), 4:29-36 (“the same source code may be written” for both kernel- and user-mode operations), 7:11-26 (describing benefits of Callender’s “single implementation” of common operations), claim 1 (reciting “providing a user mode implementation of the at least one operation”). Moreover, the user-mode functions perform operations that “bypass” the kernel, and as such are not intermediaries to the kernel, as I discussed in Section VII.A. Callender, 5:15-18. In contrast, in my opinion, POSAs understood that O’Rourke’s user-mode “proxies,” which the ’058 patent’s applicants distinguished, invoke kernel-level code. EX1002, 323 (applicants citing O’Rourke, 6:12-19 and 10:12-33); O’Rourke (EX1009), 6:12-19 (“primary object” is “user mode proxies for kernel mode filters”), 10:12-33 (describing “prox[ies]” as invoking “kernel mode” components that perform operations).

165. Thus, Callender’s SLCSEs are *functional replicas of the OSCSEs*.

ii. [1D.2] “are accessible to some of the plurality of software applications and...”

166. The *plurality of software applications* that access Callender’s HCA adapter utilize the “sending” and “receiving” functions in the “user mode library,” as I discussed in Section VII.B.1.d.ii for [1C.2]. These functions are *SLCSEs*, as I

discussed in Section VII.B.1.d.i for [1C.1]. In my opinion, POSAs understood that these *SLCSEs* are *accessible to* the applications, for multiple independent reasons, as I explained below.

167. First, as I discussed in Section VII.B.1.d.i for [1C.1], Callender’s “user mode library” containing the SLCSEs is a “DLL.” POSAs understood that a DLL’s purpose is to serve as a “library” accessible to multiple applications. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Petzold (EX1015), 959 (“The term dynamic linking refers to the process that Windows uses to link a function call in one module to the actual function in the library module.”), 960 (“Thus, one purpose of dynamic link libraries is to provide functions and resources that can be used by many different programs.”), 965-966 (“Multiple applications can use the same DLL simultaneously, but under Windows 95 these applications are shielded from interfering with each other.”); Nelson (EX1097), 18:31-46 (“As is well known, DLL is a library ... that is dynamically linked to application programs when they are loaded or run rather than as the final phase of compilation. This means that *the same block of library code can be shared* between several tasks rather than each task having to containing copies of the routines it uses. At run time, ether the system loader or the tasks entry code *arranges for library calls to be patched with the addresses of the real shared library routines.*”); Walsh (EX1032), 1:28-31 (“A dynamic-link library is an

executable module containing services that application programs can call to perform useful tasks. Dynamic-link libraries exist primarily to provide services to application programs.”).

168. As I discussed in Section VII.B.1.d.ii for [1C.2], POSAs understood that in Callender, the plurality of applications (such as the exemplary “InfiniBand application 210A” shown in Figure 2A) use SLCSEs in the “user-mode” library by calling through the “User-Mode InfiniBand Library” to access the user-mode library and the SLCSEs. Thus, because the applications access the SLCSEs by calling through the “User-Mode InfiniBand Library,” the SLCSEs are *accessible to* the applications.

169. POSAs understood that a function is a predefined set of instructions that carry out a specific action. *See* Section VII.A above (discussing, *e.g.*, Hatch (EX1007), 1:22-29). POSAs also understood that executing the instructions that make up a function involves the processor reading those instructions from memory. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: LaBerge (EX1054), 1:14-17 (“A conventional computer system includes a processor that retrieves or **reads program instructions**...and **executes the program instructions** to perform a corresponding function.”); Colling (EX1055), 1:22-27 (“Thus, for a computerized system such as a personal computer or wireless telephone to perform a requested function, the computerized

system must be able to **read and execute operating system (OS) or platform specific executable instructions** that cause the computerized system to perform the function.”); Pashan (EX1056), 4:18-21 (“Functions 61-63 are illustratively implemented as instructions stored in a read-only memory 59 which processor 20 executes when called upon to do so.”).

170. Therefore, POSAs understood that when the applications use the SLCSEs, which are functions as discussed above, the processor that is executing the application reads the executable instructions that make up the SLCSEs; thus, Callender’s SLCSEs meet Google’s litigation construction of “*accessible*,” which requires that “two or more of the plurality of the software applications can read SLCSEs stored in the shared library.” EX1059, 17.

171. Callender’s SLCSEs also meet VirtaMove’s litigation construction, requiring that “some of the plurality” of the applications can “use” SLCSEs. EX1059, 17-18. POSAs understood that by reading and executing the instructions making up the SCLSEs as discussed above, the applications use the SCLSEs. I also note that VirtaMove conceded in its claim construction briefing that the claim term “*access*” encompasses reading. EX1067, 21. Furthermore, because the *plurality* of applications collectively use each SLCSE, at least *some* of the applications use SLCSEs.

172. Therefore, the SLCSEs are *accessible to at least some of the plurality of software applications*.

- iii. [1D.3] “**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**”

173. The '058 patent says SLCSEs “*form a part* of at least some of the software applications **by being linked** thereto.” EX1001, 2:10-15 (“a shared library having critical system elements (SLCSEs) stored therein...wherein some of the CSEs stored in the shared library... *form a part* of at least some of the software applications **by being linked** thereto”); *see also* claim 16 (reciting “SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto”).

174. The parties’ agreed litigation construction of “forms a part of...” adopts the “linked to” requirement. EX1058, 6.

175. Callender’s SLCSEs are in a “dynamic *link* library (‘DLL’).” Callender, 5:13-15; *see also* my discussion in Section VII.B.1.d.i for [1C.1]. POSAs understood that a “DLL is a library...that is dynamically linked to application programs.” Nelson (EX1097), 18:31-34; *see also* Moberg (EX1033), 4:47-49 (“dynamic linking using DLLs provides a mechanism to link applications to libraries at run-time”). POSAs also understood that a DLL exists on a computer in the form of “a file that gets mapped into a process’s address space such that any

functions in the DLL appear to be *part of* the process.” Silberschatz (EX1018), 745; *see also* Petzold (EX1015), 965-966 (“Although I’ve just categorized a DLL as an extension to Windows 95, it is also an extension to your application program. Everything the DLL does is done on behalf of the application. For example, all memory it allocates is owned by the application. Any windows it creates are owned by the application. Any files it opens are owned by the application.”); Draves (EX1017), 3:53-61 (in “BACKGROUND” section, stating: “There is, however, an important shortcoming of this [prior-art] virtual memory scheme: it does not allow sharing of memory-resident and position-dependent code between user-mode and kernel-mode threads. To illustrate this, FIG. 5 shows a virtual memory space 23 in the context of a user process A. A potentially is [*sic*] ***shareable program module such as a DLL (dynamic link library)*** is loaded in a range of virtual memory addresses within the address space of the user process.”), Fig. 5 (annotated below). POSAs therefore understood that when an application *accesses* an SLCSE (*see* Section VII.B.1.e.ii), the SLCSE *forms a part of* that application because the DLL is linked to the application and gets mapped into its address space.

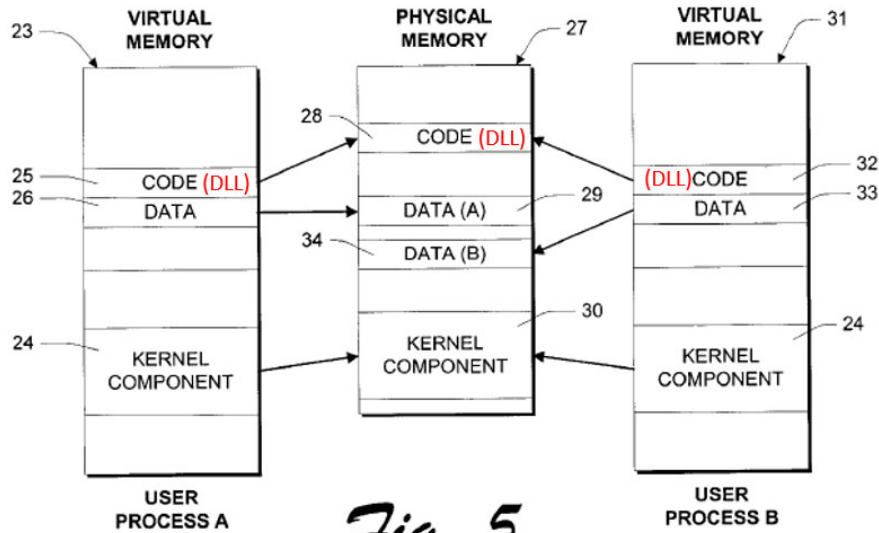


Fig. 5
Prior Art

Draves, Fig. 5

176. Thus, *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.*

f. [1E]

i. [1E.1] “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...”

177. POSAs understood that “*instance*” is not a term of art specific to CSEs; instead, POSAs understood that “*instance*” customarily refers to an object or a copy of an object. My testimony regarding a POSA’s background knowledge is

corroborated by, e.g.: Collin (EX1034), 186 (“instance *noun*” is “an object or duplicate object that has been created”); Microsoft (EX1035), 276 (“instance n.” is “An object... in relation to the class to which it belongs. For example, an object *myList* that belongs to a class *List* is an instance of the class *List*.”).

178. The specification of the '058 patent does not define an *instance* of a SLCSE; however, the specification says the “intent” of having applications “link” to the SLCSE library “is to *provide* an *application* with a unique *instance of a CSE*.” EX1001, 3:20-29.

179. The specification also does not explain what it means for an *instance* of an SLCSE to *run in a context of an application*. However, the specification says the “ability to allow a CSE to execute in the same context as an application...allows...an ability to deploy multiple instances of a CSE.” EX1001, 1:46-54. The specification also says SLCSEs *run in a context* of an application because the SLCSEs reside in a “shared library” that is linked to the application. EX1001, 5:22-26 (“Embodiments of the invention enable the **replication** of critical system elements.... 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**.”), 8:27-33 (“The **CSE library** includes replicas or substantial functional equivalents or replacements of kernel functions.... These functions can be directly called by the applications 42 and **as such can be run in the same**

context as the applications.”), 9:15-20 (“...some system elements that are critical to the operation of a software application are replicated...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....”).

180. As I explain below, Callender discloses applications accessing SLCSEs using the same technique as in the ’058 patent, which POSAs understood would yield the same result.

181. Callender’s functions in the “user mode library” for performing the “sending and receiving” “operations” are claimed *SLCSEs*, as I discussed in Section VII.B.1.d.i for [1C.1]. As I also discussed in that section, the “user mode library” is a “dynamic **link** library,” which is a shared library “**link[ed]**” to applications, just as the ’058 patent’s SLCSE library is “linked” to applications. Callender, 5:13-15; EX1001, 3:20-29. As the ’058 patent itself admits, as I discussed above (*see* EX1001, 5:22-25, 8:27-33, 9:15-20), the well-known consequence of linking a DLL to an application is that *an instance of each function is provided to each application from the shared library and is run in the context of that application.*

182. The ’058 patent does not expressly explain how SLCSE *instances* are provided to applications *without being shared* between applications. However, the patent states that although there is one copy of the computer “instructions” (or

“code”) representing an SLCSE that is “shared by all applications,” each application “uses [its own] separate data area[]” when executing the instructions representing the SLCSE:

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.

EX1001, 3:30-42. “In this manner,” “CSEs are not shared among applications even though the code is shared.” EX1001, 3:42-44.

183. POSAs understood that in Callender, applications access SLCSEs in the same manner described in the '058 patent. SLCSEs are provided to Callender’s applications in a DLL, as I discussed in Section VII.B.1.d.i for [1C.1]. POSAs understood that the different applications share the code representing the “sending” and “receiving” functions but use their own data areas in memory when executing the code, since this was the known way for DLLs to be used. My

testimony regarding a POSAs background knowledge is corroborated by, *e.g.*: Petzold (EX1015) 965-966 (“Each process has its own address space for any data the DLL uses.”); Silberschatz (EX1018), 745 (“A DLL... gets mapped into a process’s address space such that any functions in the DLL appear to be part of the process.”).

184. Alternatively, POSAs would have found it obvious to implement Callender such that applications share the code for functions in the DLL containing the SLCSEs (*i.e.*, the user-mode library) but use application-specific data areas when executing the code. The ’058 patent admits that such an implementation was “common practice,” and POSAs understood that using this technique would beneficially allow multiple applications to invoke the same function; POSAs would also reasonably have expected success in implementing this common practice. EX1001, 3:30-44 (“This sharing of code is common practice. All applications currently share code.... Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application[.]”). *See also* Petzold (EX1015), 965-966 (“Each process has its own address space for any data the DLL uses.”); Silberschatz (EX1018), 745 (“A DLL... gets mapped into a process’s address space such that any functions in the DLL appear to be part of the process.”).

185. Thus, in Callender, *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.*

- ii. **[1E.2] “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, and...”**

186. POSAs understood that the *plurality of software applications run under the operating system*, or at least would have found it obvious to implement Callender this way, because POSAs understood that this was the standard relationship between applications and the OS. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Nickerson (EX1036), 1:13-14 (“A computer system often concurrently runs several applications under an operating system.”); Calusinski (EX1037), [0003] (“Computer systems typically include operating system software that controls the basic function of the computer, and one or more software application programs that run under the control of the operating system to perform desired tasks.”).

187. The user-mode “sending” and “receiving” functions, *i.e.*, the *SLCSEs*, are in the “user mode library,” which is a DLL, as I discussed in Section VII.B.1.d.i for [1C.1]. Because DLLs are shared libraries, POSAs understood that

multiple applications running under the computer's OS can call the same function in a DLL, *i.e.*, that multiple applications *have use of* the same function *for performing same function* in a DLL. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Petzold EX1015, 959-960 (“...one purpose of [DLLs] is to provide functions and resources that can be used by many different programs...”), 965-966; Nelson (EX1097), 18:31-46 (“As is well known, DLL is a library ... that is dynamically linked to application programs when they are loaded or run rather than as the final phase of compilation. This means that ***the same block of library code can be shared*** between several tasks rather than each task having to containing copies of the routines it uses. At run time, ether the system loader or the tasks entry code ***arranges for library calls to be patched with the addresses of the real shared library routines.***”).

188. The '058 patent says an application obtains use of a “*unique instance*” of an SLCSE by “linking” to the “library containing the [SLCSE].” *See* EX1001, 3:25-29. That passage states: “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.” EX1001, 3:25-39. POSAs reading the '058 patent understood that the “CSE” referred to in this passage is an SLCSE, because this passage refers to a CSE in a “library” that each application “link[s]” to, *i.e.* a library that is shared.

189. The '058 patent also states that because “[e]ach application has its own unique data space...[SLCSEs] are unique to an application.” EX1001, 3:36-39. In Callender, each application is linked to the “user mode library” containing the SLCSEs, as I explained in VII.B.1.e.iii for [1D.3]. Furthermore, each application has its own unique data space in memory, as I discussed in VII.B.1.f.i. Thus, in Callender, *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*

- g. [1F] “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.”**

190. The functions for “sending and receiving information” in Callender’s “user mode library 230A” are claimed *SLCSEs*, as I discussed in Section VII.B.1.d.i for [1C.1]. POSAs understood, or would at minimum have found obvious, that functions are invoked by reference to some *predetermined* identifier, *e.g.*, the *function* name, as this was the typical way to invoke functions contained in shared libraries and was within a POSA’s skill. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, Wilner (EX1069), 10:55-58 (“Code module 126a includes executable code that includes instructions that

reference functions called ‘foo()’ and ‘goo(),’ neither of which are part of code module 126a.”). Thus, each SLCSE is, and thus *relates to*, a *predetermined function*. Additionally, these SLCSEs are *provided to the plurality of the software applications*, including the *first* and *second* of the plurality, because they are in a DLL accessed by the applications, as I explained in Section VII.B.1.e.ii for [1C.2].

191. Furthermore, the SLCSEs are provided to the applications *for running instances of* a SLCSE; the SLCSEs are in a DLL, and when an application invokes an SLCSE from the DLL, an *instance* of the SLCSE *runs* in that application’s context, I discussed in Section VII.B.1.f.i for [1E.1].

192. Finally, the SLCSEs are provided to the *first* and *second* of the *plurality of applications* for running the *first* and *second* instances *simultaneously*. I have reviewed the ’058 patent, and “*simultaneous[]*” appears nowhere in the ’058 patent’s specification. Element [1F] was originally filed as dependent claim 6. EX1002, 22. During prosecution, the applicants distinguished as-filed claim 6 over Cabrero by arguing that CSEs are “replicated and **can be accessed *simultaneously*** by...an application in user mode **by way of** the application using an instance of the CSE within a shared library,” which was in “contradistinction” to Cabrero, where CSEs are “independent applications that neither reside in the OS nor in shared libraries.” EX1002, 241-242, 244. The applicants also said “claim 6 *allows* different instances of an SLCSE...to be executed simultaneously by another

one or more applications.” EX1002, 244. Thus, according to the applicants, Element [1F]’s “*for running...simultaneously*” is met by allowing two different applications to have simultaneous access to an SLCSE in a shared library. As I discussed above, Callender’s SLCSEs are operations provided by a DLL, and POSAs understood that one purpose of DLLs was to allow multiple applications to access the same code in the DLL “*simultaneously*.” Petzold (EX1015), 959 (DLLs are “separate files containing functions that can be called by programs and other DLLs to perform certain jobs”), 960 (“one purpose of [DLLs] is to provide functions and resources that can be used by many different programs”), 965-966 (“Multiple applications can use the same DLL *simultaneously*”); *see also* Nelson (EX1097), 18:31-46 (the effect of applications linking to DLLs “means that the same block of library code can be shared between several tasks”).

193. Additionally, it was known to implement computers with multiple processors that could each execute at the same time. *See, e.g.*, Kagan (EX1013), [0006] (referring to known “multi-CPU systems”). POSAs would have been motivated to implement the computer in Callender with multiple processors to achieve the performance benefits of multi-processor systems, and would reasonably have expected success doing so since such systems were well-known. In such an implementation, different *software applications, i.e.*, InfiniBand applications using the user-mode library (*see* Section VII.B.1.d.ii ([1C.2])) can

each run on a different processor and run the instructions making up an SLCSE *simultaneously*. This is an alternative reason why Callender meets Element [1F].

2. Claim 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”

194. In Callender, *first* and *second* applications can *simultaneously* each run an *instance* of an SLCSE, as I discussed in Section VII.B.1.g for [1F]. The SLCSE is *stored in a shared library*, as I discussed in Section VII.B.1.d.i for [1C.1]. The *first* and *second* instances together comprise *multiple instances*.

195. If VirtaMove argues that “*multiple*” requires more than two instances, this would have been obvious. POSAs understood that three or more applications running on a single computer could all use the same InfiniBand hardware. My testimony is corroborated by, *e.g.*: Ganapathy (EX1062), 1:11-39 (noting in “BACKGROUND” that “Infini[B]and...allows *applications* running on a computer...to send messages...by directly accessing the hardware without going through the operating system layer.”). Moreover, as I discussed in Section VII.B.1.a, Callender also discloses plural applications using the HCA adapter in a computer system. POSAs also understood that there were at least three different types of applications for which InfiniBand could be used. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Trammel (EX1063), 1:15-21 (“InfiniBand products are ideally suited for clustering, I/O extension, and

native attachment”); Mann (EX1071), [0004] (“InfiniBand based networks are designed to satisfy bandwidth-hungry network applications, such as those combining voice, data, and video on the Internet.”).

196. POSAs would have been motivated to run at least three different InfiniBand applications on the same computer to achieve the efficiency benefit of using the same hardware for multiple tasks, and would reasonably have expected success given that Callender itself discloses running multiple applications, I as noted above.

197. POSAs understood that when the SLCSE instances are run (whether simultaneously or not), *i.e., in operation*, they run *within the operating system*. The '058 patent does not explain what it means for an SLCSE instance to run “*within*” the OS. However, the patent refers to “**applications** [that] run[] under the [OS]” and that have “use of a unique **instance** of a corresponding critical system element.” EX1001, Abstract, 2:18-21, 2:35-40. Furthermore, POSAs understood that the prior art sometimes referred to typical application execution as applications running “within” an OS. *See, e.g.*, Goossen (EX1038), 1:18-20 (referring to “conventional computer systems having...software applications running within the [OS]”); Green-162 (EX1039), 1:6-15 (“The present invention relates generally to managing processes within an operating system ... When a user runs an

application or program on a computer, the user is actually requesting the operating system to load and run one or more processes associated with that application”).

198. Thus, POSAs understood that claim 2’s reference to “*instances of an SLCSE...run[ning]...within the [OS]*” encompasses applications running within the OS and using their own instances of SLCSEs.

199. In Callender, the first and second applications are provided with their own unique instances of an SLCSE, as I discussed in Sections VII.B.1.f.i-VII.B.1.f.ii for [1E]. POSAs understood that in Callender, the applications *run within* the OS because the application’s processes are “*run within* [the OS’s] user mode.” Callender, 1:29-35 (“Many **operating systems provide** at least two process modes,” namely “user mode” and “kernel mode,” and [g]enerally, **application processes run within user mode**”), claim 1. *See also* Silberschatz (EX1018), 3 (“An **operating system**...provides a basis for application programs and acts as an intermediary between a user of a computer and the computer hardware.”), 5 (“An operating system...manages the execution of user programs[.]”).

200. Thus, in Callender’s computer system, *in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.*

201. Additionally, Callender's *SLCSEs* replicate the functionality of CSEs in the OS (*i.e.*, *OSCSEs*), as I discussed in Section VII.B.1.e.i for [1D.1]. The *SLCSEs* are also in shared libraries from where they can be run simultaneously, as I discussed in Section VII.B.1.g for [1F]. This is in contrast to the '058 patent applicants' characterization of Cabrero when distinguishing claim 2 during prosecution. EX1002, 242 ("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.").

3. Claim 3: "A computing system as defined in claim 1 wherein *OSCSEs* corresponding to and capable of performing the same function as *SLCSEs* remain in the operating system kernel"

202. As I discussed in Section VII.B.1.c.ii for [1B.2], Callender's "operations for sending and receiving information from adapter 150" in the "*kernel* mode implementation" are claimed *OSCSEs*. Callender, 3:40-4:16. As I discussed in Section VII.B.1.d.i for [1C.1], these *OSCSEs* *correspond to* and are *capable of performing the same function as* the *SLCSEs*, which are the functions in the "user mode library" for performing the "operations" for "sending and receiving information." *See also* Callender, 3:53-4:3 (explaining desirability of Callender's techniques with "**corresponding operations** implemented in **both** user mode...and kernel mode").

203. The OSCSEs are part of the OS kernel, as I discussed in section VII.B.1.c.ii for [1B.2]; thus, the OSCSEs *remain in the [OS] kernel*. See also EX1002, 242 (applicants saying during prosecution that claim 3 “reinforces the fact that OSCSEs that correspond to SLCSEs remain within the kernel”).

4. Claim 4

- a. **[4A] “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...”**

204. I have reviewed the '058 patent, and “*exclusive*” appears nowhere in the '058 patent’s specification. However, the specification refers to “an application” being “provided” a “unique instance” of a SLCSE via “link[ing]” to the SLCSE library. EX1001, 3:25-29.

205. In Callender, the *SLCSEs* are in a “DLL” that the *applications* “link[]” to, as I discussed in Section VII.B.1.e.iii for [1D.3]. An instance of an SLCSE is *provided to one of the plurality of software applications* when the application calls the SLCSE from the linked library, as I discussed in Section VII.B.1.f.i for [1E.1]. Additionally, when Callender’s *SLCSEs* are called by an application, the SLCSEs run in the context of the application, and an executing SLCSE is not shared, as I discussed in Section VII.B.1.f.i for [1E.1]. That is, while the DLL containing the source code for SLCSEs is shared among applications, an instance of an SLCSE that is executed from that source code runs in the exclusive context of an

individual application in Callender, just as in the '058 patent. EX1001, 3:30-44. Thus, the application *has exclusive use of* the SLCSE instance that is *provided to* the application by virtue of the application calling the SLCSE.

206. In my opinion, POSAs would not read claim 4 to require an SLCSE to only be usable by one application; claim 4 depends from claim 1, which requires that the **executable** SLCSEs (as opposed to an **executing** SLCSE) are in a shared library and are usable by the plurality of applications (see Limitation Elements [1C.1]-[1C.2]). Nevertheless, if VirtaMove argues that claim 4 requires an SLCSE to only be usable by one application at a time, and if claim 4 were interpreted this way, this would have been an obvious implementation of Callender. POSAs understood that it was well-known to implement functions in a DLL such that even though many different applications can call functions in the DLL, only one application may call a given function at a time, using a number of well-known techniques. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Microsoft (EX1032), 472 ("semaphore" is "In programming, a signal—a flag variable—used to govern access to shared system resources. A semaphore indicates to other potential users that a file or other resource is in use and prevents access by more than one user."), 355 ("mutual exclusion" is "A programming technique that ensures that only one program or routine at a time can access some resource, such as a memory location, an I/O port, or a file, often

through the use of semaphores, which are flags used in programs to coordinate the activities of more than one program or routine.”). POSAs would have been motivated to implement the *SLCSEs*, *i.e.* the routines/functions in the user-space TCP/IP protocol library, in this manner to achieve benefits such as security and avoiding conflicts between applications. POSAs would have reasonably expected success with such an implementation because this implementation was well-known, as I noted above.

207. If VirtaMove argues that claim 4 requires an SLCSE to only be usable by a single application and not accessible to other applications at all, and if claim 4 were interpreted this way, this would also have been an obvious implementation of Callender. As I discussed above, it was known to use various techniques to only allow a single application to use a resource, and POSAs would have been able to use similar techniques to enable only one application to access the DLL itself.

b. [4B] “use system calls to access services in the operating system kernel”

208. POSAs understood a “system call” is a mechanism for “an application to invoke a kernel service.” Silberschatz (EX1018), 463 (“A *system call* is a function called by an application to invoke a kernel service.”) (emphasis original). As I explain below, POSAs understood that the user-mode sending and receiving functions, which are *SLCSEs* (as I discussed *supra* VII.B.1.d.i for [1C.1]), use

kernel-bypass features (*e.g.*, within the sending/receiving operation specifically) but also involve related operations for which the kernel is *not* bypassed.

209. Callender’s “user-mode library” has “kernel-bypass IO features.” Callender, 5:15-18. The user-mode sending and receiving functions in the user-mode library that perform the sending and receiving operations use these kernel-bypass IO features, as I discussed in Section VII.A. However, although the library has kernel-bypass features, and although Callender’s Figures 2A and 2C show an arrow going between the “HCA Adapter” and the “user-mode library,” POSAs understood that operations in the user-mode library would still access kernel services. Callender notes that “some software drivers bypass kernel mode for **certain operations**,” but Callender never refers to drivers bypassing the kernel for all operations. Callender, 1:59-61. As I explain below, POSAs understood that the user-mode sending and receiving functions use kernel-bypass features but also involve related operations for which the kernel is *not* bypassed—*e.g.*, queue setup, interrupt setup, and event notification.

i. Queue Setup

210. As I discussed in Section VII.A, Callender’s “**user-mode library**” includes the “**Vendor Specific Library**,” also called the “PMI” or “process mode independent” library. Callender, 5:20-28 (“The vendor specific library 232A of user mode library 230A maps the abstraction of InfiniBand library 220A to the

hardware-specific operations of HCA adapter 280A. The same vendor specific library appears as vendor specific library 242A of miniport driver 240A, as described in more detail below. Accordingly, vendor specific library 232A and vendor specific library 242A represent, at least in part, a process mode independent ('PMI') HCA IO access library.”), 6:45-48 (“The process mode independent library (e.g., vendor specific library 232C and vendor specific library 242B) are statically linked into both miniport 240C and user mode library 230C.”), Fig. 2A (annotated below).

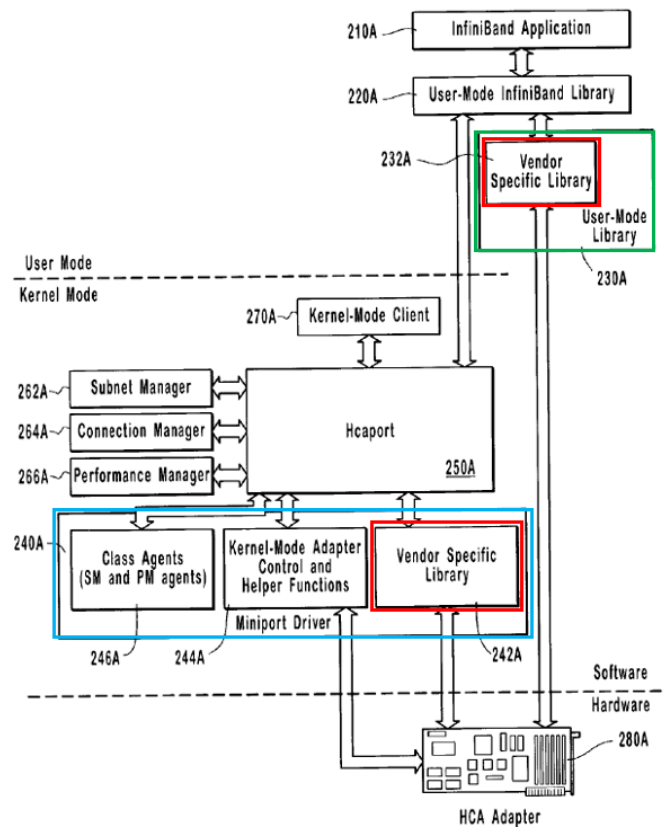


Fig. 2A

211. The functionality in the PMI library includes “the **user mode side** of protection domain / completion queue / work queue (‘PD/CQ/WQ’) creation and destruction.” Callender, 6:56-60. POSAs understood that the “completion” and “work” queues are used by the user-mode library (“user mode side”) to send data to and receive data from the adapter. *See* Callender, 6:56-7:1 (“IO requests directly manipulate an allocated queue pair”); Kagan (EX1013), [0008] (“The host network adapter in IB systems is known as a host channel adapter (HCA). An application process on the host, referred to as a ‘consumer,’ interacts with the HCA by manipulating transport service instances, known as ‘queue pairs’ (QPs). When a consumer needs to open communications with some other entity via the IB fabric, it asks the HCA to provide the necessary transport resources by allocating a QP for its use. Each QP has two work queues: a send queue and a receive queue...”), [0009] (“To send and receive messages over the IB fabric, the consumer initiates a work request (WR) on a specific QP. Submission of the WR causes a work item, called a work queue element (WQE), to be placed in the appropriate queue of the specified QP for execution by the HCA. Executing the WQE causes the HCA to communicate with corresponding QPS of other channel adapter over the network by generating one or more outgoing packets and/or processing incoming packets.”), [0010] (“When the HCA completes execution of a

WQE, it places a completion queue element (COE) on a completion queue, to be read by the consumer.”).

212. Callender explains that “[t]he PMI routines for PD/CQ/WQ creation...operate in a two step process. The first step acquires needed resources for allocation of a queue pair by the miniport in kernel mode. The second step checks the results and if a request fails, frees resources allocated to the process.” Callender, 6:57-67. POSAs understood from this explanation that when the PMI library tries to create a queue, the following operations happen: (1) the library acquires resources (i.e. the “first step” of the “user mode side” (Callender, 6:56-66)); (2) the “miniport in kernel mode” (Callender, 6:64-67) attempts to allocate the queue pair; and (3) the library checks the results (i.e., the “second step” of the “user mode side” (Callender, 6:56-67)).

213. POSAs also understood that a “miniport” is a low-level driver that “directly manages a network interface card (NIC) and provides an interface to higher-level drivers.” Buddhikot (EX1040), 15:4-12.

214. In Callender, an element “in kernel mode” is in the “kernel,” *i.e.*, the *OS kernel*, as I discussed in Section VII.B.1.c.ii for [1B.2]. Thus, POSAs understood that Callender’s “miniport in kernel mode” (6:66) is in the *OS kernel*. At minimum, POSAs would have found it obvious to implement Callender such that the miniport is part of the *OS kernel*, because POSAs understood that it was

customary to implement components like the miniport, which operates in kernel mode and provide interfaces to hardware, as part of the OS kernel. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Callender, 1:40-43 (noting in "Background" that it was typical that "the [OS] kernel acts as a gatekeeper to computer resources"); Silberschatz (EX1018), 456 ("To encapsulate the details and oddities of different devices, the kernel of an operating system is structured to use device-driver modules. The device drivers present a uniform device-access interface to the I/O subsystem[.]"), 696-697 ("describing "Linux 1.0" kernel release in 1994 and noting that "[a] range of extra hardware support was also included in this release"); Andjelic (EX1019), 1:16-34 (explaining in "BACKGROUND OF THE INVENTION" section that "important kernel functions include interrupt handling, process management and synchronization, as well as I/O management including network communications."); Oliver (EX1027), 8:64-9:5 ("Device drivers 502 may reside in a kernel space of the OS...[t]he OS may be a Windows operating system[.]"); Hsu (EX1041), 8:56-64 ("In order to accommodate the variety of specific hardware, representing peripheral devices, that can be attached to a computer system 10, the Unix operating system permits device driver control modules to be integrated with the operating system kernel. Typically, each peripheral device is required to have a supporting device driver

within the kernel to accommodate the specifics of the hardware implementing the peripheral device.”).

215. Where the miniport is part of the *OS kernel*, the miniport’s services for allocating queues in the queue pair are *services in the OS kernel*. Furthermore, POSAs would have found it obvious to implement *accessing* these services *using a system call*, because system calls were a well-known and customary way for user-mode processes to access kernel services. *See, e.g.*, Callender, 1:36-38 (noting in “Background”: “User processes switch to kernel mode when making *system calls*, generating an exception or fault, when an interrupt occurs, etc.”).

216. Finally, POSAs would have found it obvious to implement the user-mode sending and receiving functions, which are *SLCSEs* (as I explained in Section VII.B.1.d.i for [1C.1]), to check to see if a communication queue is available and create one if needed, since this was a typical operation in InfiniBand, as corroborated by, *e.g.*, Kagan (EX1013), [0008] (“When a consumer needs to open communications with some other entity via the IB fabric, it asks the HCA to provide the necessary transport resources by allocating a QP for its use.”). Such an implementation would be within a POSA’s skill because it involves known, simple programming techniques, like conditional instructions. Furthermore, such an implementation would beneficially ensure that queues needed for communication are available when needed but not allocated before they are needed.

217. When the sending and receiving functions are implemented this way, they each *use a system call to access services in the OS kernel*, and thus the SLCSEs collectively *use system calls to access services in the OS kernel*.

ii. Interrupt Setup and Event Notification

218. The '058 patent mentions initializing “interrupt handling” as an example of an SLCSE using a system call to access kernel services. EX1001, 2:64-65, 8:53-59.

219. Callender’s HCA adapter uses the InfiniBand standard, as I discussed in Section VII.A. POSAs understood that communication under the InfiniBand standard involved generating “interrupts” that result in “events” that signify the “completion” of an operation. Kagan (EX1013), [0007] (“Packet network communication adapters are a central element in new high-speed, serial input/output (I/O) bus architectures.... A number of architectures of this type have been proposed, culminating in the ‘InfiniBand™’ (IB) architecture.... delays due to interrupt handling and event generation can cause significant performance bottlenecks in IB systems.”), [0008] (“An application process on the host, referred to as a ‘consumer,’ interacts with the HCA by manipulating transport service instances, known as ‘queue pairs’ (QPs). When a consumer needs to open communications with some other entity via the IB fabric, it asks the HCA to provide the necessary transport resources by allocating a QP for its use. Each QP

has two work queues: a send queue and a receive queue...”), [0009] (“To send and receive messages over the IB fabric, the consumer initiates a work request (WR) on a specific QP. Submission of the WR causes a work item, called a work queue element (WQE), to be placed in the appropriate queue of the specified QP for execution by the HCA...”), [0010] (“When the HCA completes execution of a WQE, it places a completion queue element (CQE) on a completion queue, to be read by the consumer. The CQE contains or points to the information needed by the consumer to determine the WR that has been completed. To inform the consumer that it has added a new COE to its completion queue, the HCA typically asserts an interrupt on the host CPU. The interrupt causes the host to generate and process an event, as described above, so that the consumer process receives the completion information written by the HCA.”).

220. Callender’s “miniport” provides “adapter control and helper functions.” Callender, 5:31-33. These functions include “management” of “interrupts” and also include “kernel mode helper functions” including “event request management, such as the creation and destruction of the event queue and the processing of events from the event queue.” Callender, 6:13-33.

221. POSAs understood, or would at least have found obvious, that the user-mode sending and receiving functions, *i.e.*, *SLCSEs* (as I discussed in Section VII.B.1.d.i for [1C.1]), rely on the interrupt and event-request management

functions in the miniport, because these functions are a normal part of sending and receiving data in the InfiniBand standard, as I discussed above. The services provided by the miniport, including the interrupt and event-request management services, are *services in the OS kernel* for the same reasons I discussed in Section VII.B.4.b.i with respect to the queue setup operation. Furthermore, POSAs would have found it obvious to implement *accessing* these services *using a system call*, as I also discussed in Section VII.B.4.b.i. When the sending and receiving functions are implemented this way, they each *use a system call to access services in the OS kernel*, and thus the SLCSEs collectively *use system calls to access services in the [OS] kernel*.

5. Claim 5

- a. [5A]: “A computing system according to claim 1 wherein the operating system kernel comprises a kernel module...”**

222. A “kernel module” is “[a] set of functions that reside and execute in kernel mode as extensions to the [OS] kernel.” EX1001, 6:56-59. The parties’ agreed litigation construction adopts this definition. EX1058, 6.

223. Callender discloses an “Hcaport” that operates in “kernel mode” and performs tasks including “establish[ing] an operating environment that provides the miniport with operating system functionality.” Callender, 5:29-58, Fig. 2A (showing Hcaport in “Kernel Mode”). Such an element operating in “kernel

mode” is in the “kernel,” *i.e.*, the *operating system kernel* (as I explained in Section VII.B.1.c.ii for [1B.2])). The Hcport is therefore a “set of functions that resides and executes in the kernel.” EX1001, 6:56-59.

224. Additionally, POSAs understood that the Hcport could be an “extension” to the OS kernel (EX1001, 6:56-59). An “extension” is a “module that adds functionality to or extends the effectiveness of a program.” Microsoft (EX1035), 203. Callender says the Hcport “adds functionality to” the kernel by “provid[ing] the miniport” in the kernel “with operating system functionality.” Callender, 5:29-65.

225. Having the Hcport reside and execute in the kernel as an extension of the kernel would have also been obvious. The Hcport operates in kernel mode and provides interfaces to hardware, as noted above. POSAs understood that it was customary to implement components with these properties as extensions of the kernel. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Callender, 1:40-43 (noting that it was typical that “the [OS] kernel acts as a gatekeeper to computer resources”); EX1001, 6:57-59 (’058 patent admitting that “[i]t is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.”); Silberschatz (EX1018), 456 (“To encapsulate the details and oddities of different devices, the kernel of an operating system is structured to use device-driver modules. The device drivers

present a uniform device-access interface to the I/O subsystem[.]”), 696-697 (describing “Linux 1.0” kernel release in 1994 and noting that “extra hardware support was also included in this release”); Andjelic (EX1019), 1:16-34 (explaining in “BACKGROUND OF THE INVENTION” section that “important kernel functions include interrupt handling, process management and synchronization, as well as I/O management including network communications.”); Oliver (EX1027), 8:64-9:5 (“Device drivers 502 may reside in a kernel space of the OS...[t]he OS may be a Windows operating system[.]”); Hsu (EX1041), 8:56-64 (“In order to accommodate the variety of specific hardware, representing peripheral devices, that can be attached to a computer system 10, the Unix operating system permits device driver control modules to be integrated with the operating system kernel. Typically, each peripheral device is required to have a supporting device driver within the kernel to accommodate the specifics of the hardware implementing the peripheral device.”).

226. Thus, Callender’s *[OS] system kernel comprises a kernel module, i.e., the Hcaport.*

- b. [5B]: “adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.”**

227. The ’058 patent does not define “*interface*,” but it describes an embodiment where a “kernel module” acts as a “device interface” enabling “data

exchange between a user mode CSE and a device driver in kernel mode.”

EX1001, 9:60-10:20. The patent also describes an exemplary “kernel module” serving as an “interface” to a device driver by notifying a “service in the context of an application” of an “interrupt” from the device. EX1001, 8:46-52, Fig. 5 (below). These examples are consistent with how POSAs understand “interface” in the context of applications and device drivers. See Microsoft (EX1035), 279 (“interface” is “[s]oftware that enables a program to work with...another program...or with the computer’s hardware”).

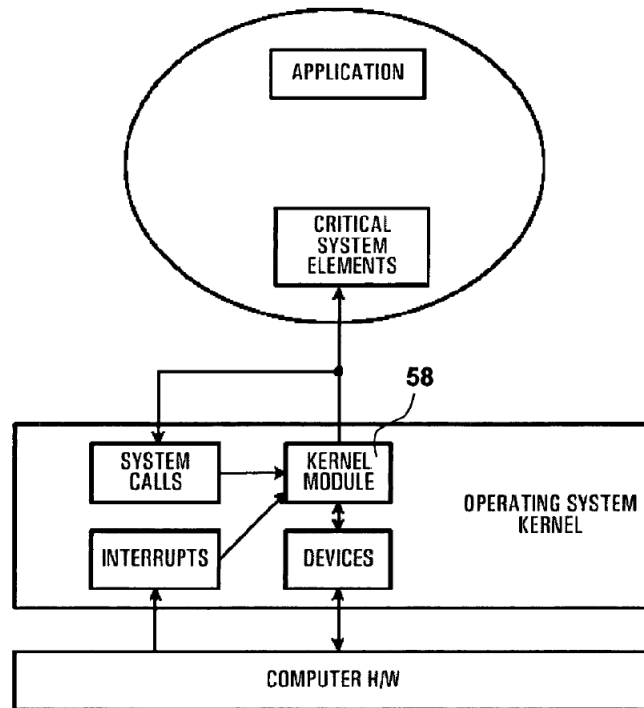


FIG. 5

228. The sending and receiving functions in Callender’s “user mode library” are *SLCSEs*, as I discussed in Section VII.B.1.d.i for [1C.1]. POSAs

understood that these functions access the services in the “miniport” for queue creation, interrupt setup, and event notification. See Section VII.B.4.b ([4B]). Callender states that the Hcaport “provides an abstracted general HCA device *interface.*” Callender, 5:43-44, Fig. 2A (annotated below).

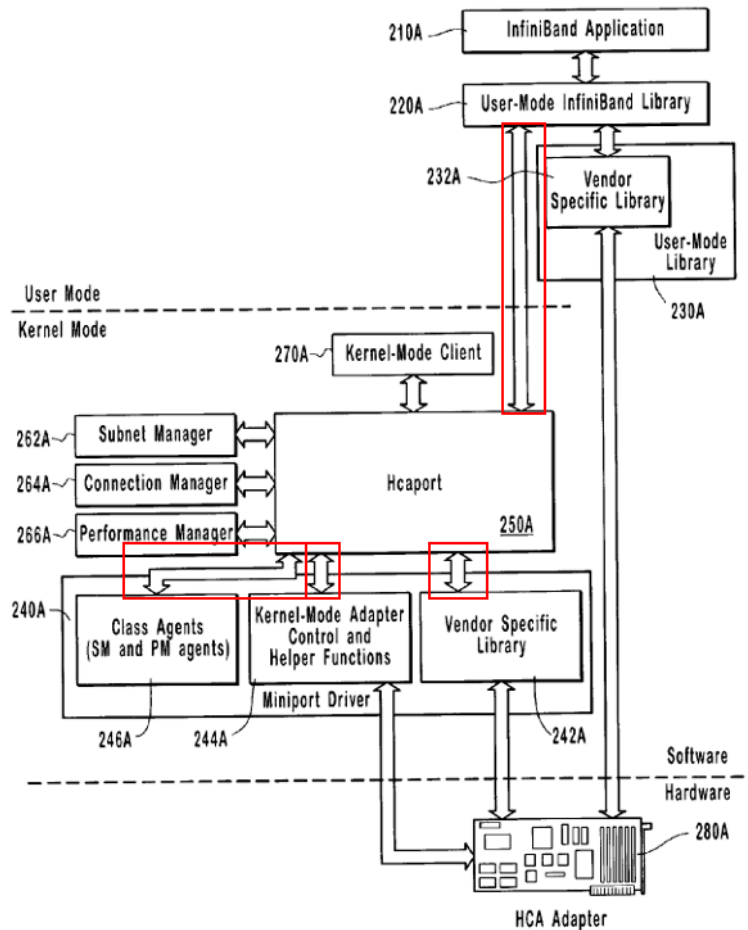


Fig. 2A

229. For multiple reasons, POSAs understood, or would at least have found obvious, that the user-mode library accesses the Miniport through the Hcaport. As I explained in Section VII.B.4.b for [4B], the user-mode library has access to

components (like the Hcaport) that run in kernel mode. Moreover, Callender's figures show arrows going from the "User-mode InfiniBand Library" to the Hcaport and then from the Hcaport to the miniport (which indicates to POSAs that user-mode processes can access the miniport through the Hcaport) and a bidirectional arrow between the user-mode library and user-mode InfiniBand library (which indicates to POSAs that the user-mode library can access user-mode InfiniBand library services as well as vice-versa). Callender, Figs. 2A, 2C.

230. In such an implementation, the Hcaport is *adapted to serve as an interface between* the sending or receiving function and a *device driver*, i.e., the miniport (*see* Section VII.B.4.b.i, discussing [4B]). The sending/receiving functions are *SLCSEs* running *in the context of an application*, as I explained in Section VII.B.1.f.i for [1E.1].

231. Thus, Callender discloses or renders obvious that the Hcaport, which is *a kernel module* (as I explained in Section VII.B.5.a), is *adapted to serve as an interface between an SLCSE in the context of an application program and a device driver* (the miniport).

6. **Claim 6: “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.”**

232. POSAs understood that Callender’s HCA adapter receives data in “packet” form, since the HCA adapter uses the InfiniBand protocol (*see* Section VII.A). *See* Kagan (EX1013), [0007] (“***Packet*** network communication adapters are a central element in new high-speed, serial input/output (I/O) bus architectures that are gaining acceptance in the computer industry.... A number of architectures of this type have been proposed, culminating in the ‘InfiniBand™’ (IB) architecture, which is described in detail in the InfiniBand Architecture Specification.”); Lin (EX1042), 1:44-49 (“Please see FIG. 1, which shows a block diagram of a packet receiving structure used in transmission technology of Infiniband. As shown in FIG. 1, an embodiment comprises a host channel adapter 1 (HCA 1), a hardware module of which supports two or more physical layer ports in order to receive packets from physical layer 2...”). POSAs also understood that receiving data in InfiniBand involves the HCA adapter generating an “interrupt” that results in an “***event***” that signals to an application that incoming packets have been processed. *See* my discussion of Kagan (EX1013), [0007]-[0010] in Section VII.B.4.b.ii above (*e.g.*, paragraph 219).

233. An “*asynchronous*” event is one that may occur at any time. Microsoft (EX1035), 38 (“asynchronous” means “Pertaining to, being, or characteristic of something that is not dependent on timing. For example, asynchronous communications can start and stop at any time instead of having to match the timing governed by a clock.”). POSAs understood that packet arrival is an *asynchronous event*. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, Griffin (EX1043), 1:12-17 (“On the majority of computer hardware systems hosting commercial operating systems, devices exist whose function it is to generate hardware interrupts to signal the occurrence of asynchronous events such as a packet arriving on a wire for a network interface card (‘NIC’) or an event signaling the completion of a disk I/O request.”).

234. POSAs also understood that this event *requires information to be passed to* the user-mode receive function, which is *an SLCSE that runs in the context of an application program* (as I discussed in Section VII.B.1.f.i for [1E.1]), *from outside the application*, for two reasons: (1) the event requires the SLCSE to be notified of the packet’s arrival by the miniport, which is *outside* the application; and (2) in response to the event, the SLCSE gets incoming data from the HCA, which is also outside the application. Kagan (EX1013), [0009]-[0010]; Callender, Fig. 2A (below).

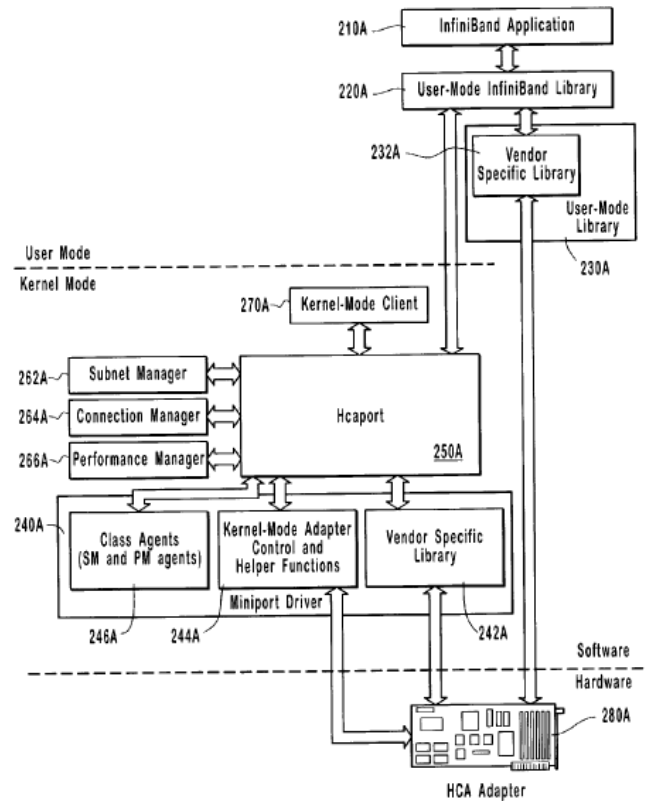


Fig. 2A

235. While Callender’s miniport provides “interrupt” and “event request” “management” (as I discussed in Section VII.B.4.b.ii for [4B]), Callender’s Hcapiport, *i.e.*, the *kernel module*, is the interface between the user-mode receive function and the miniport (as I discussed in Section VII.B.5.b for [5B]). Thus, POSAs understood that the Hcapiport *is adapted to provide a notification* of the above-discussed *event* to the receive function by passing the notification from the miniport. *See* my discussion of interrupts and events in InfiniBand in, *e.g.*, paragraph 219 above (citing Kagan (EX1013), [0007]-[0010]).

7. Claim 7: “A computing system according to claim 6, 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.”

236. POSAs understood that “*notifying the SLCSE*” includes notifying about the *event* recited in claim 6 (from which claim 7 depends). As I discussed in Section VII.B.6 for claim 6, this *event* is caused by an interrupt. See Kagan (EX1013), [0007]-[0010].

237. Callender’s miniport is the *kernel model* of claim 5 (from which claim 6 depends, and from which claim 7 in turn depends). See Section VII.B.5.a ([5A]). The miniport performs “helper” functions including “management” of “interrupts.” Callender, 6:11-20. POSAs understood that it was customary to manage interrupts using an interrupt handler. My testimony regarding a POSA’s background knowledge is corroborated by, e.g., Kagan (EX1013), [0003] (“To deliver data to a host central processing unit (CPU), high-speed network communication adapters typically use direct memory access (DMA) to write the data to the host system memory. When the adapter has finished writing, it asserts an interrupt on one of the interrupt lines of the host bus or on an appropriate pin of the CPU itself. The interrupt causes the CPU to call an interrupt handler routine. While the interrupt handler is running, other software processes are blocked, and all interrupts are disabled.”).

238. POSAs would also have found it obvious to implement the interrupt handler to notify the user-mode receive function, *i.e.* the *SLCSE*, of the *event* that an incoming packet has been processed via an *upcall mechanism*, which is “[a] means by which a service in kernel mode executes a function in a user mode application context.” EX1001, 6:60-61. The ’058 patent did not invent upcalls, which were well-known in the art, as I discuss below.

239. POSAs understood that using an upcall to inform user-mode processes about interrupts was a known technique that would have been within a POSA’s skill that would have beneficially signaled the user-mode receive function to read incoming data. My testimony regarding a POSAs’s background knowledge is corroborated by, *e.g.*: Lever (EX1044), [0042] (“WINDOWS CE™ balances performance and ease of implementation by breaking interrupt processing into two parts, a kernel-mode part and a user-mode part. The kernel mode part is called performs overhead operations corresponding to the ISR, while the core of the ISR is performed by the user-mode part. Optionally, all of the ISR operations can be performed in the kern[e]l-mode part.”); Chow (EX1045), 26:36-53 (“BYNET currently supports two basic types of messages, an in-band message, and an out-of-band message.... With a BYNET out-of-band message, the header data in a circuit message causes the interrupt handler in the BYNET driver to create the channel program that is used to process the rest of the circuit data being received. For both

types of messages, the success or failure of a channel program is returned to the sender via a small message on the BYNET back channel. This back channel message is processed as part of the circuit shutdown operation by the channel program at the sender.... After the circuit is shutdown, an up-call interrupt is (optionally) posted at the destination to signal the arrival of a new message.”); Whitmore (EX1046), 14:55-59 (“If the server 70 or application is interrupt driven with upcalls/notifications into the application layer, then the server 70 can engage the application to run before the server’s TCP stack can provide an acknowledgement to the content request 58.”).

8. Claim 8: “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.”

240. Callender’s kernel-mode processes have access to “all available memory,” which POSAs understood includes user-mode memory. Callender, 1:38-41. Thus, POSAs understood that *in operation*, the *up call mechanism* I discussed in Section VII.B.7 for claim 7 that is used by the interrupt handler to invoke the user-mode receive function, which is *an SLCSE*, can *execute instructions from* that SLCSE, which is *resident in user mode space*, while remaining in *kernel mode*.

241. POSAs would have been motivated to implement the interrupt handler this way to beneficially avoid the “impact [on] performance” of switching to user

mode, and would reasonably have expected success, because Callender’s kernel-mode processes can access all memory. Callender, 1:53-54.

9. Claim 9: “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.”

242. The ’058 patent says that “[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.” EX1001, 8:62-64. The ’058 patent did not invent overlays, which were known in the prior art, as I discuss below.

243. POSAs understood that the user-mode sending and receiving functions are “replace[ments]” for “function[s]” in the OS “that would normally be called” (EX1001, 8:62-64) because the sending and receiving operations these functions perform were normally provided by the OS, as I discussed in Section VII.B.1.d.ii for [1C.2].

244. Thus, in Callender, *a function overlay is used to provide one of the plurality of software applications access to operating system services* because the sending and receiving services normally supplied by the OS to applications are now supplied by user-mode functions, as I discussed in Section VII.B.1.d.i for [1C.1].

10. Claim 10: “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.”

245. During prosecution, the applicants asserted that then-pending claim 11, which became claim 10, “distinctly claims that user mode CSEs are part of an application” rather than “independent applications.” EX1002, 247.

246. Callender’s *SLCSEs* form a part of applications by virtue of applications linking to the DLL containing the SLCSEs, and are therefore not “independent applications” (EX1002, 247), as I discussed in Section VII.B.1.e.iii for [1D.3]. Furthermore, because Callender’s SLCSEs are part of a dynamic *link[ed]* library, they are *linked to particular software applications of the plurality of software applications*, namely the applications using the DLL, and each application has *a link that provides unique access to a unique instance of a CSE* for the reasons I discussed in Section VII.B.1.f.ii for [1E.2].

247. POSAs would have found it obvious to implement Callender to link the DLL to applications *as the applications are loaded*, because this was a standard way of linking shared libraries. For example, the ’058 patent itself admits this. *See* EX1001, 9:18-22 (CSEs “are contained in a shared library. *As such* they are *linked* to a software application *as the application is loaded*.”). As another

example, Nelson (EX1097) says: “As is well known, [a] DLL...is dynamically linked” to applications “*when they are loaded* or run.” Nelson (EX1097), 18:31-46. POSAs understood that when Nelson says “loaded or run,” Nelson is not referring to “loaded” and “run” as two distinct options for when the DLL can be linked; rather, Nelson refers to “load[ing]” as part of the “run[ning]” process, so Nelson’s reference to linking DLLs when applications are “loaded or run” is a reference to linking DLLs when they are loaded. *See* Nelson (EX1097), 18:37-40 (“At run time, ether the system loader or the tasks entry code arranges for library calls to be patched with the addresses of the real shared library routines.”). As another example, Hunt (EX1047) states:

In addition to exporting function entry points to applications, DLLs in Windows NT also export a special entry point to the operating system, the DllMain function. The DllMain function is invoked by the operating system on initialization or termination of an application or any of its threads. ... When loaded into an application’s address space, the DllMain function of the COIGN RTE DLL applies inline redirection to the COM API functions.

EX1047, 42:66-43:10.

11. Claim 11: “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.”

248. Callender’s *SLCSEs* (the user-mode sending and receiving functions, as I explain in Section VII.B.1.d.i for [1C.1]), both *utilize kernel services supplied by the [OS] kernel* for the purposes recited in claim 11, as I explained below

a. Device access

249. The miniport provides “access to [the] HCA adapter,” *i.e., device access*. Callender, 4:57-61.

250. Furthermore, SLCSEs *utilize* the “queues” the miniport creates for “IO,” *i.e. input/output, i.e., access*, to the HCA adapter. Callender, 6:56-64 (which I discussed in Section VII.B.4.b.i for [4B]).

251. The miniport also performs security functions necessary to enable the SLCSEs to access the adapter. Callender, 4:41-5:5 (“Of course, some operations are unique to kernel mode implementation 140. For example, because the kernel is responsible for enforcing security, initiating and terminating access to adapter 150 occurs through kernel mode implementation 140. Once kernel mode implementation 140 (under the direction of application 110) provides the appropriate security parameters to adapter 150, adapter 150 performs the corresponding security checks when it is accessed, such as verifying that the accessing process has been properly authorized through kernel mode

implementation 140.... kernel mode miniport driver 240A is responsible for creating and destroying communication queues, whereas user mode library 230A permits user mode data transfer to the communication queues at HCA adapter 280A.”).

252. Callender’s “miniport” is part of the OS kernel, as I discussed in Section VII.B.4.b.i for [4B]. Thus, *the SLCSEs utilize kernel services supplied by the operating system kernel, i.e. the miniport’s services, for device access.*

b. Interrupt Delivery

253. The user-mode receiving function is notified by a miniport interrupt handler when the HCA adapter has processed an incoming packet, as I discussed in Section VII.B.7 for Claim 7. Additionally, the InfiniBand sending operation involves generating an interrupt to provide notification that the HCA adapter has completed a transmission. *See Kagan (EX1013), [0009]-[0010]. Thus, the receiving and sending functions utilize kernel services as claimed for interrupt delivery.*

c. Virtual Memory Mapping

254. POSAs understood that “*virtual-to-physical*” memory “address *mapping*” is a standard *kernel service supplied by the operating system kernel*. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*, Draves (EX1017), 1:48-60. Additionally, POSAs understood that applications

linking to the “user mode library”—a DLL (Callender, 5:13-15)—would have the library in their “virtual address space.” *See* Draves (EX1017), 3:52-4:32 (discussing how “DLL (dynamic link library) is loaded in a range of virtual memory addresses within the address space of [a] user process...”), Fig. 5.

255. Thus, POSAs would have found it obvious to implement Callender’s user-mode library’s sending and receiving functions to use the kernel’s services *for virtual memory mapping* to achieve known benefits of virtual memory, *e.g.*, “isolating processes from each other” (EX1017, 1:50-56), and would reasonably have expected success since such use of the kernel was well-known.

12. Claim 12: “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.”

256. Callender’s *SLCSEs* send/receive data via the HCA adapter, which uses the “InfiniBand” networking protocol (as I discussed in Section VII.A). Therefore, the *SLCSEs include services related to at least networking protocol processes*.

257. If claim 12 also requires the *SLCSEs* to include services related to *management of files*, this would have been obvious. The ’058 patent’s specification never mentions “management,” but mentions “access[ing] files that reside in different locations” as an example of “File System services” that are *CSEs*. EX1001, 6:10-17. It was well-known to use InfiniBand to connect to

“network attached storage (NAS)” providing “file access.” Fujiwara (EX1064), [0002]-[0003]. Thus, in an obvious use of Callender’s SLCSEs to access files on NAS, the SLCSEs include services *related to the management of files*.

13. Claim 13: “A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.”

258. POSAs understood that a known alternative to using interrupts to signal completion of an InfiniBand operation (as I discussed in Section VII.B.4.b.ii for [4B]) was to poll to see if the operation is complete, and also understood that using interrupts vs. polling involved a tradeoff between reducing latency (the overall time between the start and end of an operation) via polling or reducing processor resource usage via interrupts. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Liss (EX1048), 1 (“Infiniband is a high-performance SAN architecture.... Applications can send and receive data at high rates when accessing IB through user-level networking interfaces, *e.g.*, VIA [Virtual Interface Architecture].... The introduction of high-performance user-level SANs to DSM systems...eliminated OS protocol processing, and reduced extra memory copying through remote memory operations. Responsiveness, however, remains a problem: constant polling is the most responsive method, but wastes valuable CPU cycles....”), 2 (The Verbs interface defines the semantics for utilizing various HCA resources (Fig. 1).... A Verbs consumer can poll a CQ for

completions, or request Completion Notification for a certain CQ when the next CQE is inserted.” ”); Patel (EX1049), 281 (“The VI architecture provides several mechanisms to notify completion of a descriptor. The mechanisms on a send or receive queue are poll, wait, and notify. In a poll mode the application continually checks (i.e polls) a status bit to determine whether the VI provider has completed work on a descriptor.”), 282 (“User-mode communication mechanisms...show low latencies for data transfer. Frequently, this low latency is obtained using a polling mechanism. The polling mechanism expends CPU cycles waiting for a message... VI provides an interrupt-based mechanism as an alternative to polling.”).

259. Thus, POSAs would have been motivated to *modify* the user-mode sending and receiving functions, *i.e.* the *SLCSEs*, to use polling *for a particular one of the software applications* that requires minimizing latency, and would reasonably have expected success since polling was a known technique.

14. Claim 14: “A computing system according to claim 13 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.”

260. The interrupt-based and polling-based versions of the SLCES are used by different applications, as I discussed in Section VII.B.13 above for claim 13. These different SLCSE versions are “respective versions” that different applications are “provided with,” *i.e.*, *SLCSEs that are application specific*. EX1001, 3:57-60 (“software **applications are provided with respective versions**

of [CSEs].... the system elements which are *application specific* reside in user mode[.]”). These SLCSEs are in the user-mode library, as I discussed in Section VII.B.1.d.i for [1C.1], *i.e., reside in user mode.*

261. Callender’s OS *critical system elements* (OSCSEs), are in, and therefore *reside in*, the *operating system kernel*, as I discussed in Section VII.B.1.c.ii for [1B.2]. The ’058 patent defines a “compute *platform*” as [t]he combination of computer hardware and a single instance of an operating system.” EX1001, 6:29-30. Callender’s OSCSEs are hardware-specific because they use “vendor specific library” for the HCA adapter, which is “hardware,” and they are OS instance-specific because they are part of the operating system kernel; thus, the OSCSEs are *platform specific*. Callender, 4:51-5:28; *see also* my discussion of [1B.2] in Section VII.B.1.c.ii.

15. Claim 15: “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.”

262. Callender’s Hcaport is a *kernel module*, and Callender’s “*kernel mode* miniport *driver*” (Callender, 4:56, Fig. 2A), which is a *device driver* (as I discussed in Section VII.B.5 for claim 5), is *in kernel mode*. Callender’s *SLCSEs* are in the

“*user mode* library,” as I discussed in Section VII.B.1.c.i for [1C.1], and are thus *in user mode*. Callender, 4:51-5:18.

263. The '058 patent says that “[a]s a **device interface**, the kernel module enables *data exchange* between a user mode CSE and a device driver in kernel mode” using “virtual memory such that data is transferred in both directions without a copy.” EX1001, 10:60-66. Immediately following this statement, the patent states that “[s]ervices exported **for device interface** typically include” a list of possibilities including “[i]nitialization,” “[e]stablish[ing] a channel between a CSE in user mode and a specific device,” and “[i]nform[ing] the interrupt service that this CSE requires notification.” EX1001, 9:66-10:20. Callender’s Hcaport serves as an interface between the miniport and the SLCSEs for purposes including (1) SLCSEs sending queue-creation requests to the miniport; and (2) the miniport sending event notifications to the SLCES. *See* my discussion of claim 5 in Section VII.B.5. Therefore, the Hcaport is *adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and data is transferred both* from the SLCSE to the driver and vice-versa as claim 15 recites.

264. One well-known way to exchange data between two entities was via data structures in memory, *e.g.*, queues, accessible to both entities. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Andjelic (EX1019), 2:22-39 (describing well-known “virtual interface” standard using

“queues” “mapped directly to user address space” for data exchange). POSAs would have found it obvious to *use mapping of virtual memory* to implement these data structures to achieve the well-known benefits of virtual memory, and would reasonably have expected success doing so because virtual memory mapping was well-known. *See Draves (EX1017), 1:48-60.*

16. Claim 16: “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.”

265. Callender’s SLCSEs *form a part of* software applications *by being* in a DLL *linked* to the applications, as I discussed in Section VII.B.1.f.i for [1E.1].

17. Claim 17: “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.”

266. Callender explains that “access to [the] HCA adapter...through [the] user mode library...is practical” because in InfiniBand, “[i]nitiating and terminating access are controlled through kernel mode,” but “[a]fter initiation...extremely fast user mode access” is enabled. Callender, 4:57-67. Callender also says the miniport “is responsible for creating and destroying communication queues,” whereas the user mode library “permits user mode data transfer to the communication queues at [the] HCA adapter.” Callender, 4:57-5:5. Thus, POSAs understood that after the miniport creates queues and initiates access,

the user mode sending and receiving functions do not need the kernel for anything else except for interrupts and any virtual memory access that may occur. This understanding is supported by, *e.g.*, Kagan (EX1013), which describes in its “Background” section well-known InfiniBand sending and receiving methods, which do not utilize kernel services. Kagan (EX1013), [0009]-[0010].

267. Thus, other than *utilizing kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping*, which the user-mode sending and receiving functions, *i.e. the SLCSEs*, do (as I discussed in Section VII.B.11 for claim 11), *the SLCSEs otherwise execute without interaction from the operating system kernel.*

18. Claim 18: “A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.”

268. POSAs understood that Callender discloses implementing kernel-mode sending and receiving operations and their corresponding user-mode operations with different code. Callender teaches defining a common “interface” for user-mode and kernel mode operations. *E.g.*, Callender, 7:5-20, Fig. 3; claims 1, 9, 17. While the same code **can** be used for the underlying implementations of operations (4:30-32), Callender does not **require** this; rather, Callender says “user mode implementations of a given operation **tend to differ** from kernel mode implementations” and that as an alternative, “same source code **may** be written.” Callender, 4:17-32. While Callender acknowledges the benefits of using the same

code, such as “reduc[ing] development time, testing time, and programming errors” (4:33-40), Callender does not teach away from different code.

Furthermore, POSAs understood that using different code provided benefits such as allowing the user-mode library to be updated periodically without requiring modifications to the OS kernel, thus avoiding the risk of damaging the kernel.

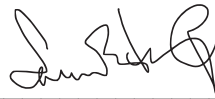
269. Thus, in one obvious implementation of Callender, the user-mode functions for the sending and receiving operations, *i.e.* *SLCSEs*, do not use the same code as, and therefore *are not copies of*, the kernel-mode implementations of those operations, *i.e.* *OSCSEs*.

* * *

I declare that all statements made herein of my own knowledge are true, that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

I declare under penalty of perjury that the foregoing is true and correct.

Dated: Jan 31, 2025



Samrat Bhattacharjee

VIII. APPENDIX: CLAIM LISTING

The following claim listing assigns element labels (e.g., 1[PRE], [1A], etc.) to certain claims for convenience of reference.

Claim 1
[1PRE] A computing system for executing a plurality of software applications comprising:
[1A] a) a processor;
[1B.1] b) an operating system having an operating system kernel having...
[1B.2] OS critical system elements (OSCSEs)...
[1B.3] for running in kernel mode using said processor; and,
[1C.1] c) a shared library having shared library critical system elements (SLCSEs) stored therein...
[1C.2] for use by the plurality of software applications in user mode and
[1D.1] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and...
[1D.2] are accessible to some of the plurality of software applications and...
[1D.3] 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,
[1E.1] 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...
[1E.2] 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, and
[1F] 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 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.
Claim 3

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

Claim 4

[4A] 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,

[4B] use system calls to access services in the operating system kernel.

Claim 5

[5A] A computing system according to claim 1 wherein the operating system kernel comprises a kernel module

[5B] adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

Claim 6

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.

Claim 7

A computing system according to claim 6 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.

Claim 8

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.

Claim 9

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.

Claim 10

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.

Claim 11

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.

Claim 12

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.

Claim 13

A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

Claim 14

A computing system according to claim 13 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.

Claim 15

[15A] 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,

[15B] 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.

Claim 16

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.

Claim 17

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.

Claim 18

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