UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

GOOGLE LLC,
Petitioner,

v.

VIRTAMOVE, CORP.,
Patent Owner.

Case No. IPR2025-00490
Patent No. 7,784,058

DECLARATION OF SAMRAT BHATTACHARJEE, PH.D.

Google Exhibit 1003
Google v. VirtaMove

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 16

- A. Background and Specification 16
- B. Person of Ordinary Skill in the Art (“POSA”) 26
- C. Prosecution History 27
- D. Challenged Claims 29

VI. Claim Construction 29

VII. Elnozahy and Draves Render Claims 1-18 Obvious 30

- A. Elnozahy 30
- B. Draves 34
- C. Elnozahy+Draves Combination 35
- D. Claim-by-Claim Analysis 39
  - 1. Claim 1 39
    - a. [1PRE]: “A computing system for executing a plurality of software applications comprising:” 39
    - b. [1A] “a) a processor;” 40
    - c. [1B] 41
      - i. [1B.1] “b) an operating system having an operating system kernel…” 41
      - ii. [1B.2] “[OS kernel] having OS critical system elements (OSCSEs)…” 43
      - iii. [1B.3] “[OSCSEs] for running in kernel mode using said processor” 48
    - d. [1C] 50

i. [1C.1] "c) a shared library having shared library critical system elements (SLCSEs) stored therein…".....50

ii. [1C.2] "for use by the plurality of software applications in user mode…".....56

e. [1D] 58

i. [1D.1] "i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and…" 58

ii. [1D.2] "are accessible to some of the plurality of software applications and…".....61

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".....63

f. [1E].....65

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…" 65

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…" 70

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.".....71

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".....74

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" 77
4. Claim 4 77
   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…" 77
   b. [4B] "use system calls to access services in the operating system kernel".....80
      i. File Retrieval.....80
      ii. Initializing interrupt handling 83
5. Claim 5: "A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver." 86
   a. Initializing Communication with Network-Interface Drivers.....88
   b. File retrieval 89
   c. Interrupt setup 92
6. Claim 6: 93
   a. [6A]: "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…".....93
      i. File retrieval.....93
      ii. Packet arrival 95
   b. [6B]: "wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application 95
      i. File retrieval.....95
      ii. Packet Arrival 96
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." 97

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." 99

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." 101

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." 102

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." 103

    a. Device access 103

    b. Interrupt Delivery.....104

    c. Virtual Memory Mapping 104

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." 106

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." 107

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." 108

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." 109

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." 112

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." 112

18. Claim 18: "A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs." 113

VIII. APPENDIX: CLAIM LISTING 116

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
1006	U.S. Patent App. Pub. No. 2003/0041118 ("Elnozahy")
1007	U.S. Patent No. 6,263,376 ("Hatch")
1008	U.S. Patent No. 6,260,075 ("Cabrero")
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-966")
1012	U.S. Patent App. Pub. No. 2004/0216145 ("Wong")
1015	Excerpts from Charles Petzold, *Programming Windows 95* (Microsoft Press 1996) ("Petzold")
1017	U.S. Patent No. 6,349,355 ("Draves")
1018	Excerpts from Silberschatz et. al, *Operating System Concepts* (Wiley 6th ed. 2002) ("Silberschatz")
1020	Chart re: '058 Patent accompanying Plaintiff VirtaMove Corp.'s Supplemental Preliminary Disclosure of Asserted Claims and Infringement Contentions, in *VirtaMove, Corp. v. Google LLC*, 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")
1024	U.S. Patent No. 7,080,172 ("Schmalz")
1032	U.S. Patent No. 5,375,241 ("Walsh")
1033	U.S. Patent No. 6,698,015 ("Moberg")
1034	Excerpts from Collin, *Dictionary of Computing* (Collin 4th Ed. 2002) ("Collin")
1035	Microsoft Computer Dictionary (Microsoft 5th ed. 2002) ("Microsoft")
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")
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, *Efficient Exploitation of Kernel Access to Infiniband: a Software DSM Example*, 11th Symposium on High Performance Interconnects, 2003 ("Liss")
1049	Patel *et. al.*, *A Model of Completion Queue Mechanisms Using the Virtual Interface API*, Proc. IEEE Int'l Conf. on Cluster Computing, 281-282 (IEEE 2002) ("Patel")
1052	Google LLC's Proposed Claim Terms for Construction, *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 1, 2024)
1053	Plaintiff's Disclosure of Proposed Claim Constructions, *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 1, 2024)
1056	U.S. Patent No. 5,278,969 ("Pashan")
1057	Excerpts of *Webster's New World Dictionary* (4th Ed. 2003)
1058	Joint Claim Construction Statement, *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Dec. 18, 2024)
1059	Google's Opening Claim Construction Brief, *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 22, 2024)
1065	U.S. Patent No. 6,442,752 ("Jennings")
1067	VirtaMove's Responsive Claim Construction Brief, *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Nov. 12, 2024)
1068	VirtaMove's Surreply Claim Construction Brief, *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Dec. 13, 2024)
1069	U.S. Patent No. 7,213,247 ("Wilner")
1072	U.S. Patent App. Pub. No. 2003/0037337 ("Yona")
1073	U.S. Patent App. Pub. No. 2002/0083127 ("Agrawal")
1074	U.S. Patent App. Pub. No. 2002/0004834 ("Guenther")
1075	U.S. Patent No. 6,049,892 ("Casagrande")
1076	U.S. Patent App. Pub. No. 2003/0023580 ("Braud")
1077	U.S. Patent No. 6,917,627 ("Footer")
1078	U.S. Patent No. 6,361,335 ("Calanni")
1079	U.S. Patent App. Pub. No. 2002/0078135 ("Venkatsubra")
1080	U.S. Patent No. 6,594,690 ("Cantwell")
1081	U.S. Patent No. 5,394,547 ("Correnti")
1082	U.S. Patent No. 5,931,925 ("McNabb")
1083	U.S. Patent No. 5,966,543 ("Hartner")
1084	U.S. Patent No. 7,171,494 ("Karamanolis")

1085	U.S. Patent No. 6,029,160 ("Cabrera")
1086	U.S. Patent No. 6,931,501 ("Narayanaswamy")
1087	U.S. Patent No. 5,630,141 ("Ross")
1088	U.S. Patent App. Pub. No. 2003/0007488 ("Rao")
1091	U.S. Patent No. 7,937,559 ("Parameswar")
1092	U.S. Patent No. 7,200,761 ("Freeman")
1093	U.S. Patent App. Pub. No. 2002/0027909 ("Brinkerhoff")
1094	U.S. Provisional Patent Application No. 60/504,213
1095	Redline Comparison between U.S. Provisional Patent Application No. 60/504,213 and specification of U.S. Patent No. 7,784,058
1096	U.S. Patent No. 6,728,839 ("Marshall")
1098	U.S. Patent No. 5,815,701 ("Slavenburg")
1099	U.S. Patent No. 5,581,768 ("Garney")
1100	European Patent Application EP1164480 ("Temple")
1101	U.S. Patent Application Pub. No. 2002/0129085 ("Kubala")
1102	U.S. Patent No. 6,874,144 ("Kush")
1103	U.S. Patent No. 5,630,076 ("Saulpaugh")
1104	U.S. Patent No. 5,983,021 ("Mitrovic")
1105	U.S. Patent No. 5,305,461 ("Feigenbaum")
1106	U.S. Patent App. Pub. No. 2002/0032821 ("Garrigues")

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. The table below identifies the references, applicable claims, and basis for each ground of unpatentability.

Ground Number and Reference(s)		Claims	Basis
1	Elnozahy (EX1006), Draves (EX1017)	1-18	Obviousness

30. The ’058 patent was filed on September 21, 2004. *See* EX1001, code (22). The ’058 patent claims priority to U.S. Provisional Patent Application No. 60/504,213 (the “’213 Provisional,” EX1094), which was filed September 22, 2003. *See* EX1001, code (60).

31. I am informed and understand that for a patent to be entitled to the priority date of a provisional application, the provisional application must provide written description support for the patent’s claims, meaning the provisional 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 provisional application’s filing date.

32. I have reviewed the ’213 Provisional and the ’058 patent. Based on my review, it is my opinion that the ’213 provisional does not provide written description support for the ’058 patent’s claims, at least because the ’213 Provisional does not provide written description support for all limitations of claim 1 of the ’058 patent, which is the sole independent claim, as I explain below.

33. As one example, Element [1E.1] recites: “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." *See* the claim listing in Section VIII below. I have reviewed the '213 Provisional and I see no discussion of this limitation. To the extent there is any language in the '058 patent related to this limitation at all, this language appears only in the '058 patent. As I discuss in Section VII.D.1.f.i below for Element [1E.1], the '058 patent that there is one copy of the computer "instructions" (or "code") representing an SLCSE that is "shared by all applications," but each application "uses [its own] separate data area[]" when running the SLCSE, and "[i]n this manner," "CSEs are not shared among applications even though the code is shared." EX1001, 3:30-44. Based on my review, this discussion does not appear in the '213 Provisional. This is confirmed by EX1095, which is an automatically-generated redline showing differences between the '058 patent's specification and the '213 provisional; as shown at p. 3-4, the relevant language I discussed above appears in the '058 patent's specification but not in the '213 Provisional.

34. Elnozahy was published on February 27, 2003. Thus, because the '058 patent's filing date was September 21, 2004, I understand that Elnozahy is prior art to the '058 patent.

35. Furthermore, Elnozahy was filed on August 23, 2001. Therefore, even if the ’058 patent is entitled to claim priority to the ’213 Provisional (which in my opinion it is not), Elnozahy is still prior art to the ’058 Patent.

36. Draves issued on February 19, 2002. Therefore, I understand that Draves is prior art to the ’058 Patent.

37. Elnozahy is a publication of a patent application. I understand that this application eventually resulted in an issued patent, Elnozahy-966 (EX1011). As I discuss in Section V.C, Elnozahy-966 was of record during prosecution of the ’058 patent but was not considered in combination with Draves.

V. THE ’058 PATENT

A. Background and Specification

38. 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 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").

39. The computer's "operating system (OS)" (EX1001, 2:1) "traditionally…provides mechanisms to…control [the applications'] access to shared resources." EX1001, 1:22-24. 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.

40. 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. 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.").

41. 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 gives the kernel 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."); Draves, 1:23-47 (comparing user mode and kernel mode with each other and noting that user mode is "non-privileged," in contrast to "kernel mode," which Draves also calls "privileged mode").

42. 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."), 6:32-35 ("In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode."); Draves, 1:23-47.

43. 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).

44. 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.[1]

45. 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 (which the '058 patent also refers to as "contexts," *see* EX1001, 6:31-36) between user and kernel mode takes time, and thus could negatively impact performance. My testimony is corroborated by, *e.g.*, Slavenburg (EX1098), 1:29-32 ("In designing processors, there are a number of cost/performance tradeoffs. Higher performance often comes at the expense of interrupt overhead, interrupt latency, and context switch degradations."). 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").

46. 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,

[1] All emphases added unless otherwise indicated.

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”).

47. 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.

48. 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.

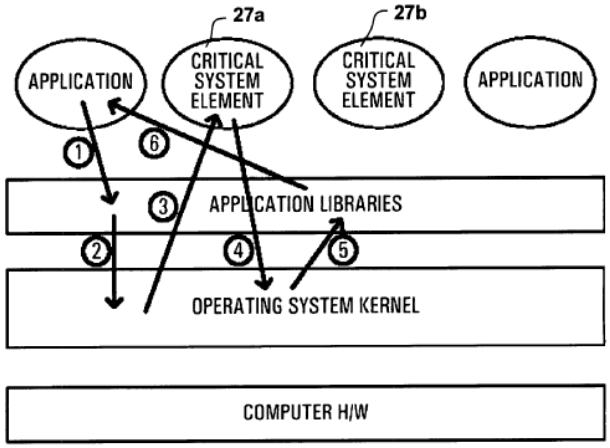


FIG. 2a
Prior Art

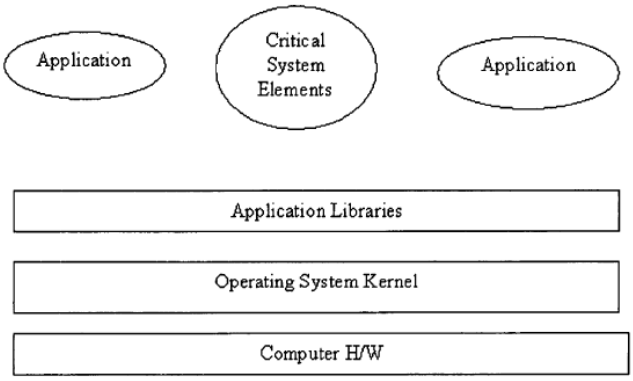


FIG. 2b
Prior Art

49. 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.

50. “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.”).

51. 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").

52. 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.

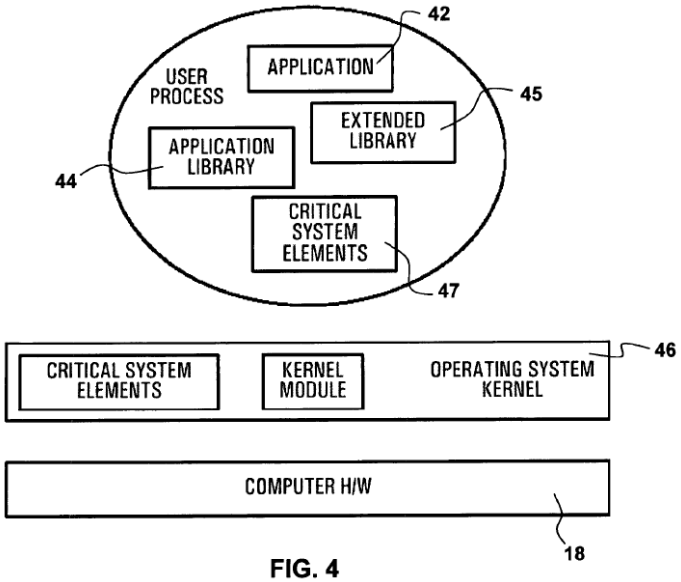


FIG. 4

53. 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.").

54. 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….").

55. 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, the combination of Elnozahy and Draves 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”)

56. 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.

57. 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.

58. 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

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

60. 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.

61. 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.

62. 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-966 (EX1011) and Wong (EX1012).

63. 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.

64. The examiner stated that neither Elnozahy-966 nor Wong disclosed the amended language of claim 1. I have reviewed Elnozahy-966 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 Draves having been before the examiner during prosecution. I also see no evidence that the examiner considered Elnozahy-966 in combination with a reference like Draves, which discloses shared libraries as I discuss in Section VII.B below, in lieu of Wong.

D. Challenged Claims

65. I have been asked to determine whether claims 1-18 would have been obvious over the combination of Elnozahy and Draves. Claims 1-18 are reproduced in the Claim Listing (§VIII below).

VI. CLAIM CONSTRUCTION

66. 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.

67. 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 regardless 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. ELNOZAHY AND DRAVES RENDER CLAIMS 1-18 OBVIOUS

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

A. Elnozahy

69. Elnozahy is directed to a "web server that integrates portions of operating system code to execute substantially within user space." Elnozahy, [0008]. POSAs understood that except in Elnozahy's claim 9 (which I discuss in Section VII.D.1.b below), Elnozahy uses "web server" to refer to server software, as opposed to the computer the software runs on, which Elnozahy calls the "server device." Elnozahy, [0015] ("…the present invention contemplates a web server application…"), [0017] (referring to "the server device on which web server 200 resides").

70. Elnozahy explains that conventionally, when the web server software receives a "request" (*e.g.*, for a web page) from a "client" (*e.g.*, an end-user computer), the request was processed by a "Transmission Control Protocol/Internet Protocol ('TCIP/IP') block." Elnozahy, [0005]. POSAs understood that TCP/IP was a well-known networking protocol. Elnozahy, [0005] (referring to "network protocol routines indicated… by the [TCP/IP] block").

71. This TCP/IP block is conventionally invoked by the "operating system kernel"; this block "extracts an HTTP formatted request" from one or more "packets" in the request and then passes the HTTP request to the server. Elnozahy, [0005], Fig. 1 (below).

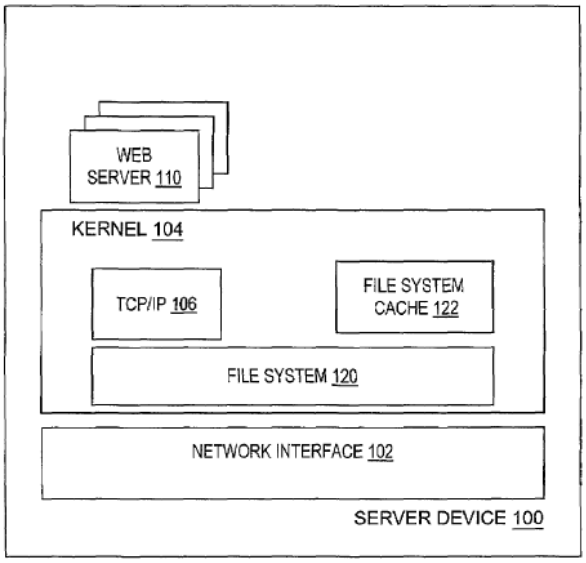


FIG. 1
PRIOR ART

72. POSAs understood that HTTP, or "Hypertext Transfer Protocol," was a well-known protocol for formatting requests from a browser to a web server and for formatting the server's response. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*, EX1035, 259 ("HTTP" is an "[a]cronym for Hypertext Transfer Protocol. The protocol used to carry requests from a browser to a web server and to transport pages from web servers back to the requesting browser.").

73. In contrast to the above-described scheme that uses the kernel, Elnozahy discloses a web server application that includes a "user space transmission protocol library that enables the server to perform its own network processing of requests and responses." Elnozahy, [0008], Fig. 2 (below). This library, which Elnozahy also refers to as a "network protocol library," includes "TCP/IP library routines that provide a user space TCP/IP protocol stack" to the web server. Elnozahy, [0022].

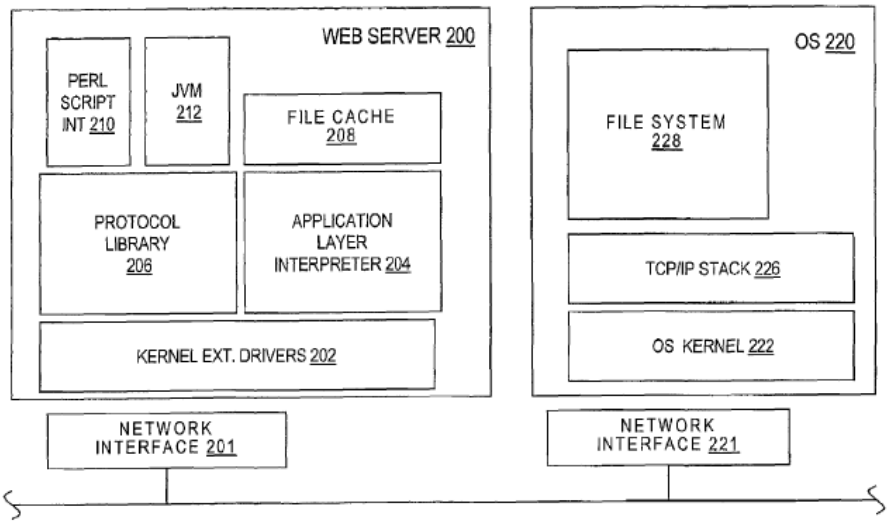


FIG. 2

74. The web server also uses "kernel extension drivers" to communicate directly with a "network interface" without the need for "kernel calls." Elnozahy, [0020]. POSAs understood that a "network interface" is a hardware device that

lets a computer access a network. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*, EX1035, 363.

75. Additionally, the operating system has its own "protocol stack," also called a "TCP/IP block" or "TCP/IP Stack," that "interfaces with [a] general purpose network interface" that is separate from the interface used by the web server application; this "general purpose" interface is used by "other applications" or used for "other functions" than those performed by the web server. Elnozahy, [0005] (referring to "Transmission Control Protocol/Internet Protocol (TCP/IP) block 106" shown in Fig. 1), Fig. 1 (below, showing "TCP/IP 106" in the "kernel"), [0017] ("If the server device on which web server 200 resides is configured to execute other applications or perform other network functions, a general purpose interface 221 distinct from interface 201 is included to handle them."), [0025] (referring to "extensions of the operating system kernel code and protocol stack").

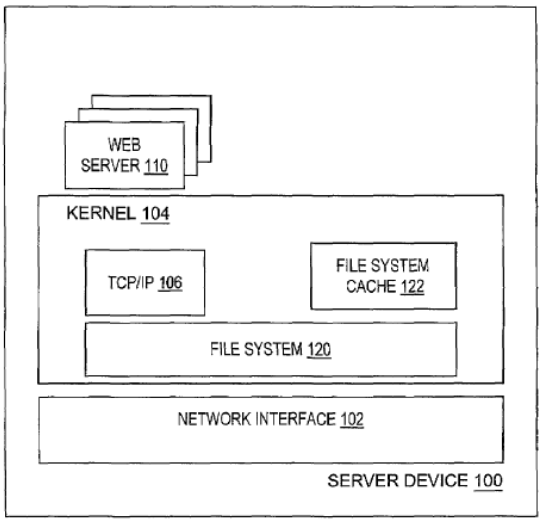


FIG. 1
PRIOR ART

B. Draves

76. Draves is directed to allowing "sharing of executable program modules between kernel-mode threads and user-mode threads." Draves, 4:51-57 ("…most operating systems do not allow sharing of executable program modules between kernel-mode and user-mode threads. The inventors, however, have discovered a way to allow such sharing…"), 4:65-5:5 ("The virtual addresses included in the two respective address spaces are not mutually exclusive…. Within both the user address space and the kernel address space, this shared range of addresses is designated for shared executable components."). POSAs understood that a "thread" is a task executed by a processor. *See*, *e.g.*, EX1035, 518 ("thread" is "a process that is part of a larger process or program").

77. Draves accomplishes this sharing by creating a "shared range of addresses in memory" that appears in both the "user address space and the kernel address space" and placing "shared executable components" in that address space. Draves, 4:65-5:5. An "address space" refers to a range of memory locations. EX1035, 20. One exemplary "program module" Draves discusses that is "sharable" between user- and kernel-mode threads using Draves's technique is a "DLL (dynamic link library)." Draves, 3:57-61. 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* Draves, 3:57-61; EX1001, 2:48 ("dynamic linked library (DLL)")

C. Elnozahy+Draves Combination

78. POSAs would have been motivated to implement Elnozahy's user-space "protocol library" as a shared library usable by multiple applications, for the reasons I explain below.

79. Elnozahy expressly contemplates "other applications" running on the same computer as the web server application. Elnozahy, [0017]. Elnozahy mentions the "other applications" using the "general purpose interface 221," and also mentions that a "protocol stack 226," which is labeled "TCP/IP Stack 226" in

Figure 2, "interfaces with the general purpose network interface." Elnozahy, [0017], [0025].

80. Thus, Elnozahy discloses other applications aside from the web server running on the same computer as the web server and also using TCP/IP. This is consistent with a POSA's background knowledge that it was well-known to run multiple applications using TCP/IP on a single computer. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Yona (EX1072), [0007] ("E-Mail applications use transmission control protocol/Internet protocol (TCP/IP)."); Agrawal (EX1073), [0031]-[0032] (instant messaging); Guenther (EX1074), [0057] (examples including email, news, and terminal emulation); Casagrande (EX1075), 1:26-40 ("file transfer protocol (FTP)"); Braud (EX1076), [0044] (known for multiple TCP/IP applications differentiated by an assigned "port" to run on same server); Footer (EX1077), 3:39-62 (disclosing "server" running "several different application programs" using TCP/IP: "In the system 10, at least one application server 12 contains at least one application program 14…. The application program 14 preferably can operate with any interactive communications protocol, such as TCP/IP…. Separate application servers 12 may be used to implement several different application programs 14, or one application server 12 may house several different application programs 14.").

81. Furthermore, Elnozahy discloses that its user-space protocol library provides benefits such as "improved" "performance" due to "context switching and protocol overhead" being reduced. Elnozahy, [0015]. Additionally, POSAs understood that it was "desirable to implement applications in a modular fashion, and to re-use modules already implemented by other applications." Wilner (EX1069), 2:21-23.

82. Thus, in view of (i) Elnozahy's own teachings about the advantages of its user-space protocol library and (ii) a POSA's background knowledge about the desirability of module re-use, POSAs would have been motivated to implement Elnozahy's user-space TCP/IP stack in a shared "protocol library" (Elnozahy, [0022]) that at least some of the "other applications" (Elnozahy, [0017]) use.

83. Additionally, Draves discloses that a DLL is a type of "potentially [] sharable program module." Draves, 3:57-61. POSAs understood that a DLL was a library and was a well-known and customary way to share functionality between applications, as the '058 patent itself acknowledges. 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."); 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"); Walsh (EX1032), 1:28-35 ("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. These libraries play an important role in operating systems, such as Windows and OS/2, which use them to make their services and resources available to the application programs.").

84. POSAs would thus have been motivated to implement Elnozahy's user-space protocol library as a DLL.

85. In the resulting Elnozahy+Draves combination, Elnozahy's user-space "protocol library" (Elnozahy, Abstract, [0008]) is implemented as a DLL as taught in Draves (Draves, 3:57-61). This DLL is shared with Elnozahy's "web server" as well as with "other applications" running on the same computer that also use TCP/IP, as discussed above. Elnozahy, [0017]. POSAs would have reasonably expected success with such an implementation, for multiple reasons. First, as I discussed above, it was known for multiple applications using TCP/IP to run on the same server. *See*, *e.g.*, ¶80 (citing EX1076, [0044]; EX1077, 3:39-62).

86. Second, as I also discussed above, it was known to share functionality between applications using a shared library such as a DLL. *See* ¶83 (citing EX1001, 2:45-50; EX1015, 959-960; EX1032, 1:28-35).

D. Claim-by-Claim Analysis

1. Claim 1

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

87. Elnozahy discloses a "server device" on which the "web server application" "reside[s]," where the "server device" also "***execute[s]***" other "***applications***." Elnozahy, [0015], [0017]. In Elnozahy+Draves, the "web server" and at least some other "applications" use Elnozahy's user-space protocol library that provides a TCP/IP stack, as I discussed in Section VII.C above. POSAs understood that a "server device" is a *computing system*. Calanni (EX1078), 1:33-35 (referring in "Background" to "large mainframe computers, servers or other high capacity computing systems"). Elnozahy also discloses that its invention may be "implemented as a set of ***computer executable instructions (software)***." Elnozahy, [0016].

88. Thus, POSAs understood that Elnozahy's "server device" is a *computing system for executing a plurality of software applications*, namely the applications using the user-space TCP/IP library.

b. [1A] "a) a processor;"

89. Elnozahy claims a "web server" comprising a "***processor***." Elnozahy, claim 9. Elnozahy also says that running "server class operating systems" may require a "server processor" to switch between executing different processes. Elnozahy, [0007].

90. In the district court, the parties have agreed that a *processor* should be construed as a "physical computer processor." EX1058, 6. POSAs understood, or at least would have found obvious, that the processor is a *physical* computer processor because it was customary for processors in "computer" "systems" to be implemented as physical processors. *See* EX1096, 1:34-39 ("Background" describing a "processor" as a typical "hardware unit" in "computer architecture").

91. POSAs also understood that Elnozahy's claimed "web server" comprising a "processor" is an example of a "server device." Elnozahy, [0017].

92. Thus, POSAs understood that Elnozahy's "server device," *i.e. computing system* (as I discussed in Section VII.D.1.a for [1Pre]), includes a *processor* under the agreed litigation construction.

c. [1B]

i. [1B.1] "b) an operating system having an operating system kernel…"

93. Elnozahy's "server device" ([0017]) *i.e. computing system* (as I discussed in Section VII.D.1.a), includes an "***operating system*** [OS]." Elnozahy, [0025], Fig. 2 (below).

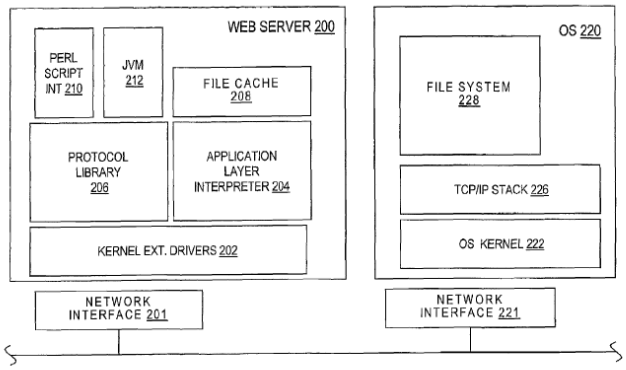


FIG. 2

94. In litigation, Google has argued that *operating system* should be construed as "the software that controls the allocation and usage of hardware resources such as memory, central processing unit (CPU) time, disk space, and peripheral devices"; VirtaMove has argued that no construction is necessary. EX1058, 4.

95. POSAs understood that Elnozahy's OS 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"

(EX1035, 378), consistent with Elnozahy's discussion of the "operating system" as including a "file system" that "retrieve[s] data from disk or other non-volatile storage facility" (Elnozahy, [0025]). Thus, Elnozahy's "operating system" meets Google's litigation construction.

96. In litigation, Google has argued that *kernel* should be construed as "the core of an operating system—the portion of the system that manages memory, files, and peripheral devices; maintains the time and date; launches applications; and allocates system resources." EX1058, 4. VirtaMove has argued no construction is necessary. *Id.*

97. Elnozahy's OS has a "kernel." Elnozahy, [0025]. POSAs understood that Elnozahy'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" (EX1035, 257), consistent with Elnozahy's description of the kernel as handling "processing of interrupts," starting and prioritizing application threads, and mapping hardware to applications. Elnozahy, [0020], [0027] ("The web server is assigned a high priority by the operating system kernel. The kernel then maps … the dedicated network interface to the web server.") Thus, Elnozahy's "kernel" meets Google's litigation construction.

98. Thus, Elnozahy's server device *comprises an operating system having an operating system kernel*.

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

99. 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.

100. In my opinion, the outer boundaries of what is "normally" supplied by an operating system, or what is "critical" to the operation of an application, are not well-defined to a POSA. However, the '058 patent says that examples of CSEs include "[n]etwork services including TCP/IP." EX1001, 6:12-13; *see also* EX1001, 3:22-30 (noting that a "kernel may provide a TCP/IP service" and describing this as an example of a CSE).

101. Elnozahy explains, consistent with a POSA's understanding, that in a "conventional web server architecture" such as shown in Figure 1, the "kernel…invoke[s]…network protocol routines indicated in FIG. 1 by the Transmission Control Protocol/Internet Protocol (TCP/IP) block 106," which Figure 1 shows as being part of the kernel. Elnozahy, [0005], Fig. 1 (below).

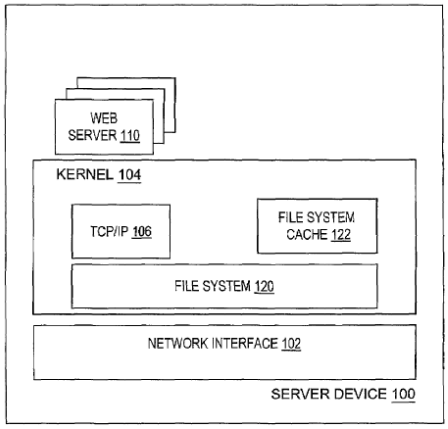


FIG. 1
PRIOR ART

102. The *kernel* in Elnozahy's operating system is a conventional kernel, as I discussed in Section VII.D.1.c.i for [1B.1]. Thus, POSAs would have understood Elnozahy to disclose an implementation of its *kernel* that includes a TCP/IP block as shown in Figure 1 (along with an alternate implementation where TCP/IP is implemented in a separate module from the kernel, as shown in Figure 2, which shows separate boxes for "TCP/IP Stack 226" and "OS Kernel 222").

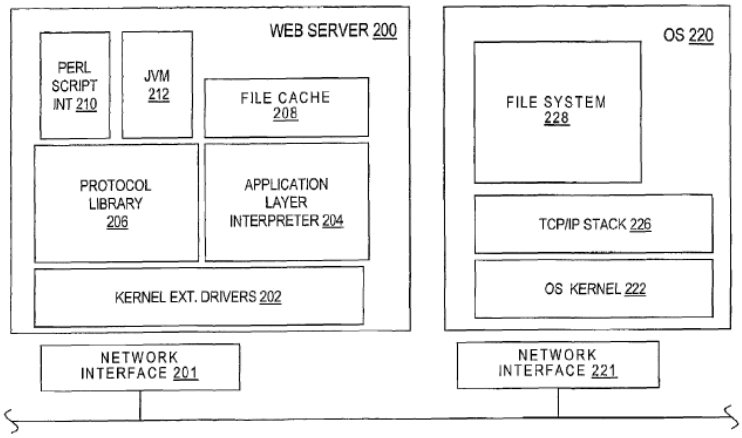


FIG. 2

103. At a minimum, POSAs would have found it obvious to implement Elnozahy+Draves this way, because this was the "conventional" implementation, which Elnozahy does not teach away from, and would have been within a POSA's skill. Elnozahy, [0005] (describing "conventional web server architecture").

104. In such an implementation, the *OS kernel has* the TCP/IP stack.

105. Elnozahy discloses that a TCP/IP block includes "routines" (plural) and "functions" (plural) including "extract[ing] an HTTP formatted request" from packets and "hand[ing] the extracted HTTP request to a web server application." Elnozahy, [0005]. Thus, while the full scope of "normally supplied" and "critical" in the '058 patent's definition of *CSE* (EX1001, 6:6-8) are not well-defined for POSAs, POSAs understood that the TCP/IP "routines" and "functions" in Elnozahy's TCP/IP block in the kernel are *CSEs*.

106. In litigation, VirtaMove has argued that the term *critical system element* should be construed as "any 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, the TCP/IP "routines" and "functions" in Elnozahy's TCP/IP block in the kernel are *CSEs* even if VirtaMove's litigation construction were accepted, since these functions/routines match examples of CSEs provided in the '058 patent. *See* EX1001, 6:12-13 ("[n]etwork services including TCP/IP"); EX1001, 3:22-30 (noting that a "kernel may provide a TCP/IP

service" and describing this as an example of a CSE). I note that in its claim construction briefing, VirtaMove cited "TCP/IP" and "network services" as examples of CSEs in the '058 patent. EX1067, 17.

107. 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 language does impose a claim requirement, Elnozahy+Draves meets this requirement. 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."). POSAs also understood that functions and routines are also sets of instructions, as I discussed in Section V.A. *See*, *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."); EX1035, 458 ("routine" and "function" are used as synonyms).

108. It was also known that dynamically loading functions/routines was one of two known options, *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"). POSAs would have been motivated to implement Elnozahy's TCP/IP block functions/routines as dynamic objects, and reasonably expected success, because dynamically loading objects was one of two known options, *i.e.* static and dynamic.

109. Finally, as I discussed above, Elnozahy's *OS kernel has* the TCP/IP stack. Therefore, the CSEs in the TCP/IP block in the OS are *OS critical system elements (OSCSEs)*.

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

110. In litigation, the parties have agreed to construe *kernel mode* as "[t]he context in which the kernel portion of an operating system executes. Application code cannot run in kernel mode." EX1058, 6.

111. Elnozahy refers in its "BACKGROUND" to a "user space" and "kernel space" that "identify conceptual spaces defined by the operating system that have different levels of protection." Elnozahy, [0007]. Elnozahy's "BACKGROUND" also refers to having to perform "***context*** switching" when "an application program operating in user space invokes a process that executes in the kernel space." Elnozahy, [0007]. Thus, POSAs understood that Elnozahy uses the term "kernel space" to refer to *the context in which the kernel portion of an operating system executes*, and would further have understood that *application code cannot run in* kernel space (because a context switch is required to invoke a kernel-space process), thus meeting the parties' agreed litigation construction of *kernel mode*.

112. At a minimum, in Elnozahy+Draves, POSAs would have found it obvious to implement Elnozahy's computing system with a "kernel mode" as disclosed in Draves, since Draves describes such a mode as "common." Draves, 1:23-39 (explaining that "[it] is common for a computer processor and associated operating system to have two different levels of resources and protection": one

being "privileged mode, system mode, or kernel mode" and the other being unprivileged mode). Draves further explains that this "common" "kernel mode" is distinct from the "user mode" that is used by "application programs." Draves, 1:25-28, 1:37-40. POSAs would reasonably have expected success in implementing Elnozahy+Draves with the "kernel mode" disclosed in Draves as it was "common for a computer processor and associated [OS]" to have kernel and user modes. Draves, 1:23-39.

113. Elnozahy says in its "BACKGROUND" section when discussing Figure 1 that the "kernel… invoke[s] one or more network protocol routines indicated in FIG. 1 by the [TCP/IP] block 106" that is inside "Kernel 104" in Figure 1 (below). Elnozahy, [0005]. I have reviewed Elnozahy, and I see nothing in Elnozahy that teaches away from this "BACKGROUND" implementation. Thus, in Elnozahy/Draves, a POSA would at least have found it obvious to implement Elnozahy's system so that the OS kernel invokes the TCP/IP block that is part of the OS kernel itself. Furthermore, Draves notes that "critical operating system components" "execute from," *i.e.* run from, "kernel mode" (also called "system mode"). Draves, 1:23-48. Thus, the OSCSEs, which are in the TCP/IP block as I discussed in Section VII.D.1.c.ii for [1B.2], are *for running in kernel mode.*

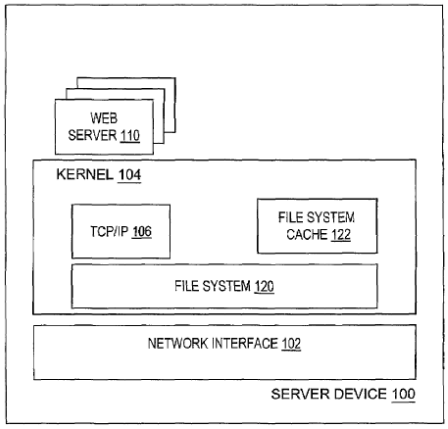


FIG. 1
PRIOR ART

114. Furthermore, POSAs understood that the OSCSEs run in kernel mode *using said processor* because running software such as a TCP/IP stack is the purpose of a processor.

115. Therefore, Elnozahy's OSCSEs are for running in kernel mode using said processor.

d. [1C]

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

116. 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 ("same physical memory space, that is, shared code space").

117. The '058 patent states that a "dynamic linked library (DLL)" (which the prior art also refers to as a "dynamic link library," *see* Draves, 3:57-61), 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, 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.”)[2]; 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 use the same DLL simultaneously, but under Windows 95 these applications are shielded from interfering with each other.”).

118. In Elnozahy+Draves, Elnozahy’s user-space “protocol ***library*** 206” (Elnozahy, [0022]) is implemented as a DLL, as Draves discloses (Draves, 3:57-61). *See* Section VII.C. Furthermore, Draves refers to a “DLL (dynamic link ***library***)” as an example of a “***shareable*** program module.” Draves, 3:57-61.

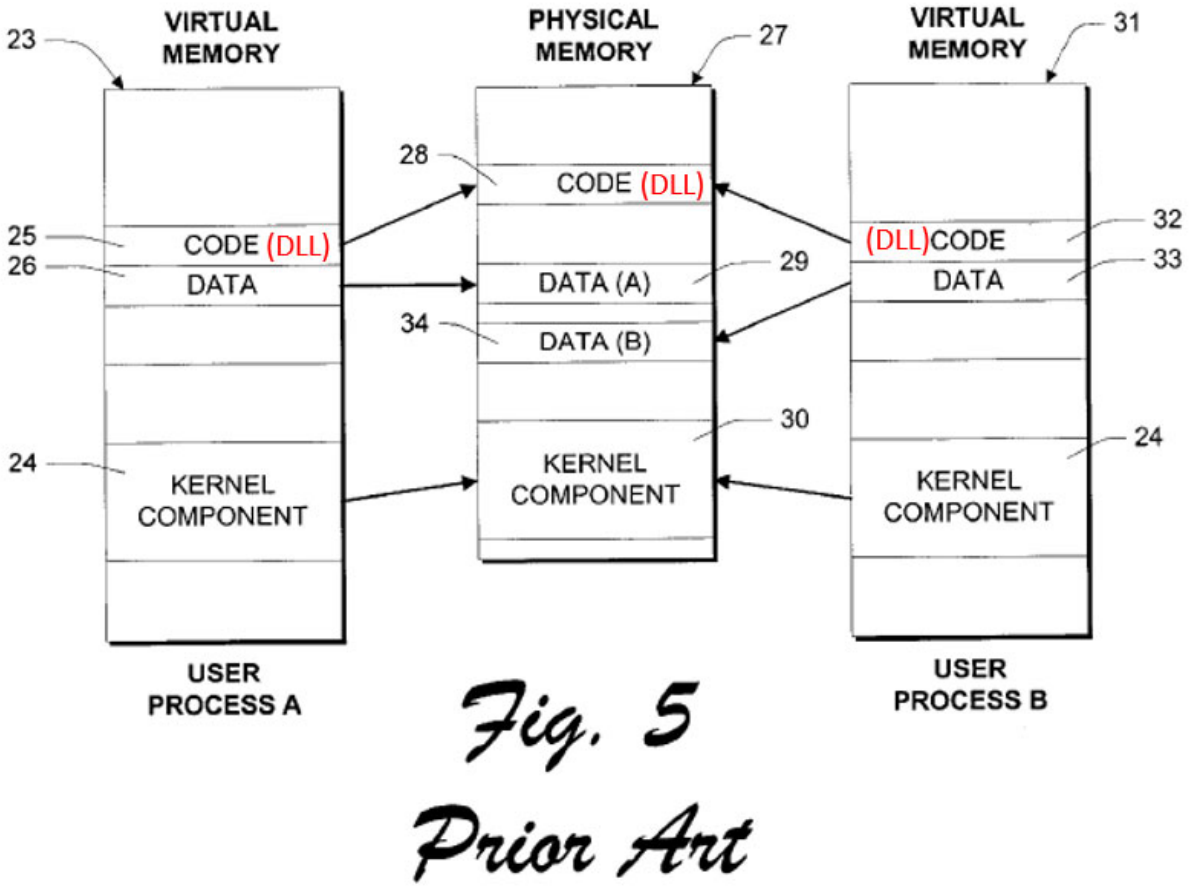
---

[2] 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.

Thus, in Elnozahy+Draves, the "protocol ***library***" implemented as a DLL is a *shared library*.

119. Furthermore, in an obvious implementation of Elnozahy+Draves, the protocol library implemented as a DLL satisfies the '058 patent's specification's above-quoted definition of *shared library*, which Google's litigation construction adopts (EX1059, 16). Draves describes "DLLs," which are "***shareable***," as residing in a single physical memory *address space*, *i.e. code space*, that is *different from that occupied by the kernel and its associated files* and that is mapped to a "code portion" in the "virtual ***address space***" of two different applications, and thus *accessible to multiple applications*. Draves, 2:28-31 (describing "typical approach" involving "allocat[ing] some portion of the memory to the operating system or kernel and another portion for application or user processes."), 3:52-4:32 (describing "code portions 25" and "data portion 26" of "DLL" as mapped to separate region of memory from "kernel," as shown in Fig. 5), Fig. 5 (annotated below). This is consistent with a POSA's understanding. *See*, *e.g.*, 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 use the

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



Draves, Fig. 5

120. The protocol library implemented as a DLL also meets VirtaMove’s litigation construction, which only requires “[a]n application library whose code space is shared among all user mode applications.” EX1068, 11. As I noted above, “code space” in the ’058 patent refers to physical memory space. EX1001, 3:39-43 (“same physical memory space, that is, shared code space”). POSAs understood

that the shared physical memory space into which the DLL is loaded is the DLL's code space, as VirtaMove itself has argued. *See* EX1067, 19 (VirtaMove arguing that "'code space' is a space occupied by code"). Furthermore, 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 Draves describes (*see* my discussion in the previous paragraph), the physical code space of the DLL is shared with and executable by all user-mode applications by being mapped into applications' virtual address space.

121. Because the "protocol library" includes "TCP/IP library routines that provide a user space TCP/IP protocol stack" (4:38-40), and because Elnozahy states "[p]rotocol library 206 may be implemented as a fully general TCP/IP library capable of handling all of protocol supported events and tasks" (Elnozahy, [0022]), POSAs understood Elnozahy to disclose that the protocol library has all of

the TCP/IP "routines" and "functions" that are in the operating system's TCP/IP block, and that are *CSEs* for the reasons I discussed in Section VII.D.1.c.ii above for [1B.2].

122. The "protocol library" containing the user-mode TCP/IP "routines" and "functions," which are *CSEs* as discussed above, is implemented as a DLL in Elnozahy+Draves. *See* Section VII.C above. POSAs understood that a library "***store[s]***" its contents. EX1035, 309 ("library" is "[i]n programming, a collection of routines stored in a file"); Jennings (EX1065), 1:46-49 ("Libraries allow a computer programmer to keep," *i.e.*, store, "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 these CSEs are in a shared library, i.e., the protocol library, these CSEs are *SLCSEs*. EX1001, 2:45-47 ("In accordance with this invention, SLCSEs are placed in shared libraries[.]").

123. Thus, Elnozahy+Draves's protocol library is a *shared library having SLCSEs stored therein*.

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

124. In litigation, the parties have agreed that "*user mode*" should be construed as "the context in which applications execute." EX1058, 6.

125. Elnozahy refers in its "BACKGROUND" to a "user space" and "kernel space" that "identify conceptual spaces defined by the operating system that have different levels of protection." Elnozahy, [0007]. Elnozahy's "BACKGROUND" also refers to having to perform "***context*** switching" when "an ***application*** program operating in user space invokes a process that executes in the kernel space." Elnozahy, [0007]. Thus, POSAs understood that Elnozahy uses the term "user space" to refer to *the context in which applications execute*, *i.e.*, *user mode* under the parties' agreed litigation construction.

126. The SLCSEs are in Elnozahy's "protocol library," which in Elnozahy+Draves is a DLL (*shared library*); Elnozahy, Abstract, [0008]; Section VII.D.1.d.i ([1C.1]), Section VII.C. Elnozahy says the protocol library "includes TCP/IP library routines that provide a ***user space*** TCP/IP protocol stack to web server 200." Elnozahy, [0022]. Furthermore, in Elnozahy+Draves, *the plurality of software applications* use the protocol stack provided by the protocol library. *See* Section VII.D.1.a ([1Pre]), VII.C. Thus, the SLCSEs in the protocol library 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…"

127. Elnozahy's "TCP library routines" and "functions" in the "protocol library" are *SLCSEs stored in the shared library*, as I discussed in Section VII.D.1.d.i for [1C.1]. 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. In my opinion, although the patent provides examples of what a "replica" could be, as I discuss below, the boundaries of the term *functional replica* are not well-defined to a POSA.

128. The specification provides an example of "***replication***" of an OSCSE using an SLCSE where TCP/IP services are provided by a shared library. *See* EX1001, 5:22-25. 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 kernel." EX1001, 5:22-25, 5:34-55.

129. Likewise, in Elnozahy+Draves, the *SLCSEs* are TCP/IP routines/functions are in the "protocol library" that is implemented as a shared library (a DLL) and that provides "a user space TCP/IP protocol stack" providing the same services as the operating system's TCP/IP block, as I discussed in VII.D.1.d.i for [1C.1]. Elnozahy, [0022].

130. Thus, although the full scope of "functional replicas" is not well-defined to POSAs, POSAs understood that Elnozahy's TCP/IP "routines" and "functions," which are the claimed *SLCSEs* (*see* Section VII.D.1.d.i ([1C.1])), qualify as *functional replicas* of their kernel-mode counterpart functions/routines, which are the claimed *OSCSEs* (*see* Section VII.D.1.c.ii ([1B.2])), because the relationship Elnozahy describes between the kernel-mode and user-mode TCP/IP routines/functions and their kernel-mode counterparts matches the SLCSE/OSCSE replication example in the '058 patent's specification.

131. Additionally, the "functions" and "routines" in the user-mode protocol library in Elnozahy+Draves perform the same operation as their kernel-mode counterparts. Thus, POSAs would have understood that the user-mode protocol library/functions have the same relevant attributes (*i.e.*, "characteristic or quality," EX1057, 41-42), namely the services provided, as their kernel-mode counterparts,

in the same manner as 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.D.1.e.iii below for [1D.3].

132. The “functions” and “routines” in the user-mode protocol library are also *functional replicas of OSCEs* even if VirtaMove’s litigation construction, “substantial functional equivalents or replacements of kernel functions” (EX1067, 22), were accepted, because in Elnozahy+Draves the SLCSE/OSCE relationship matches the example described in the patent, as I described above.

133. During prosecution, the applicants distinguished O’Rourke by arguing that a “functional replica” cannot be a mere “intermediary” to the kernel. EX1002, 323. POSAs understood that Elnozahy’s functions and routines in the user-mode protocol library (the *SLCSEs*) are not mere intermediaries. In characterizing O’Rourke as merely disclosing an “intermediary,” the applicants cited O’Rourke 6:12-19 and 10:12-33. These passages show that, in contrast to the protocol library of Elnozahy, which allows applications to bypass the kernel (Elnozahy, [0020]),

O'Rourke disclosed a "generic proxy object" available in user mode that did nothing but invoke kernel-level code. *See* O'Rourke, 6:12-19 ("…it is a primary object of this invention to provide user mode proxies for kernel mode filters. Other objects of the present invention include … providing a generic proxy object that may be used for virtually all kernel mode filters…"), 10:12-33 (describing various "prox[ies]" shown in Figure 3 as invoking "kernel mode" components).

134. Thus, Elnozahy's *SLCSEs stored in the shared library are functional replicas of OSCSEs* as claimed.

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

135. POSAs would have understood that the SLCSEs in Elnozahy+Draves are *accessible to some of the plurality of software applications*, or alternatively would have found it obvious to implement Elnozahy+Draves this way, as I explain below.

136. The *SLCSEs* are in a DLL used by *the plurality of software applications*, as I discussed in Section VII.D.1.d.ii for [1C.2]. POSAs understood that a purpose of a DLL is to make its contents *accessible* to the applications using the DLL. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*, 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 use the same DLL simultaneously, but under Windows 95 these applications are shielded from interfering with each other.").

137. In litigation, Google has argued that "*accessible*" should be construed as "wherein two or more of the plurality of the software applications can read SLCSEs stored in the shared library." EX1059, 17. POSAs understood that accessing the SCLSEs from the protocol library involved *reading* them, as I explain below.

138. The *SLCSEs* are the TCP/IP "functions" and "routine" in the user-space protocol library, as I explained in Section VII.D.1.d.i for [1C.1]. POSAs understood that a function/routine is a predefined set of instructions that carry out a specific action. *Supra* §V.A; EX1007, 1:22-29 (function is a predefined set of instructions that carry out a specific action); EX1035, 458 ("routine" and "function" are used as synonyms). POSAs also understood that executing the instructions that make up a function/routine involves *reading* those instructions. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*, 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.").

139. Thus, Elnozahy+Draves's SLCSEs meet Google's litigation construction of "*accessible*."

140. The SLCSEs also meet VirtaMove's litigation construction, which requires the SLCSE to be accessible to "some applications," and which only requires "use" by the applications. EX1067, 20. POSAs understood that by reading and executing the instructions making up the SCLSEs as discussed above, the applications *use* the SCLSEs. Furthermore, because the *plurality* of applications (at least) collectively use each SLCSE, *some applications use* each SLCSE.

141. 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"

142. 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").

143. The parties' agreed litigation construction of "forms a part of…" adopts the "linked to" requirement. EX1058, 6.

144. In Elnozahy+Draves, the SLCSEs are part of a DLL or "dynamic ***linked*** library," as Draves teaches. Draves, 3:58; Section VII.D.1.d.i ([1C.1]). POSAs therefore understood that when an application accesses an SLCSE, the SLCSE *forms a part of* that application because the DLL is linked to the application. Silberschatz (EX1018), 745 ("A DLL is 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"); Moberg (EX1033), 4:47-49 ("dynamic linking using DLLs provides a mechanism to link applications to libraries at run-time"); 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.").

145. 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* as claimed.

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…"

146. 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* (in object-oriented programming)" is "an object or duplicate object that has been created"); Microsoft (EX1035), 276 ("instance n." is "An object, in object-oriented programming, 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*.").

147. 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.

148. 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….").

149. As I explain below, POSAs understood that in an obvious implementation of Elnozahy+Draves, SLCSEs using the same technique as in the '058 patent, which POSAs understood would yield the same result.

150. The SLCSEs are in 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. Draves, 3:57-61; Section VII.D.1.d.i ([1C.1]); EX1001, 3:20-29. Furthermore, POSAs understood that an obvious way for applications to invoke the

functions/routines in the protocol library was to "call" them, just as the '058 patent's software applications "call" functions in the SLCSE library. Wilner (EX1069), 2:23-29 ("Under traditional operating system implementations, such re-usable code modules (called 'shared libraries')… can be implemented in the user space, and any task that desires access to a function of the shared library may be allowed to directly call the function."); EX1001, 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."). As the '058 patent itself admits, as I discussed above, the well-known consequence of calling a library's function is that *an instance of* each operation is *provided* to each application *from the shared library* and is *run in the context* of that application. EX1001, 5:22-25, 8:27-33, 9:15-20.

151. 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. The above-described code-sharing is "common practice," as the '058 patent notes; thus, as POSAs understood, this technique of code-sharing was conventional in the prior art.

152. POSAs understood that in Elnozahy+Draves, applications access SLCSEs in the same manner described in the '058 patent. SLCSEs are provided to applications in a DLL, as Draves teaches. *See* Section VII.D.1.d.i ([1C.1]).

153. Draves also teaches, consistent with a POSA's background knowledge, that different applications that use a DLL share the code representing the DLL but use their own data areas in memory when running the code. Draves, 4:16-33 & Fig. 5 (showing two different programs A and B having their own data

areas when accessing a DLL); Petzold (EX1015), 965-966 ("Each process has its own address space for any data the DLL uses."); Silberschatz (EX1018), 745 ("A DLL is 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.").

154. At a minimum, POSAs would have found it obvious to implement Elnozahy+Draves such that applications share the code for operations but use application-specific data areas when running 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 use the same operation; furthermore, POSAs would reasonably have expected success with such an implementation because it was common practice. EX1001, 3:32-44 ("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…."); Petzold (EX1015), 966 ("Each process has its own address space for any data the DLL uses."); Silberschatz (EX1018), 745 ("A DLL is 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.").

155. Thus, in Elnozahy+Draves, *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…"

156. POSAs understood that the *plurality of software applications run under the operating system*, or at least would have found it obvious to implement Elnozahy+Draves this way, since this was the standard relationship between applications and OSs. Calusinski (EX1037), [0003] ("Computer systems typically include… one or more software application programs that run under the control of the operating system.")

157. In Elnozahy+Draves, *SLCSEs* are in a DLL, which are shared libraries, as discussed in Section VII.D.1.d.i ([1C.1]). Because DLLs are shared libraries, POSAs understood that multiple applications running under the computer's OS can call the same operation in a DLL, *i.e.*, that multiple applications *have use of* the same operation 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 ("Multiple applications can use the same DLL simultaneously").

158. 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.

159. 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.

160. In Elnozahy+Draves, each application is linked to the “user mode library” containing the SLCSEs, as I explained in VII.D.1.e.iii for [1D.3]. Furthermore, each application has its own unique data space in memory, as I discussed in VII.D.1.f.i. Thus, in Elnozahy+Draves, *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."

161. In Elnozahy+Draves, the "functions"/"routines" in the protocol library are the claimed *SLCSEs*, as I discussed in Section VII.D.1.d.i for [1C.1]. POSAs would at minimum have found it obvious to implement Elnozahy+Draves so that applications that invoke these functions/routines are written in such a way as to reference them using a *predetermined* identifier, such as a function/routine name, as this was a known and typical way to invoke functions/routines from libraries that 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 relates to a *predetermined function*.

162. 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. *See* Sections VII.D.1.d.i-VII.D.1.d.ii ([1C.1]-[1C.2]). Furthermore, the SLCSEs are provided to the applications *for running instances of* a SLCSE. Additionally, the SLCSEs are in a DLL, and when an application invokes an SLCSE from the DLL, the application gets an *instance* of the SLCSE that *runs* in that application's context. *See* Section VII.D.1.f.i ([1E.1]).

163. 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, Elnozahy+Draves'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***").

164. Additionally, it was known to implement computing systems such as Elnozahy's "server device" (Elnozahy, [0015]) with multiple processors that could each execute at the same time. POSAs would have been motivated to implement the server device in Elnozahy+Draves 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.*, TCP/IP applications using the user-space TCP/IP protocol library (*see* Section VII.D.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 Elnozahy+Draves 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"

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

166. If VirtaMove argues that "*multiple*" requires more than two instances, this would have been obvious. POSAs understood that a single computer system could run three or more applications that use TCP/IP. *See* Guenther (EX1074), [0057] (example "TCP/IP based applications" include email, news, and terminal emulation). POSAs would have been motivated to run at least three different TCP/IP 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 Elnozahy itself discloses running multiple applications. Elnozahy, [0017]; Section VII.A.

167. 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").

168. 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.

169. In Elnozahy+Draves, the first and second applications are provided with their own unique instances of an SLCSE, as I discussed in Sections VII.D.1.f.i-VII.D.1.f.ii for [1E]. POSAs understood that in Elnozahy+Draves, the applications run within the OS because Elnozahy teaches that applications run in the kernel "context," and the kernel is part of the OS, as I discussed above for [1B.3] and [1B.1]. *See also* Silberschatz (EX1018), 3 ("An ***operating system***…provides a basis for application programs[.]"), 5 ("An [OS]… manages the execution of user programs[.]").

170. Thus, in Elnozahy+Draves's computer system, *in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system*.

171. Additionally, Elnozahy+Draves's *SLCSEs* replicate the functionality of CSEs in the OS (*i.e.*, OSCSEs), as I discussed in Section VII.D.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.D.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”

172. In an obvious implementation of Elnozahy+Draves, OSCSEs are part of the TCP/IP stack that is inside the kernel, as I discussed in Section VII.D.1.c.ii for [1B.2]. Thus, the OSCSEs *remain in the operating system kernel*.

173. Furthermore, these OSCEs *correspond to* and are *capable of performing the same function as* the SLCSEs, because each SLCSE performs the same TCP/IP routine/function as an OSCSE, as I discussed in Section VII.D.1.e.i for [1D.1].

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…”

174. 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.

175. In Elnozahy+Draves, the *SLCSEs* are in a "DLL" that the *applications* "link[]" to, as I discussed in Section VII.D.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.D.1.f.i for [1E.1]. Additionally, when Elnozahy+Draves'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.D.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 Elnozahy+Draves, 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.

176. 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 execut**able** SLCSEs (as opposed to an execut**ing** 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, and if claim 4 were interpreted this way, this would have been an obvious implementation of Elnozahy+Draves. 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. 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.

177. 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 Elnozahy+Draves. 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"

178. 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, in Elnozahy+Draves, *one or more SCLSEs use system calls to access services in the [OS] kernel*.

i. File Retrieval

179. Elnozahy's "Web server 200" includes an "HTTP interpreter" that can respond to received TCIP/IP requests by providing an "HTML file." Elnozahy, Abstract, [0028]-[0029]. One action the HTTP interpreter performs is to retrieve the HTML file that is to be provided. The HTTP interpreter retrieves the file to be provided in one of two ways. If the file is in the "file cache," the interpreter retrieves the file from the cache. Elnozahy, [0029]. POSAs understood that the "file cache" is a user-mode, since it is part of the web server, which is a "user space" application. Elnozahy, [0008]. If the file is not in the user-mode file cache,

the server "invoke[s] the operating system file system…to retrieve the [file] from disk." Elnozahy, [0029] ("Depending upon whether the requested file is in the file cache, the document is retrieved from disk (block 418) using the operating system's file system or from the cache (block 419)."), [0025] ("When non-cached data is accessed, Web server 200 will invoke the operating system file system 228 to retrieve the data from disk or other non-volatile storage facility.").

180. The '058 patent says examples of CSEs include "[n]etwork services including TCP/IP, Bluetooth, ATM; or message passing protocols." EX1001, 6:6-13. Examples of CSEs also include "File System services that offer extensions to those supplied by the OS" such as "[a]ccess[ing] files that reside in different locations as though they were all in a single locality" and "[a]ccess[ing] files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format." EX1001, 6:14-20. Examples of CSEs also include "[i]mplementation of file system optimizations for specific application behavior." EX1001, 6:20-22.

181. POSAs understood that Elnozahy's above-discussed feature of retrieving an HTML file from either the server's file cache or a disk matches examples of CSEs in the '058 patent, including "[a]ccess[ing] files that reside in different locations as though they were all in a single locality" and "[i]mplementation of file system optimizations for specific application

behavior." EX1001, 6:14-22. Furthermore, POSAs understood that the file-retrieval feature is provided in user mode, because Elnozahy describes retrieval as a task performed by the web server (Elnozahy, [0025]) and also describes the web server as a "user space" application (Elnozahy, [0008]). *See also* my discussion of [1C.2] in Section VII.D.1.d.ii.

182. Thus, POSAs would have found it obvious to implement this feature as a function/routine in a shared library such as a DLL for the same reasons of reusability that I discussed in Section VII.D.1.d.i above for Element [1C.1]. Additionally, the file-retrieval functionality has the same attribute of the service provided (*i.e.*, obtaining a file) as the corresponding operating system functionality discussed below that retrieves a file by reading it from disk, and is thus a functional replica of OS feature, *i.e.*, of that *OSCSE*. In such an implementation, where the file-retrieval feature is implemented in a DLL linked to an application and is a functional replica a corresponding operating system feature, the file-retrieval feature is an *SLCSE* for the reasons I discussed above for claim 1 with respect to functions/routines in the TCP/IP library. Moreover, because this SLCSE is in a *DLL*, it is also *provided to one of the plurality of software applications having exclusive use thereof* for the same reasons I discussed in Section VII.D.4.a ([4A]) with respect to SLCSEs in the user-mode protocol library that is also a DLL.

183. Elnozahy discloses that user-level processes can use "kernel calls" to invoke the kernel; these kernel calls are thus system calls. Elnozahy, [0020]. Elnozahy's Figure 1, which is discussed in the "BACKGROUND" and illustrates "selected features of conventional web server architecture" (Elnozahy, [0005]), shows the "File System" as part of the kernel. Elnozahy does not teach away from this "BACKGROUND" implementation; thus, in Elnozahy+Draves, a POSA would at least have found it obvious to implement Elnozahy's system so that the file system is part of the OS kernel itself (*see* my discussion of Element [1B.2] in Section VII.D.1.c.ii). POSAs would further at a minimum have found it obvious to implement Elnozahy's above-discussed file-retrieval feature, which is an *SLCSE* in obvious implementations of Elnozahy+Draves, to use a "kernel call," *i.e.* a *system call* as I discussed above, to invoke the file system to read a file in the case when the desired file is not in the web server's file cache. Using system calls was the customary way to invoke kernel services, and POSAs would have reasonably expected success in using this customary technique. Silberschatz (EX1018), 463. In such an implementation, the file-retrieval feature, which is an *SLCSE*, *uses system calls* to access the file system, *i.e.*, a *service in the operating system kernel*.

ii. Initializing interrupt handling

184. The '058 patent provides an example where "a critical system element in the context of an application program," *i.e.* an *SLCSE*, "uses system calls to

access services in the kernel module" by relying on "[i]nterrupt handling…initialized through a system call" to enable the "kernel module" to "notif[y]" the SLCSE about an "interrupt":

> FIG. 5 represents the function of the kernel module 58, described in more detail below. **A *critical system element* in the context of an application program *uses system calls* to access services in the kernel module**. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. **A service in the context of an application is notified of an interrupt by the kernel module**.
>
> FIG. 6 represents interrupt handling. **Interrupt handling is initialized through a *system call***. A handler contained in the kernel module 58 is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The **handler notifies a service in the context of an application** through the use of an up call mechanism.

EX1001, 8:45-59.

185. Elnozahy discloses that in one embodiment, its web server "replac[es] conventional interrupt driven processing with a polling architecture in which the server periodically monitors the network interface for new requests." Elnozahy, [0008]. However, Elnozahy does not **require** the use of a polling architecture. *See, e.g.*, Elnozahy, claims 1, 9 (independent claims not requiring polling).

186. Furthermore, POSAs understood that the benefits Elnozahy discloses from performing TCP/IP processing in user mode, such as reduced context switching and the ability to optimize TCP/IP processing for specific applications (see Elnozahy, [0006]-[0007]), would accrue even if an interrupt-based architecture were used. Moreover, POSAs understood using interrupts v. polling involved a tradeoff; polling might reduce latency (the overall time between the start and end of an operation), at the possible expense of wasting processor resources. EX1048, 1; EX1049, 282. Similarly, Elnozahy discloses that when the polling mechanism is used, the web server is "given a high priority compared to other processes," which POSAs recognized might not be optimal in all circumstances (*e.g.*, when the computing system on which the web server application runs also runs other applications). Elnozahy, [0017].

187. Thus, in Elnozahy+Draves, POSAs would have been motivated to alternatively implement the TCP/IP functions/routines in the protocol library, i.e. the SLCSEs (as I discussed in Section VII.D.1.d.i for [1C.1]), to use interrupts rather than polling for a particular one of the software applications where minimizing latency is not required or where giving higher priority to the application using the SLCSE is not desirable.

188. POSAs would have reasonably expected success at such an implementation because interrupt-based processing was "conventional," as Elnozahy notes (Elnozahy, [0005]).

189. POSAs understood that interrupt handling was conventionally a kernel responsibility; thus, initializing interrupt handling is a *service in the OS kernel*. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Lever (EX1044), [0042]. Thus, in the above-described Elnozahy+Draves implementation, POSAs would have understood, or at least found obvious, that the user-space TCP/IP library functions/routines, *i.e.* the *SLCSEs*, rely on the kernel to initialize interrupt handling. Furthermore, POSAs would have found it obvious to implement Elnozahy+Draves such that interrupt handling is initialized through a "kernel call" (Elnozahy, [0020]), *i.e.* a *system call*, just as the '058 patent describes (EX1001, 8:45-59), as this is how Elnozahy discloses accessing kernel services (as I discussed in Section VII.D.4.b.i above).

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

190. 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.

191. The ’058 patent does not define “*interface*,” but it describes an embodiment where a “kernel module” acts as a “***device interface***” that “***enables*** data exchange between a user mode CSE and a ***device driver*** in kernel mode.” EX1001, 9:60-10:20. The patent then notes that “services exported for ***device interface*** typically include” “[i]nitialization” and “[e]stablish[ing] a channel between a CSE in user mode and a specific device.” EX1001, 9:66-10:20, Fig. 5 (below). These examples are consistent with how POSAs understood “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”).

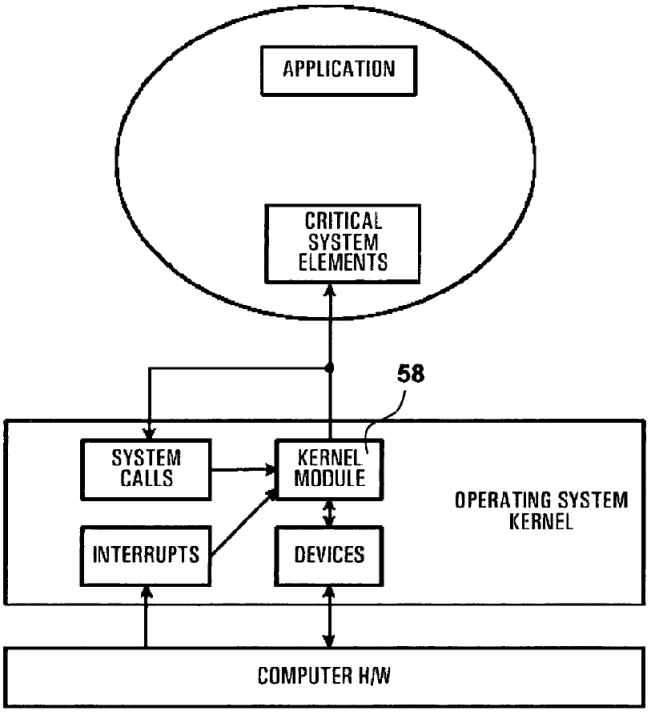


FIG. 5

192. Elnozahy+Draves meets claim 5 in three alternative ways, as I explain below.

a. Initializing Communication with Network-Interface Drivers

193. Elnozahy discloses "kernel extension device ***drivers…*** configured to poll network interface 201 periodically to determine when new client requests have arrived." Elnozahy, [0021]. "[I]f polling determines that a network packet has been received, network processing of the packet is performed by" the "user space TCP/IP protocol library." Elnozahy, [0028].

194. Elnozahy explains that the kernel "maps… the dedicated network interface to the web server" by allocating "memory" for "creating a shared buffer" accessible to both. Elnozahy, [0027]-[0028]. POSAs would have found it obvious to likewise have the kernel create a "shared buffer" (Elnozahy, [0027]) for communication between the TCP/IP library and the kernel extension drivers, since Elnozahy discloses this capability.

195. Furthermore, POSAs would at least have found it obvious to implement such memory allocation in a *kernel module*, *i.e.* a "set of functions that reside and execute in kernel mode as extensions to the [OS] kernel" under the agreed litigation construction (EX1001, 6:56-59; EX1058, 6). Implementing such functionality as part of a kernel extension was a known technique and therefore would have been within a POSA's skill. Kush (EX1102), 5:49-51 ("The kernel

extension 30 implements…messaging queues, [and] memory pools…"); Garney (EX1099), 7:47-51; Temple (EX1100), [0053]; Kubala (EX1101), [0115]. And POSAs understood that using this technique would have for implementing memory mapping would have beneficially enabled Elnozahy's described functionality.

196. In such an obvious implementation, the *kernel module* providing the memory allocation service *serves as an interface between an SLCSE* in the "user-space protocol library" that "processes" the packet (Elnozahy, [0028]) and *a device driver* (the kernel-extension driver), by "enabl[ing] data exchange between," and "[e]stablish[ing] a channel between," the SLCSE and the driver, as the '058 patent's describes. EX1001, 9:60-10:20. This SLCSE, like other SLCSEs, *runs in the context of an application* that invokes it (e.g., the web-server application), as I explained in Section VII.D.1.f.i for [1E.1].

b. File retrieval

197. Elnozahy's feature of retrieving an HTML file from either the file cache or from disk is a *SLCSE*; when the file is to be read from disk, the file-reading functionality uses a system call to access the file system in the kernel, as I discussed in Section VII.D.4.b.ifor [4B]. POSAs understood that it was common to access physical storage such as a disk using a driver. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*, Cantwell (EX1080), 1:15-17 ("Typical hardware devices which require device drivers include printers,

scanners, and storage devices.”). Thus, POSAs would at minimum have found it obvious to implement Elnozahy+Draves so the file system uses a driver to read the file from disk; this was customary as discussed above, and thus POSAs would have reasonably expected success with such an implementation.

198. Elnozahy discloses that the file system is part of the kernel (as I discussed in Section VII.D.4.b.i) but does not provide details of how this arrangement is implemented. However, POSAs understood that it was customary to incorporate a set of functions for interfacing with the file system into the kernel by making that set of functions an *extension* of the kernel, since there are different types of file systems that must each be implemented differently. My testimony regarding a POSA’s background knowledge is corroborated by, *e.g.*: Correnti (EX1081), 5:27-31 (“Kernel extension mechanisms are well known in the art and are typically utilized to dynamically load device drivers and file systems within an [OS] kernel.”); McNabb (EX1082), 3:45-62 (describing use of kernel extension for “multimedia file system (MMFS)”); Hartner (EX1083), 3:58-63 (referring to POSA’s understanding that an “installable file system” is a type of “[OS] kernel extension”).

199. POSAs also understood that a file system is interfaced using a set of functions including functions to allow communication with the disk driver for tasks such as reading from or writing to the disk. My testimony regarding a POSA’s

background knowledge is corroborated by, *e.g.*: Microsoft (EX1035), 213 ("file system" refers to "portion of an operating system that translates requests for file operations from an application… into… tasks that can be understood by the drivers"); Karamanolis (EX1084), 3:32-37 ("The upper layers of the standard NFS [network file system] client protocol… implement the generic file system operations, such as read (), write (), open(), etc."); Cabrera (EX1085), 14:4-6. Thus, POSAs would at minimum have found it obvious to implement Elnozahy's file system interface as a kernel extension providing a set of functions including functions for communicating with the disk driver; this was the standard implementation, as discussed above, and POSAs would thus reasonably have expected success.

200. In such an implementation, the file system's interface (which is part of the kernel) is a set of functions that reside and execute in kernel mode as extensions to the [OS] kernel, *i.e.* a *kernel module*, that *serves as an interface between the SLCSE* (the file-reading functionality) and *a device driver* (the disk driver). This SLCSE, like other SLCSEs, *runs in the context of an application* that invokes it (e.g., the web server application), as I explained in Section VII.D.1.f.i above for [1E.1].

c. Interrupt setup

201. As I discussed in Section VII.D.4.b.ii, in an alternate obvious implementation of Elnozahy+Draves, the user-mode TCP/IP functions use interrupts to be notified of a packet's arrival. In such an implementation, "network processing of the packet is performed by the server's user space TCP/IP protocol library" (Elnozahy, [0028]) in response to an interrupt, rather than based on polling. POSAs would have found it obvious to implement interrupt handling, including setup and notification, using a *kernel module*, *i.e.* a "[a] set of functions that reside and execute in kernel mode as extensions to the [OS] kernel" (EX1001, 6:56-59; EX1058, 6). POSAS understood that implementing such functionality in the kernel as part of a kernel extension was a known technique that was within a POSA's skill. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Feigenbaum (EX1105), 5:38-41 ("Interrupt manager 34 preferably manages interrupt handlers which are set up by the operating system kernel 12 and kernel extensions to take whatever action is necessary to respond to the occurrence of an interrupt."); Garrigues (EX1106), [0037] ("In the example illustrated, it consists of inserting these new services into the table. This function can include six parameters, as illustrated, which serve as **interfaces between the basic kernel and the interrupt managing extension module**."). Furthermore, using this technique would have beneficially enabled interrupt-based processing.

202. In such an implementation, the *kernel module* providing the interrupt services *serves as an interface between an SLCSE* in the "user-space protocol library" that "processes" the packet (Elnozahy, [0028]) and *a device driver* (the kernel-extension driver) by "enabl[ing] data exchange between," and "[e]stablish[ing] a channel between," the SLCSE and the driver, as in the '058 patent's examples. EX1001, 9:60-10:20. This SLCSE, like other SLCSEs, *runs in the context of an application* that invokes it (e.g., the web server application), as I explained in Section VII.D.1.f.i above for [1E.1].

6. Claim 6:

a. [6A]: "A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program…"

i. File retrieval

203. In Elnozahy+Draves, the "file system" in the kernel, which meets the claimed *kernel module*, is used to read a file from the disk, I discussed in Section VII.D.5 for claim 5. POSAs understood that the typical and customary way to read a file from a disk involved issuing a request to the driver for the file and then waiting for an interrupt that indicates the file has been retrieved. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Narayanswami (EX1086), 1:12-25 ("The operating system (OS) of a computer is constantly sending read and write commands to a disk driver which in turn sends

the commands to the disk… For each command being sent to the driver, an interrupt is generated to notify the central processing unit (CPU) it is done processing the command…"); Griffin (EX1043), 1:12-21 ("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. The handling of these kinds of asynchronous events has traditionally been accomplished in one of two ways, including polling and interrupt-driven.").

204. Thus, in Elnozahy+Draves, POSAs would have at minimum found it obvious to implement the file system to read files by waiting for an interrupt; this was typical, as discussed above, and thus would have been within a POSA's skill to implement.

205. Furthermore, since the file system itself waits for an interrupt when reading a file, POSAs would have found it obvious to implement the file-reading functionality, which meets claim 5's *SLCSE* and *runs in the context of an application program* (as I explained in Section VII.D.5 for claim 5), to wait for *notification* of the *event* of the completed reading of the file *provided by* the file system. POSAs understood that it was logical for the file-retrieval functionality to wait for notification given that the file system itself would be waiting for an

interrupt, and it would have been within a POSA's skill to implement such functionality using well-known programming techniques.

ii. Packet arrival

206. In the interrupt-based implementation of Elnozahy+Draves I discussed in Section VII.D.5.c, the functions/routines in the TCP/IP library, which are *SLCSEs* that *run in the context of the application program*, wait for *notification* of the *event* of packet arrive *provided by* the interrupt handling *kernel module*.

b. [6B]: "wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application

i. File retrieval

207. As I discussed in section VII.D.6.a, the *event* is the completion of reading the file. POSAs understood that an "asynchronous" event is something that can occur any time. Microsoft (EX1035), 38 ("asynchronous" defined as "Pertaining to, being, or characteristic of something that is not dependent on timing"). POSAs understood that the completion of reading a file from disk is an *asynchronous event*, because the completion time is not deterministic (and hence the time at which completion occurs is not known in advance). My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Griffin (EX1043), 1:12-21; 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.").

208. POSAs also understood that this event *requires information to be passed to the SLCSE from outside the application*. The event requires the SLCSE to be notified that the file system has finished reading a file as I discussed in Section VII.D.4.b.i for [4B], and the file system is in the kernel and thus *outside* the web server application.

ii. Packet Arrival

209. POSAs understood that packet arrival is an *asynchronous event*, *i.e.*, one that may occur at any time (Microsoft (EX1035), 38). ; EX1043, 1:12-17; EX1035, 38. 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."). POSAs also understood that this event *requires information to be passed to* the user-mode TCP/IP library function, which is *an SLCSE* that *runs in the context of an application program* (as I discussed in Section VII.D.1.f.i), *from outside the*

application, for two reasons: (1) the event requires the SLCSE to be notified of the packet's arrival at the network interface, which is *outside* the application; and (2) in response to the event, the SLCSE processes the packet, which arrived from outside the application. Elnozahy, [0020].

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."

210. POSAs understood that "*notifying the SLCSE*" includes notifying about the *event* recited in claim 6 (from which claim 7 depends).

211. In Elnozahy+Draves, the *event* of claim 6 may be (i) the competition of reading a file by the file system (*see* SectionVII.D.6.b.i); or (ii) the arrival of a packet (*see* Section VII.D.6.b.ii). In either case, the relevant SCLSE is notified of the event because of an interrupt, as I discussed in Sections VII.D.6.b.i and VII.D.6.b.ii. POSAs understood that it was customary to manage interrupts using an interrupt "*handler*." Ross (EX1087), 1:33-46 (discussing prior art interrupt handlers). POSAs would thus have found it obvious to implement Elnozahy+Draves to include an interrupt handler that processes interrupts, and would reasonably have expected success in using this known technique.

212. POSAs would also have found it obvious to implement the interrupt handler to notify the relevant SLCSE—*i.e.*, the file-retrieval feature (as I discussed in Section VII.D.6.b.i for file reading) or a user-space TCP/IP library function (as I

discussed in Section VII.D.6.b.ii for packet arrival)— 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.

213. Specifically, POSAs would have implemented the interrupt handler to invoke the file-reading functionality using an upcall once the file had been read from disk, and to invoke the user-space library packet processing function using an upcall once the packet arrives. Using an upcall to inform user-mode processes about interrupts was a known technique that would have beneficially signaled the relevant SLCSE without the need to switch from kernel mode (where the interrupt handler runs) to user mode, and POSAs would have reasonably expected success using this known technique. 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."

214. In obvious implementations of Elnozahy+Draves, an interrupt handler invokes a user-mode *SLCSE* using an upcall, as I discussed in Section VII.D.7 for claim 7. The interrupt handler is part of the kernel, as I discussed in Section VII.D.4.b.ii for [4B] and in Section VII.D.7 for claim 7. Draves discloses, consistent with a POSA's background knowledge, that kernel-mode processes have access to all address space, including user-mode address space. Draves, 1:47-49

("The kernel may access both the kernel system memory and the user process system memory.").

215. Thus, POSAs understood that the interrupt handler could access SLCSE instructions *resident in user mode space* and *execute* those *instructions* while remaining in *kernel mode*. POSAs would have been motivated to implement the interrupt handler this way because POSAs understood that having the kernel-mode interrupt handler execute the SLSCE instructions beneficially avoided the performance impact of switching to user mode. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Rao (EX1088), [0002] ("One difficulty associated with providing video data in digital format is the performance-degrading overhead incurred when a media server sends packetized audio/video data to an end-player or user. Some of the main sources of overhead associated with a media server transmitting packetized audio/video data are (1) copying data from the user space buffer to the kernel buffer and (2) context switching from user level to kernel level."); Slavenburg (EX1098), 1:29-32 ("In designing processors, there are a number of cost/performance tradeoffs. Higher performance often comes at the expense of interrupt overhead, interrupt latency, and context switch degradations.").

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."

216. Elnozahy+Draves's TCP/IP functions/routines in the user-mode protocol library *CSEs* because they supply services normally supplied by the operating system, as I discussed in Section VII.D.1.d.i for [1C.1]. Furthermore, the SLCSEs are in a user-mode library linked to the applications, as I discussed in Section VII.D.1.f.i for [1E.1]. Thus, Elnozahy-Draves's user-mode TCP/IP functions/routines *provide one of the plurality of software applications access to operating system services*.

217. 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.

218. The *SLCSEs* include "functions" in the user-mode protocol library, as I discussed in Section VII.D.1.d.i for [1C.1]. POSAs understood that these functions are "replace[ments]" for "function[s]" in the OS "that would normally be called" were the user mode library not available (EX1001, 8:62-64) because these functions were normally provided by the OS, as I discussed in Section VII.D.1.c.ii for 1B.2. Thus, in Elnozahy+Draves, *a function overlay is used to provide one of*

the plurality of software applications access to operating system services because the sending and receiving functions normally supplied by the OS are replaced by user-mode functions.

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."

219. 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.

220. In Elnozahy+Draves, the *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.D.1.e.iii for [1D.3]. Furthermore, because the SLCSEs are part of a dynamic ***linked*** 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.D.1.f.ii for [1E.2].

221. POSAs would have found it obvious to implement Elnozahy+Draves 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, 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."

222. Elnozahy+Draves's *SLCSEs* (the user-mode sending and receiving functions, as I explain in Section VII.D.1.d.i for [1C.1]), both *utilize kernel services supplied by the [OS] kernel* for the purposes recited in claim 11, as I explain below

a. Device access

223. Elnozahy's functionality of reading an HTML file from either the file cache or from disk is an *SLCSE*; when the file is to be read from disk, the file-

reading functionality uses a system call to access the file system in the kernel, as I discussed in Section VII.D.4.b.i for [4B]. POSAs understood a disk a *device*; thus, the file-reading functionality *utilizes* a *kernel service supplied by the operating system kernel*, *i.e.* the file system, *for device access*.

b. Interrupt Delivery

224. As I discussed in Section VII.D.6 for claim 6, in obvious implementations of Elnozahy+Draves, *SLCSEs* are notified by the interrupt handler of the interrupt that occurs when the file system has finished reading a file from the disk. Thus, thus, the *SLCSEs utilize kernel services supplied by the operating system kernel*, *i.e.* the file system, *for interrupt delivery*.

c. Virtual Memory Mapping

225. Draves discloses, consistent with a POSA's background knowledge, that virtual memory mapping between virtual and physical addresses is a standard OS kernel service. Draves, 1:48-60 (explaining in "BACKGROUND" that "The kernel is responsible for supervising the virtual memory system…. Each time a process uses a virtual memory address, the virtual memory system translates it into a physical address using a virtual-to-physical address ***mapping***."). Additionally, Draves discloses, consistent with a POSA's background knowledge, that different applications that linked to a DLL would have the library in their "virtual address space." Draves, 3:52-4:32, Fig. 5 (annotated below) (describing a prior art

"sharable DLL" located in a region of physical memory mapped to a "code portion" in the "virtual address space" of two different applications).

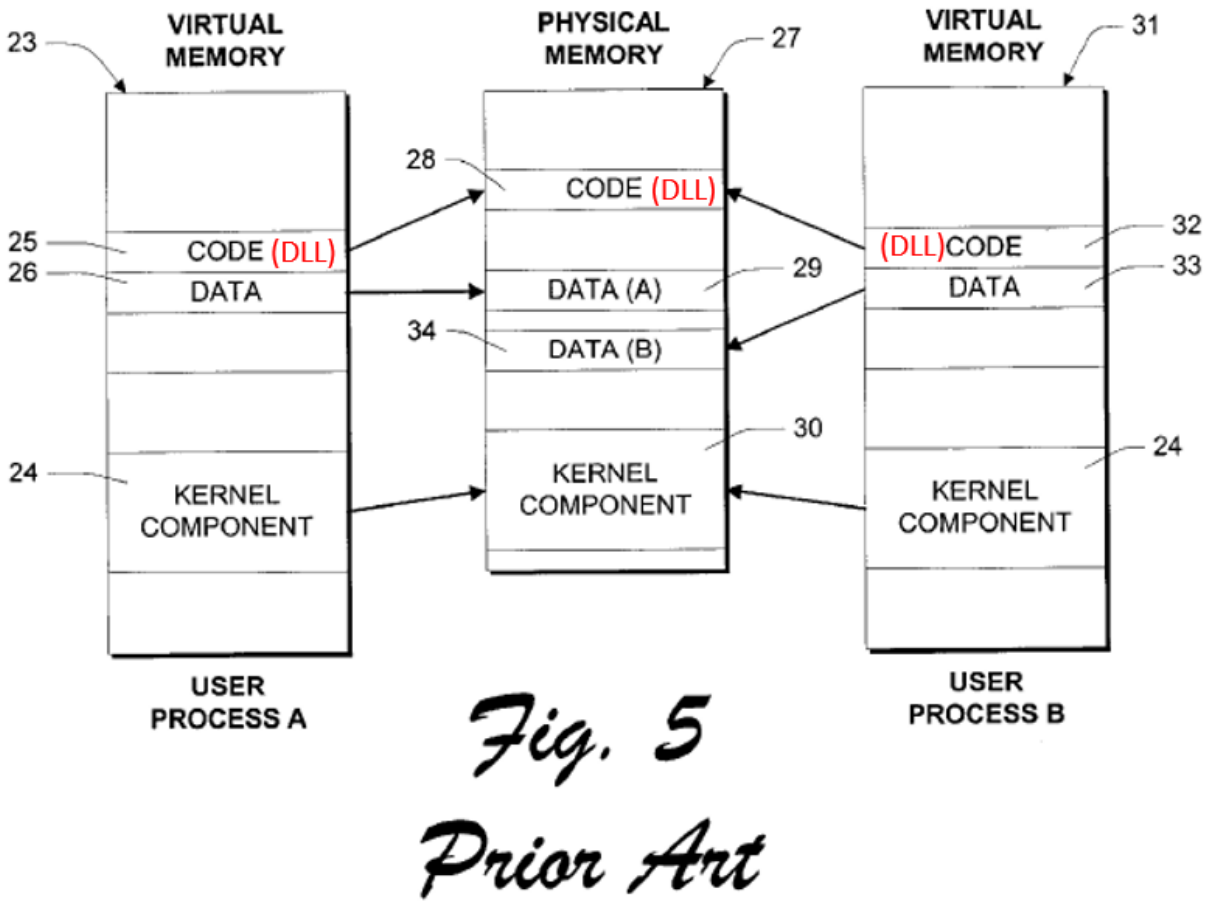


Fig. 5
Prior Art

226. Furthermore, as I discussed in Section VII.D.5.a for claim 5 (*e.g.*, paragraph 194), Elnozahy teaches that the kernel "maps" the "network interface" to the "web server," a process which includes "the kernel creating a shared buffer between the server's memory address space and the network device interface

buffers in the user-mode." Elnozany, [0027]-[0028]. POSAs understood that this type of "mapping" involved using virtual memory mapping.

227. Thus, POSAs would have found it obvious to implement Elnozahy+Draves so that the *SLCSEs*, which are in a DLL (as I discussed in Section VII.D.1.d.i for [1C.1]), rely on virtual memory mapping services supplied by the kernel to achieve the known benefits of virtual memory, such as "isolating processes from each other" (Draves, 1:50-56); POSAs would reasonably have expected success since using virtual memory was well-known, as discussed above.

228. In such an implementation, the SLCSEs *utilize a kernel service supplied by the operating system kernel for virtual memory mapping*.

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."

229. The *SLCSEs* of claim 1 are the TCP/IP "routines" and "functions" in the protocol library, as I discussed in Section VII.D.1.d.i for [1C.1]. Therefore, the SLCSEs *include services related to at least networking protocol processes*.

230. If the claim also requires the SLCSEs include services related to *management of files*, Elnozahy+Draves still meets the claim. The '058 patent's specification never mentions "management," but it does mention "access[ing] files that reside in different locations" as an example of "File System services" that are examples of CSEs. EX1001, 6:10-17. The web server's HTML file retrieval

functionality, which is an *SLCSE* (as I discussed in Section VII.D.4.b.i for [4B]), retrieves HTML files from either the web server's cache or the disk; thus, this SLCSE includes 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."

231. As I discussed in Section VII.D.4.b.ii, an obvious alternative implementation of the user-space protocol library relies on interrupts instead of pollingIn Elnozahy+Draves, POSAs would have been motivated to *modify* some of the TCP/IP functions/routines in the protocol library, *i.e.* the *SLCSEs* (as I discussed in Section VII.D.1.d.i for [1C.1]), to use interrupts rather than polling *for a particular one of the software applications* for which minimizing latency is not required and where giving that application higher priority is not desirable.

232. POSAs would have reasonably expected success in such an implementation because interrupt-based processing was "conventional," as Elnozahy notes (Elnozahy, [0005]) and because having different versions of functions/routines that accomplish the same functionality in different ways was a well-known software engineering technique that was within a POSA's skill.

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."

233. The specification of the '058 patent states that "[i]n some embodiments, the software ***applications are provided with respective versions*** of the critical system elements. In some embodiments, the system elements which are ***application specific*** reside in user mode[.]" EX1001, 3:57-60.

234. As I discussed in Section VII.D.13 for claim 13, it would have been obvious to implement SLCSEs with two versions: (1) an interrupt-based version, and (2) a polling-based version. As I also discussed in Section VII.D.13 for claim 13, these two versions would be used for different applications. Thus, these different SLCSE versions are "respective versions" that different applications are "provided with," *i.e.*, *SLCSEs that are application specific*. EX1001, 3:57-60. Because these application-specific SLCSEs are in the user-mode library, they *reside in user mode*.

235. Elnozahy+Draves's kernel-mode versions of the TCP/IP functions/routines, which are OS *critical system elements* (OSCSEs), are in the OS kernel, as I discuss in Section VII.D.1.c.ii for [1B.2]; therefore, they *reside in the operating system kernel*.

236. POSAs also understood that these CSEs are *platform specific*. 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. The OSCEs are hardware and OS instance-specific because they are part of the kernel of the operating system running on the "server device" (Elnozahy, [0025]), as I discussed in Section VII.D.1.c for [1B].

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."

237. 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, 9:60-66. 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.

238. In Elnozahy+Draves, the file system, which is part of the kernel and which can be used by an SLCSE to read a file from disk by invoking the disk driver, meets claim 5's *kernel module*. *See* Section VII.D.5. SLCSEs operate *in user mode*, as I discussed in Section VII.D.1.d.ii for [1C.2]. POSAs understood

that for an SLCSE (such as the HTML-file-reading functionality I discussed in Section VII.D.5 for claim 5) to use the file system invoke the disk driver, there should be *data exchange between* the SLCSE and the driver in both directions; for example, the SLCSE should be able to convey to the driver which file is to be read, and the driver should be able to convey to the SLCSE when the file reading is complete (as, I discussed in Section VII.D.6.a for [6A]).

239. Draves discloses a technique "to allow user processes to pass valid memory pointers to kernel functions." Draves, 5:14-15. POSAs understood that using memory pointers was a way to exchange data; for example, the SLCSE could place the name of the desired file in a memory location and pass a pointer to that location to the file system, and the file system could place a notification that the file has been read in a memory location and pass a pointer to that location to the SLCSE. My testimony regarding a POSA's background knowledge is corroborated by, *e.g.*: Parameswar (EX1091), 17:30-32 (in patent filed in 2007 but related to application filed in May 2002, *see* code (62), stating in "BACKGROUND" that "In software applications and application libraries written in high level programming languages such as C and C++, it is common practice to pass data through memory pointers."); Freeman (EX1092), 4:10-13 ("The methods of initializing code that runs in SMI space, command calling conventions, and the passing of data pointers are all well known in the art and will not be repeated

herein."); Brinkerhoff (EX1093), [0006] ("An early advancement in efficiently moving data through components involved passing pointers to the data instead of passing the data itself. The data packets or parcels are stored once in memory and pointers to the data are passed from component to component or process to process. Passing pointers to the data was far more efficient than passing the actual data which required taking the data in and out of memory.").

240. POSAs would have been motivated to implement the data exchange this way, and would have reasonably expected success in doing so, because using pointers to shared memory locations was a well-known way to exchange data. Draves further discloses that its pointer-sharing process *uses mapping of virtual memory*. *See* Draves, 8:9-42 (explaining that "the user virtual address space… is mapped into the system virtual address space at an offset location" and "[t]he kernel is configured to correct for this offset when receiving pointers or other memory references from a user process"), Fig. 7 (below).

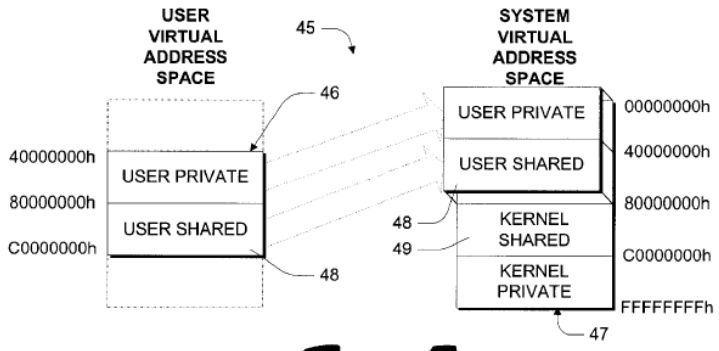


Fig. 7

241. Thus, Elnozahy+Draves meets claims 15.

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."

242. Elnozahy+Draves's SLCSEs *form a part of* software applications *by being* in a DLL *linked* to the applications, as I discussed in Section VII.D.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."

243. In Elnozahy+Draves, SLCSEs *utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping* as I explained in Section VII.D.11 for claim 11.

244. Elnozahy discloses that "[i]ntegrating user space extensions of the operating system kernel code and protocol stack into the web server enables [the] web server to operate substantially independently of the operating system." Elnozahy, [0025]. The "protocol stack" is provided by the "TCP/IP [protocol] library" (Elnozahy, [0022]), which contains the *SLCSEs* as I discussed in Section VII.D.1.d.i for [1C.1). Thus, POSAs understood that 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."

245. The user-mode HTML I discussed in Section VII.D.5.b for claim 5 is an *SLCSE*. Elnozahy discloses that the "protocol stack" provided by the "protocol library" enables the web server to operate "substantially independently" of the operating system, and describes one example of this independent operation as using a "user space file cache" to retrieve HTML files. Elnozahy, [0025]. Thus, POSAs understood, or at least found obvious, that there is an OS version of the functionality for reading HTML files (that the user-mode version is an alternative to), and that the user-mode HTML file-retrieval functionality is *not a copy* of the OS version, *i.e.* OSCSE, because only the user-space version (and not the OSCSE) would access the user-space file cache.

246. In addition, Elnozahy says "network processing of packets" in the "user space TCP/IP protocol library" is invoked if the "polling" mechanism determines that data is available. Elnozahy, [0028]. Elnozahy says this polling mechanism is different from the interrupt-driven mechanism used in traditional TCP/IP stacks that are in the OS kernel; thus, the packet "processing" routines in the user-space protocol library are *not a copy* of the corresponding OSCSE interpreters. *See* Elnozahy, [0005], [0008].

247. Finally, Elnozahy says the "user space protocol library" "eliminat[es]" certain "code" that might otherwise be found in a traditional TCP/IP block in the

OS kernel; thus, the *SLCSEs* in the "user space protocol library" are *not a copy* of the corresponding OSCSEs in the kernel's TCP/IP block. Elnozahy, [0024]-[0025].

* * *

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
A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.
Claim 6
[6A] 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,
[6B] 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.