

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AMAZON.COM, INC.,
Petitioner,

v.

VIRTAMOVE, CORP.,
Patent Owner.

IPR2025-00561
U.S. Patent No. 7,784,058

DECLARATION OF DARRELL D. E. LONG, Ph.D.
IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW
OF CLAIMS 1-16 AND 18 OF U.S. PATENT NO. 7,784,058

Amazon Ex. 1002 IPR Petition - USP 7,784,058
--

TABLE OF CONTENTS

I.	BACKGROUND-----	1
A.	Experience and Qualifications -----	1
B.	Materials Considered-----	5
II.	APPLICABLE LEGAL STANDARDS-----	7
A.	Claim Interpretation -----	8
B.	Obviousness -----	8
III.	PERSON OF ORDINARY SKILL IN THE ART -----	13
IV.	TECHNOLOGY BACKGROUND-----	14
A.	Shared Libraries -----	14
1.	Dynamic Linking of Shared Libraries -----	16
2.	Levine’s Linkers & Loaders Textbook Describes Shared Libraries and Dynamic Linking-----	18
B.	Shared Libraries With Operating System Functionality (“CSEs”)-----	19
1.	Ely and Thekkath Describe Shared Libraries for Networking -----	19
a.	Ely-----	19
b.	Thekkath-----	21
2.	Eggert Describes Shared Libraries for File Systems -----	23
V.	THE ’058 PATENT-----	24
A.	Summary of the ’058 Patent -----	24

B.	The Priority Date of the '058 Patent -----	26
C.	Claim Construction-----	26
VI.	CLAIMS 1-16 AND 18 OF THE '058 PATENT WOULD HAVE BEEN OBVIOUS -----	28
A.	Claims 1-6, 9-14, 16, and 18 Would Have Been Obvious in View of Ely and Levine -----	28
1.	Claim 1 -----	31
a.	1: Preamble -----	32
b.	1[a]: Processor-----	33
c.	1[b]: Operating System with a Kernel and OSCSEs -----	33
d.	1[c]: Shared Library with SLCSEs -----	34
i.	1[c][1]: SLCSEs are Functional Replicas of OSCSEs -----	36
ii.	1[c][2]: SLCSEs Form a Part of the Application -----	37
e.	1[d]: CSE Provided to First and Second Applications-----	39
i.	1[d][1]: First Instance of SLCSE Provided to First Application -----	39
ii.	1[d][2]: Unique Instance of Corresponding CSE Provided to Second Application -----	41
f.	1[e]: Two Instances of SLCSE Run Simultaneously -----	42

i.	1[e][1]: SLCSE Related to a Predetermined Function -----	42
ii.	1[e][2]: Second Instance of SLCSE -----	43
2.	Claim 2 -----	43
3.	Claim 3 -----	44
4.	Claim 4 -----	44
5.	Claim 5 -----	45
6.	Claim 6 -----	46
a.	6[a]: Kernel Provides Notification of an Event. -----	46
b.	6[b]: Event is Asynchronous and Outside of the Application. -----	47
7.	Claim 9 -----	48
8.	Claim 10 -----	48
9.	Claim 11 -----	49
10.	Claim 12 -----	50
11.	Claim 13 -----	51
12.	Claim 14 -----	51
13.	Claim 16 -----	52
14.	Claim 18 -----	52
B.	Claims 5-8 and 15 Would Have Been Obvious in View of Ely, Levine, and Thekkath -----	53
1.	Claim 5 -----	56

2.	Claim 6 -----	57
a.	6[a]: Kernel Provides Notification of an Event. -----	57
b.	6[b]: Event is Asynchronous and Outside of the Application. -----	58
3.	Claim 7 -----	58
4.	Claim 8 -----	59
5.	Claim 15 -----	60
a.	15[a]: Kernel Module Enables Data Exchange -----	60
b.	15[b]: Data Exchange Uses Virtual Memory Mapping. -----	61
C.	Claims 1-4, 9-12, 16, and 18 Would Have Been Obvious In View of Eggert and Levine -----	61
1.	Claim 1 -----	64
a.	1: Preamble -----	64
b.	1[a]: Processor -----	64
c.	1[b]: Operating System with Kernel and OSCSEs -----	64
d.	1[c]: Shared Library with SLCSEs -----	65
i.	1[c][1]: SLCSEs are Functional Replicas of OSCSEs -----	66
ii.	1[c][2]: SLCSEs Form a Part of the Application -----	67

e.	1[d]: CSE Provided to First and Second Applications-----	67
i.	1[d][1]: First Instance of SLCSE Provided to First Application -----	67
ii.	1[d][2]: Unique Instance of Corresponding CSE Provided to Second Application -----	68
f.	1[e]: Two Instances of SLCSE Run Simultaneously -----	69
i.	1[e][1]: SLCSE Related to a Predetermined Function -----	69
ii.	1[e][2]: Second Instance of SLCSE -----	70
2.	Claim 2 -----	70
3.	Claim 3 -----	71
4.	Claim 4 -----	71
5.	Claim 9 -----	72
6.	Claim 10 -----	73
7.	Claim 11 -----	73
8.	Claim 12 -----	74
9.	Claim 16 -----	75
10.	Claim 18 -----	75
VII.	SECONDARY CONSIDERATIONS OF NONOBVIOUSNESS-----	75
VIII.	CONCLUSION-----	76

I, Darrell D. E. Long, Ph.D., do hereby declare:

1. I am making this declaration at the request of Petitioner Amazon.com, Inc. (“Amazon”). Amazon has retained me as a technical expert in this matter.

2. I am being compensated for my work on this case. My compensation does not depend on the content of this Declaration or the outcome of these proceedings.

I. BACKGROUND

A. Experience and Qualifications

3. I am currently a Distinguished Professor Emeritus of Engineering at the Jack Baskin School of Engineering at the University of California. I have been on the faculty of UC Santa Cruz since 1988. I am also currently a Distinguished Visiting Scholar at Santa Clara University.

4. I received a B.S. degree from San Diego State University in 1984, a M.S. from the University of California, San Diego, in 1986, and a Ph.D. from the University of California, San Diego in 1988, all in computer science.

5. My previous work experience includes serving as Director of the Center for Research in Systems and Storage, a National Science Foundation Industry/University Cooperative Research Center, and the founding Director of the Storage Systems Research Center at the University of California, Santa Cruz. Before

that time, I was a research assistant at the University of California, San Diego, and a lecturer in Mathematics at San Diego State University.

6. After coming to the University of California, Santa Cruz, in 1988, I worked through the ranks from Assistant to Associate to Full Professor, culminating in my appointment as Distinguished Professor (the highest rank at the university). All this experience is covered in my curriculum vitae (attached as Ex. 1024), which provides a more detailed recitation of my employment history and tenure at various jobs.

7. I have also held numerous positions at the University of California and various government agencies and laboratories. I was vice chair and later chair of the University of California Committee on Research Policy. I also served on the University of California President's Council on the National Laboratories and the Science & Technology, National Security, and Intelligence committees for those laboratories. I currently serve on the University of California Academic Council Special Committee on Laboratory Issues (ACSCOLI).

8. I served for several years on the National Research Council's Standing Committee on Technology Insight-Gauge, Evaluate and Review (TIGER), on the Committee on Defense Intelligence Agency Technology Forecasts and Reviews, and the National Research Council's Committee on Science and Technology for Defense

Amazon.com, Inc. v. VirtaMove, Corp.

Declaration of Dr. Darrell D.E. Long – U.S. Patent No. 7,784,058

Warning. I recently served on the Intelligence Science and Technology Experts Group (ISTEG) for the National Academies of Sciences, Engineering, and Medicine.

9. Additionally, I have held visiting faculty positions at the Université Paris–Dauphine (Paris IX), the Conservatoire National des Arts et Métiers, the Université Paris–Descartes (Paris V), Sorbonne Université, the University of Technology, Sydney, the Center for Communications Research, the United States Naval Postgraduate School and Professor ad Honorem de la Universidad Católica del Uruguay. I was a European Organization for Nuclear Research (CERN) Associate Member.

10. I am conversant with several computer languages (for example, C, C++, FORTRAN, Go, LISP, Pascal, Perl, Java and others), scripting languages (various Unix/Linux shells, JavaScript, Python, and Tcl), markup languages (HTML, XHTML, XML, and numerous XML applications), operating systems (Unix and Linux of all varieties; MacOS; NetWare; Windows NT; Windows 2000, XP, Vista, Windows 7, 8, and 10, and Windows Server 2003, 2008, 2012, and 2016), and network communication protocols (TCP/IP, IPv4, and IPv6, plus numerous older stacks including IPX/SPX, NetBIOS/NetBEUI, and AppleTalk).

11. I currently am a member of various professional societies, including the Association for Computing Machinery (ACM), the Institute of Electrical and

Electronics Engineers (IEEE), and the American Association for the Advancement of Science (AAAS).

12. I was elected a Fellow of the IEEE in 2006 and a Fellow of the AAAS in 2008. I served on the IEEE Fellows Evaluation Committee in 2007, 2010, and from 2011 to 2014. From 2008 to 2011, I served as the Chair of the IEEE Reynold B. Johnson Information Storage Systems Award Committee and was a member of the IEEE Technical Field Awards Council.

13. I was the inaugural Editor-in-Chief of the IEEE Letters of the Computer Society (LOCS) and the Editor-in-Chief emeritus of the ACM Transactions on Storage (TOS). In 2002, I founded the Conference on File and Storage Technologies (FAST), one of the most prestigious venues in the computer data storage field.

14. I have co-authored numerous peer-reviewed journals in the area of secure computing, big data, computer systems, magnetics, and data storage. In particular, these peer-reviewed journals include *ACM Transactions on Storage*, *ACM/Baltzer Mobile Networks and Applications Journal*, *Cluster Computing Journal*, *Computer Networks*, *Computing Systems*, *Future Generation Computer Systems*, *IEEE Latin America Transactions*, *IEEE Micro*, *IEEE Transactions on Big Data*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Computers*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Magnetics*, *International*

Amazon.com, Inc. v. VirtaMove, Corp.

Declaration of Dr. Darrell D.E. Long – U.S. Patent No. 7,784,058

Journal in Computer Simulation, Internet Computing, Journal of Software Engineering and Knowledge Engineering, and the Journal of the ACM.

15. I have written widely for a broad range of other computing publications. For example, I was a co-author of a book titled *Avoiding Surprise in an Era of Global Technology*, which was peer-reviewed and published in 2005. In 1989, I co-authored another scholarly book with John L. Carroll titled *Theory of Finite Automata*. I have also co-authored chapters of various books, including *Distributed Data & Structures 4, Multimedia Communications: Directions and Innovations, Network Systems Design*, and *Progress in Simulation II*.

16. Based on my education and experience, I am an expert in at least the field of computer operating systems and operating environments. I have used my education, years of experience in this field, and my understanding of the perspective of a person of ordinary skill in the art to form the opinions expressed in this declaration.

B. Materials Considered

17. In preparing this Declaration, I have considered the following materials:

Exhibit No.	Description
1001	U.S. Patent No. 7,784,058 (“the ’058 patent”)

Exhibit No.	Description
1003	David Ely et al., <i>Alpine: A User-Level Infrastructure for Network Protocol Development</i> , Proc. 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01) (2001) (“Ely”)
1004	Paul R. Eggert & D. Stott Parker, <i>File Systems in User Space</i> , Proc. USENIX Winter 1993 Conference (1993) (“Eggert”)
1005	John R. Levine, <i>Linkers & Loaders</i> (2000) (“Levine”)
1006	Chandramohan A. Thekkath, et al., <i>Implementing Network Protocols at User Level</i> , 23 ACM SIGCOMM Computer Comm. Rev. 4, 64-73 (1993) (“Thekkath”)
1007	David Ely et al., <i>Alpine: A User-Level Infrastructure for Network Protocol Development</i> , 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01) (“Ely Presentation”)
1008	Excerpts from the '058 patent's file history
1009	Arindam Banerji et al., <i>Protected Shared Libraries – a New Approach to Modularity and Sharing</i> , Proc. USENIX Annual Technical Conf. (USENIX 1997) 59-75 (“Banerji”)
1010	UNIX System V Release 3 Programmer's Guide
1011	SunOS 4.1 Release Manual
1012	Solaris and SunOS System Interfaces Guide
1013	Sun Linker and Libraries Guide
1014	Linux Release Notes
1015	FreeBSD 3.3 Linker Information

Exhibit No.	Description
1016	Windows API Guide Reference Volume 3 (1992)
1020	Arnold, J. Q., “Shared Libraries on UNIX System V”, Summer Conference Proceedings, Atlanta 1986, USENIX Association, 1986.
1021	FreeBSD 3.0 Release Notes
1022	FreeBSD 3.3 Release Notes
1023	FreeBSD System Calls Manual – “select(2)”
1029	Australian Unix User Group Newsletter, Vol. 8, No. 5 excerpts

18. I have also relied on my education, training, experience, and knowledge of pertinent literature in the field of the '058 patent.

II. APPLICABLE LEGAL STANDARDS

19. I have been asked to provide my opinion as to whether the claims of the '058 patent would have been obvious to a person of ordinary skill in the art at the time of the alleged invention, in view of the prior art.

20. I am a computer scientist by training and profession. The opinions I am expressing in this report involve applying my training, technical knowledge, and experience to the evaluation of certain prior art with respect to the '058 patent.

21. Although I have been involved as a technical expert in patent matters before, I am not an expert in patent law. Therefore, the attorneys from Knobbe, Martens, Olson & Bear, LLP have provided me with guidance as to the applicable patent law in this matter. The paragraphs below express my understanding of how I must apply current principles related to patent validity to my analysis.

A. Claim Interpretation

22. Unless expressly stated herein, I have applied the plain and ordinary meaning of the claim terms, which I understand is the meaning that a person of ordinary skill in the art would have given to the claim terms at the time of the invention, based on a review of the intrinsic evidence (patent specification and file history).

23. I have been instructed that the time of the alleged invention here is around September 22, 2003, which I understand to be the earliest claimed priority date of the '058 patent. My conclusions would not change if the invention date were slightly earlier or slightly later.

B. Obviousness

24. It is my understanding that a claim is “obvious” if the claimed subject matter would have been obvious to a person of ordinary skill in the art at the time of the alleged invention. I understand that an obviousness analysis involves several considerations.

25. I understand that the following factors must be evaluated to determine whether a claim would have been obvious: (1) the scope and content of the prior art; (2) the differences, if any, between each claim of the '058 patent and the prior art; (3) the level of ordinary skill in the art in September 2003; and (4) additional considerations, if any, that indicate that the invention was obvious or not obvious. I understand that these “additional considerations” are often referred to as “secondary considerations” of non-obviousness or obviousness.

26. I also understand that the frame of reference when evaluating obviousness is what a hypothetical person of ordinary skill in the pertinent art would have known in September 2003. I understand that the hypothetical person of ordinary skill is presumed to have knowledge of all pertinent prior art references.

27. I understand that a prior art reference may be a pertinent prior art reference (or “analogous art”) if it is in the same field of endeavor as the patent or if it is pertinent to the problem that the inventors were trying to solve. Here, all of the references relied on in my obviousness analysis below are in the same field of endeavor as the '058 patent, e.g., computing systems and operating system architectures. The references are also pertinent to a particular problem the inventors were focused on, e.g., extending operating system services using shared libraries.

28. It is my understanding that the law recognizes several rationales for combining references or modifying a reference to show the obviousness of the claimed subject matter. Some of these rationales include:

- combining prior art elements according to known methods to yield predictable results;
- simple substitution of one known element for another to obtain predictable results;
- a predictable use of prior art elements according to their established functions;
- using known techniques to improve similar devices (methods, or products) in the same way;
- applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;
- choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success (in which case a claim would have been obvious to try);
- 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; and

- 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.

29. I understand that “secondary considerations” must be considered as part of the obviousness analysis when present. I further understand that the secondary considerations may include: (1) a long-felt but unmet need in the prior art that was satisfied by the claimed invention; (2) the failure of others; (3) skepticism by experts; (4) commercial success of a product covered by the patent; (5) unexpected results achieved by the claimed invention; (6) industry praise of the claimed invention; (7) deliberate copying of the invention; and (8) teaching away by others.

30. I also understand that evidence of the independent and nearly simultaneous “invention” of the claimed subject matter by others is a secondary consideration supporting an obviousness determination and may support a conclusion that a claimed invention was within the knowledge of a person of ordinary skill as of September 2003. I am unaware of any evidence of secondary considerations that would suggest that the claims of the '058 patent would have been nonobvious in September 2003.

31. I understand that when assessing obviousness, using hindsight is impermissible; that is, what is known today or what was learned from the teachings

of the patent should not be considered. The patent should not be used as a road map for selecting and combining items of prior art. Instead, obviousness must be considered from the perspective of a person of ordinary skill at the time the alleged invention was made, which is September 2003 in this case.

32. I also understand that an obviousness analysis must consider the invention as a whole, as opposed to just a part or element of the invention. I understand this “as a whole” assessment to require showing that one of ordinary skill in the art at the time of invention, confronted by the same problems as the inventor and with no knowledge of the claimed invention, would have selected the elements from the prior art and combined them in the claimed manner.

33. I understand that when a person of ordinary skill in the art would be unable to ascertain the scope of a claim with reasonable certainty, the claim is indefinite. I also understand that a patentee’s definition of a claim term can render a patent claim indefinite if an ambiguity in the definition would prevent a person of ordinary skill from ascertaining the boundaries of the claim with reasonable certainty. But even where the outer boundaries of a claim term are indefinite, I understand that it may be possible to identify examples that satisfy the claim term. When such examples are available, they may be compared to the prior art to support a finding of obviousness, even though the outer boundaries of the claim are indefinite.

34. It is my understanding that something is “inherent in,” and therefore taught by, the prior art, if it necessarily flows from the explicit disclosure of the prior art. I understand that the fact a certain result or characteristic may or may not be present in the prior art is insufficient to establish inherency. However, if the result or characteristic is necessarily present based upon the explicit disclosure in the prior art, it is inherent in the prior art and is therefore disclosed.

III. PERSON OF ORDINARY SKILL IN THE ART

35. It is my understanding that when interpreting the claims of the '058 patent and evaluating whether a claim would have been obvious, I must do so based on the perspective of a person of ordinary skill in the art at the relevant priority date. I have been instructed to assume for the purposes of my opinions that the relevant priority date of the '058 patent is September 22, 2003.

36. I understand that several factors are considered when determining the level of ordinary skill in the art. Those factors may include: (1) the type of problems encountered in the art; (2) prior art solutions to those problems; (3) the rapidity with which innovations are made; (4) the sophistication of the technology; and (5) the education level of active workers in the field. A person of ordinary skill in the art must have the capability of understanding the scientific and engineering principles applicable to the pertinent art.

37. Based on my review of the specification and claims of the '058 patent, it is my opinion that a person of ordinary skill in the art would have had a minimum of a bachelor's degree in computer engineering, computer science, software engineering, or a similar field, and approximately two years of industry or academic experience in a related field. Work experience could substitute for formal education, and additional formal education could substitute for work experience.

38. My conclusions below that the claims of the '058 patent would have been obvious would remain the same even if the priority date, field of endeavor, or level of ordinary skill in the art were slightly different.

39. I meet the above definition of a person of ordinary skill in the art and did so as of September 2003. Also, I have worked with persons of ordinary skill in the art in my professional and academic experiences, and I have an understanding of their skill level around September 2003.

IV. TECHNOLOGY BACKGROUND

A. Shared Libraries

40. Shared libraries allow multiple software applications to efficiently share common functionality without wasting memory. Rather than providing a copy of computer code for common functionality to each application, the applications (1) link to a shared library that contains that functionality and (2) use a single copy of the

library's contents whenever possible, as explained below. (Ex. 1020 (Shared Libraries on UNIX System V (1986)), 395.)

41. Different parts of a shared library are shared in different ways. One part of a shared library contains executable code that every linked application can access. (Ex. 1009 (Banerji), 60-61.) Because this code does not change from one application to another, every application shares an identical copy. (*Id.*) Sharing this code avoids duplication and saves space in the computer's memory. (Ex. 1020 (Shared Libraries on UNIX System V (1986)), 395.) One example is if multiple applications need to connect to the Internet. In this example, all of the applications can link to a single shared library that contains code for establishing such connections rather than every application having its own copy of the same code. Similarly, the '058 patent explains that it was "common practice" for all applications that access files on a computer to use shared code for doing so. (Ex. 1001 at 3:32-37 ("All applications currently share code for common services, such services include operations such as ... open a file, write to the file, read from the file, etc.").)

42. Shared libraries also contain data that is accessed by the shared code. (Ex. 1009 (Banerji), 60-61; Ex. 1010 (Unix System V Release 3 Programmer's Guide), 8-28.) Each linked application gets its own copy of the data from the shared library. (*Id.*; Ex. 1013 (Sun Linker and Libraries Guide), 100.) Each application's copy of the data is private—i.e., it is unique to that application and can be accessed

only by code running in the context of that application. (Ex. 1005 (Levine), 169-172.)

Using the previous example of multiple applications connecting to the internet, every application may share the same library code for connecting to the Internet, but each application may need to access a different web site or use a unique network configuration. Although all applications share the same library *code*, each application uses its own private copy of the library *data* to keep track of its network configuration and the unique Internet address it is accessing. (*Id.*) Giving each application its own private copy of the data allows the applications to share code without interfering with other applications' differing uses of the shared code. (*Id.*)

1. Dynamic Linking of Shared Libraries

43. Popular operating systems such as Unix included shared libraries by the 1980s. (Ex. 1010 (Unix System V Release 3 Programmer's Guide), 3-24 ("Beginning with Release 3.0, shared libraries are supported."), Ch. 8 (explaining how to build and use shared libraries).) Other operating systems implemented shared libraries shortly thereafter in the late 1980s and early 1990s. (Ex. 1011 (SunOS 4.1 Release Manual), 25 (shared libraries supported in SunOS 4.0 and 4.1 by 1990); Ex. 1012 (Solaris and SunOS System Interfaces Guide), 2; Ex. 1013 (Sun Linker and Libraries Guide), 3, 11-12; Ex. 1014 (Linux Release Notes); Ex. 1005 (Levine), 4.) By 2003, shared libraries were widely used in most operating systems. (Ex. 1009 (Banerji), 60 ("Most commercial operating systems support shared libraries in one form or another ...").)

The '058 patent confirms this. (Ex. 1001, 7:3-5 (“what is commonly done is to provide an application library in shared code space”).) The patent owner’s statements during prosecution also confirm this. (Ex. 1008, 21 (2/18/2009 office action response) (“It is fully acknowledged that [prior art from 1995] describe[s] shared libraries”).)

44. Shared libraries can be linked with applications using “static” linking, performed when an application is first compiled and installed, or “dynamic” linking, which delays the linking process until an application starts running. (Ex. 1005 (Levine), 188; Ex. 1012 (Solaris and SunOS System Interfaces Guide), 2-3; Ex. 1010 (Unix System V Release 3 Programmer’s Guide), G-7 (“Dynamic linking refers to the ability to resolve symbolic references at run time.”), G-18 (“Static linking refers to the requirement that symbolic references be resolved before run time.”); Ex. 1013 (Sun Linker and Libraries Guide), 4.) Dynamic linking was implemented in UNIX operating systems in the late 1980s and was quickly adopted by other popular operating systems such as SunOS, Solaris, and Windows. (Ex. 1012 (Solaris and SunOS System Interfaces Guide), 109; Ex. 1013 (Sun Linker and Libraries Guide), 3; Ex. 1016 (Windows API Guide Reference Volume 3 (1992)), 1.) Thus, many operating systems supported dynamic linking of shared libraries well before 2003.

2. Levine's Linkers & Loaders Textbook Describes Shared Libraries and Dynamic Linking

45. In 2000, a textbook called Linkers and Loaders was published. (Ex. 1005 (“Levine”).) Levine explained that most operating systems included shared libraries and dynamic linking by the time it was published. (Ex. 1005 (Levine), xi, 4, 187-88, 206 (Linux and Unix used “ELF” format for shared libraries), 217-18 (Microsoft Windows used “DLL” format).) Levine explains how shared libraries were commonly created and linked to applications using either static or dynamic linking. (Ex. 1005 (Levine), xiii, ch. 9 (shared libraries and static linking), ch. 10 (dynamic linking).)

46. As Levine explains, most applications “usually turn out to share a lot of common code”; therefore, if “all the programs that use a library can share a single copy of it,” significantly less storage space is needed. (Ex. 1005 (Levine), 4.) The process of linking shared libraries to applications is “nearly the same” even though different shared library formats may vary slightly. (Ex. 1005 (Levine), 190.) When an application program that is linked to shared libraries starts running, “startup code finds those [linked] libraries and maps them into the program’s address space before the program starts.” (Ex. 1005 (Levine), 188.) The program’s address space is the part of the computer’s memory allotted to that program.

B. Shared Libraries With Operating System Functionality (“CSEs”)

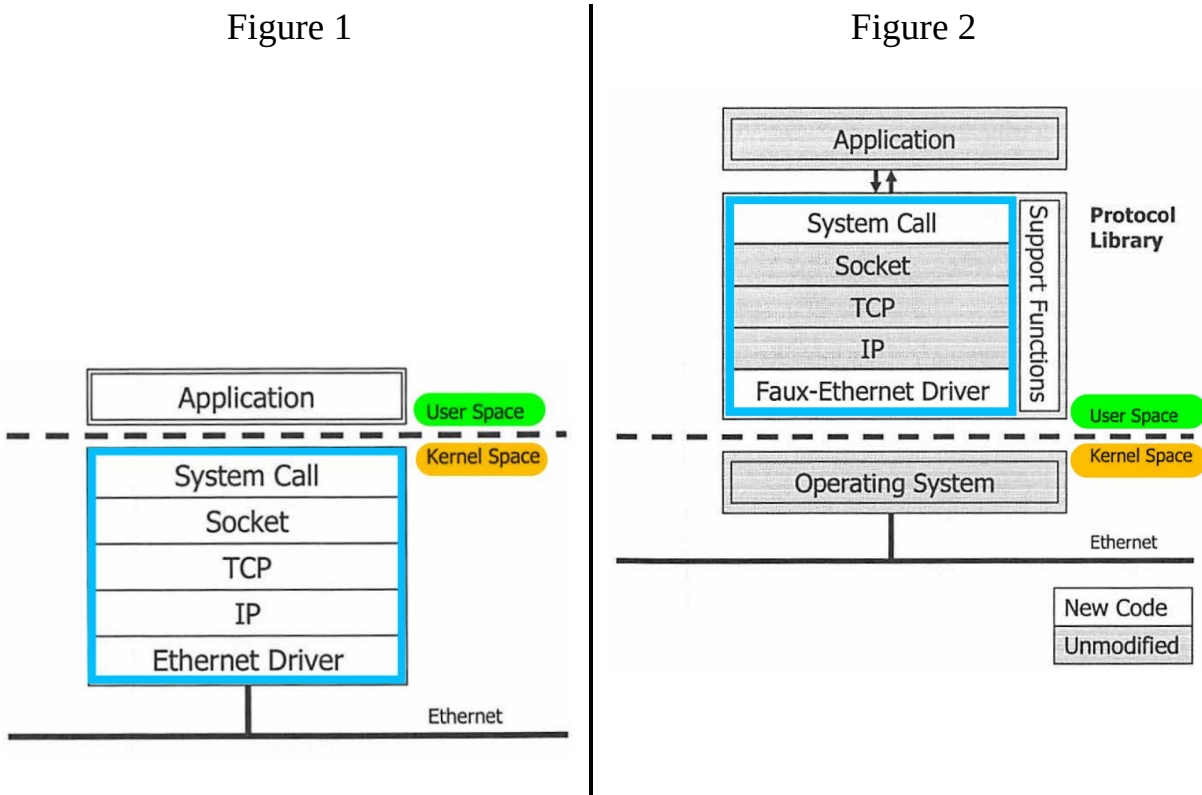
47. The ’058 patent uses the phrase “critical system elements” (“CSEs”) to describe operating system functionality replicated in the shared library. (Ex. 1001, 6:6-10.) Two examples of CSEs provided in the ’058 patent are “Network services including TCP/IP” and “File System services.” (*Id.*, 6:11-28.) The prior art disclosed replicating both of these CSEs in shared libraries.

1. Ely and Thekkath Describe Shared Libraries for Networking

a. Ely

48. In 2001, David Ely and other researchers at the University of Washington published a paper titled “Alpine: A User-Level Infrastructure for Network Protocol Development.” (Ex. 1003 (“Ely”).) They presented their paper at the 3rd USENIX Symposium on Internet Technologies and Systems. (Ex. 1007 (“Ely Presentation”).) The paper explains that traditional networking stacks were difficult for developers to work with because they were part of the “kernel,” which is the core of an operating system. (Ex. 1003 (Ely), 171.) Any changes to a traditional networking stack required rebuilding the entire kernel. (*Id.*) To overcome this inconvenience, Ely presented the “Alpine” system, which copies the networking stack from an operating system’s kernel into a shared library that is dynamically linked with applications. (*Id.*, 171, 177.)

49. Figures 1 and 2 of Ely show, respectively, a “traditional” stack where “all network processing is performed in the kernel” and Alpine’s library where “unmodified networking code is placed:”



(Ex. 1003 (Ely), Figs. 1, 2.)¹ As shown, the traditional **networking** stack is implemented in **kernel space**. (Ex. 1003 (Ely), Fig. 1.) But Alpine’s **networking** stack is copied into a library in **user space**. (*Id.*, Fig. 2.) User space is the part of a computer’s memory in which applications and shared libraries reside. Kernel space is the separate area of memory in which the kernel resides.

¹ Figures throughout this Declaration may be annotated and colored for ease of reference.

50. Alpine’s networking stack was implemented as a “dynamic library,” a well-known type of shared library widely used in popular operating systems. (Ex. 1003 (Ely), 177; ¶¶ 43-44, above.) Alpine’s library used “dynamic linking” to integrate with pre-existing applications. (Ex. 1003 (Ely), 177.)

51. Alpine was “fully implemented in the FreeBSD 3.3 operating system.” (*Id.*, 178.) “However, very little of [Alpine’s] code [was] specific to this version of FreeBSD, and most of it [was] portable to any Unix environment.” (*Id.*) Thus, Ely explained that Alpine broadly “support[ed] a FreeBSD networking stack on top of a Unix operating system.” (*Id.*, 171.)

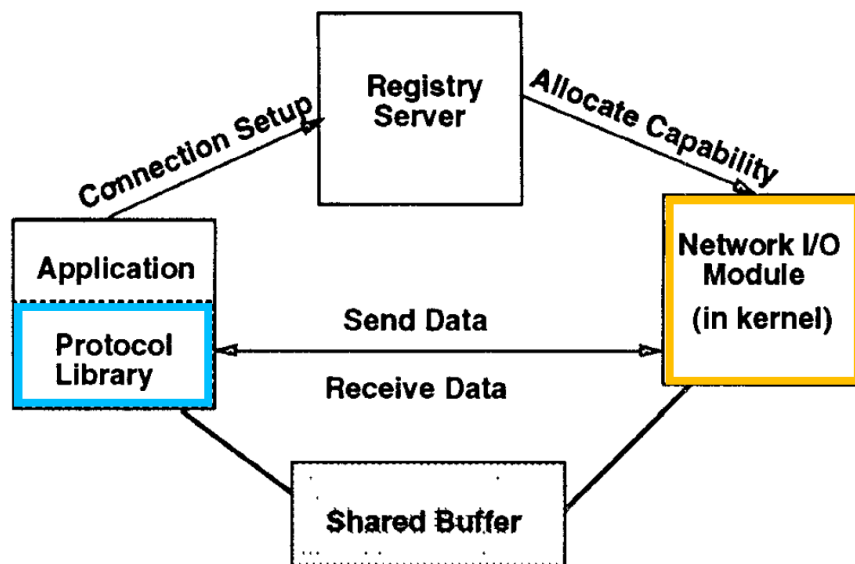
52. Presentation slides from the conference where Ely was first presented further confirm that Alpine was implemented as a shared library. The presentation shows that applications linked to Alpine using the command “LD_PRELOAD=libAlpine.so.” (Ex. 1007 (Ely Presentation), 8; *id.* at 3, 6-7.) The “.so” extension of the libAlpine file shows that it was a shared library. (Ex. 1015 (FreeBSD 3.3 Linker Information), line 326 (“a ‘.so’ extension indicates a shared library.”).)

b. Thekkath

53. In 1993, Chandramohan Thekkath and other researchers published a conference paper titled “Implementing Network Protocols at User Level.” (Ex. 1006 (“Thekkath”).) Thekkath, like Ely, described research conducted at the University of

Washington’s Department of Computer Science and Engineering. (*Id.*; Ex. 1003 (Ely).)

54. Thekkath implemented networking protocols outside of the kernel to allow for “ease of prototyping, debugging, and maintenance.” (Ex. 1006 (Thekkath), 64.) Thekkath’s system allowed for the “co-existence of multiple protocols” that could “exploit application-specific knowledge,” unlike the traditional kernel-based networking stack, which was generalized for the entire OS. (*Id.*) Like Ely, Thekkath used a shared library to store its networking protocols outside the kernel. (Ex. 1006 (Thekkath), 64, 67, 68.) This system, with the “protocol library” linked to an application, is shown in Figure 2:



(Ex. 1006 (Thekkath), Fig. 2.) Because the “protocol library [] is linked into the application” and “[t]he library contains the code that implements the communication

protocol,” applications can directly access several different networking protocols stored in the library. (*Id.*, 67, 68.)

55. Because the protocol library is in user space rather than the kernel, it does not have direct access to networking hardware. Instead, the protocol library exchanges data with a “Network I/O Module” in the kernel that does have access to networking hardware. (Ex. 1006 (Thekkath), 69.) This kernel module acts as an interface between the protocol library and networking devices. (*Id.*)

2. Eggert Describes Shared Libraries for File Systems

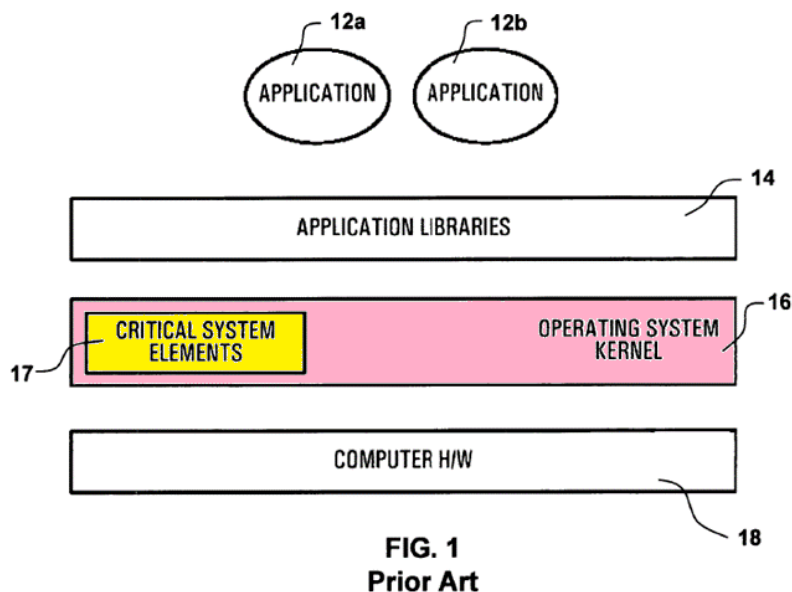
56. In 1993, Paul Eggert and another researcher published a paper titled “File Systems in User Space” and presented it at the USENIX Winter Conference. (Ex. 1004 (“Eggert”).) This paper disclosed “two user-extensible file systems,” collectively called “IFS,” that extended a computer’s default file system with added capabilities. (*Id.*, 229.) These file systems were implemented with libraries that applications could dynamically link to. (*Id.*, 229, 235, 236, 237.) When an application dynamically linked to the IFS shared library, that library intercepted the application’s calls to the operating system, allowing the application to access files in new ways. (Ex. 1004 (Eggert), 231-32.)

V. THE '058 PATENT

A. Summary of the '058 Patent

57. The '058 patent discloses shared libraries containing what it refers to as “critical system elements” (CSEs), which replicate functionality from an OS kernel. (Ex. 1001, 1:58-64.) “The term replica used [in the '058 patent] is meant to denote a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system[.]” (*Id.*, 1:66-2:3.)

58. Figure 1 shows a “conventional architecture where critical system elements execute in kernel mode”:



(Ex. 1001, Fig. 1; *id.*, 6:62-63.)

59. An “important distinction between [the '058 patent] and [some] prior art systems and architectures is the ability to allow a CSE to execute in the same

context as an application.” (*Id.*, 1:46-48.) CSEs “are replicated through the use of shared libraries” and “replicated CSEs are then able to run in the context of a software application” like any other shared library. (*Id.*, 5:22-34.)

60. “By way of example; a Linux based platform provides a TCP/IP [i.e., networking protocol] stack in the Linux kernel. [The ’058 patent] provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack from the Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD [TCP]/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library.” (Ex. 1001, 5:41-49.) The “TCP/IP stack” that the ’058 patent refers to is a “stack” of common networking protocols: the “transmission control protocol” (TCP) and the “internet protocol” (IP). These two protocols enable the reliable transmission and receipt of data over a network.

61. Figure 4 of the ’058 patent shows “how critical system elements exist in the same context as an application.” (*Id.*, 5:11-12.) In this figure, “the CSE shared library, 47, provides an implementation of a critical system element.” (*Id.*, 8:23-27.)

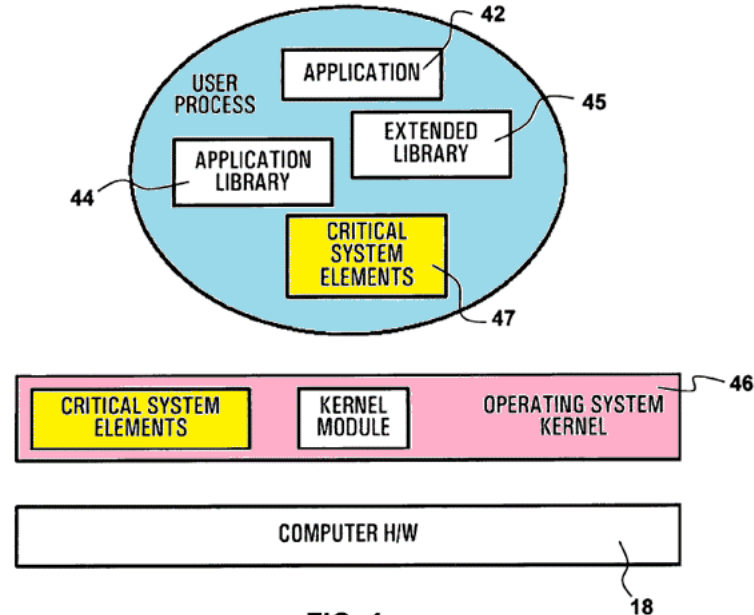


FIG. 4

(*Id.*, Fig. 4.)

B. The Priority Date of the '058 Patent

62. I understand that the '058 patent issued on August 24, 2010, from U.S. Patent Application 10/946,536 ("the '536 application"), filed September 21, 2004. I also understand that the '536 application claims priority to U.S. Provisional Application No. 60/504,213, which was filed on September 22, 2003. (Ex. 1001 (the '058 patent).) I understand that the '058 patent's earliest possible priority date is, therefore, September 22, 2003.

C. Claim Construction

63. The patent owner sued Petitioner for infringement of the '058 patent in *VirtaMove, Corp. v. Amazon.com, Inc.*, No. 7:24-cv-00030-ADA-DTG (W.D. Tex.)

(“the Litigation”). Although the parties proposed constructions for certain terms pertaining to issues in the Litigation, the obviousness challenges in this proceeding do not depend on those constructions. The prior art renders the challenged patent claims obvious regardless of whether any of the proposed constructions in the Litigation are adopted.

64. Petitioner has asserted in the Litigation that two limitations in claim 1 of the '058 patent—“critical system elements” and “functional replicas”—are indefinite because their boundaries are unclear. (Ex. 1027.) The indefiniteness of these limitations does not preclude a determination of obviousness here because the claim limitations clearly cover the prior art, even though the limitations’ outer boundaries are uncertain, as explained below.

65. For “critical system elements,” the full scope of the limitation is unclear but one of the '058 patent’s dependent claims recites two specific examples that are expressly within the scope: “services related to ... network protocol processes” and “services related to ... the management of files.” (Ex. 1001, claim 12.) The prior art relied on in this proceeding discloses both examples. Specifically, Ely and Thekkath disclose critical system elements related to network protocol processes. (¶¶ 48-52, 53-55, above.) Eggert discloses critical system elements related to the management of files. (¶ 56, above.) Therefore, regardless of what the boundaries of the term “critical system elements” may be, the prior art is within those boundaries.

66. Regarding the term “functional replicas,” the full scope of the limitation is unclear but example embodiments in the specification confirm that the prior art here is within the scope. These example embodiments make clear that examples of “functional replicas” include an alternate implementation of the TCP/IP networking protocol and an extension of a computer’s file system. (Ex. 1001, 3:17-30, 5:35-53, 6:11-15, claim 12.) The prior art here discloses both of these examples. (¶¶ 48-52, 53-55, 56, above.) Regardless of what the boundaries of the term “functional replicas” may be, the prior art is within those boundaries.

67. Accordingly, my opinion as to the obviousness of the claims would not be affected regardless of whether the terms “critical system elements” or “functional replicas” are held indefinite or not by the district court.

VI. CLAIMS 1-16 AND 18 OF THE ’058 PATENT WOULD HAVE BEEN OBVIOUS

A. Claims 1-6, 9-14, 16, and 18 Would Have Been Obvious in View of Ely and Levine

68. Ely discloses that its Alpine library was implemented in the “FreeBSD 3.3” operating system and could be easily transferred to a “Linux” operating system. (Ex. 1003 (Ely) at 178, 182.) As Levine explains, these operating systems used the “ELF” format for shared libraries. (Ex. 1005 (Levine) at 62 (“ELF has been adopted by the popular freeware Linux and by BSD variants of UNIX as well”), 206.) FreeBSD began using ELF in version 3.0 and continued to do so in version 3.3. (Ex.

1021 (FreeBSD 3.0 Release Notes) §1.3; Ex. 1022 (FreeBSD 3.3 Release Notes) §4.)

Levine also explains how shared libraries were implemented using ELF. (Ex. 1005 (Levine) at 190.) A person of ordinary skill in the art would, therefore, understand that Ely’s Alpine library, which was implemented on FreeBSD 3.3, was an ELF - shared library that worked in accordance with Levine’s explanation of such libraries.

69. Even if Ely did not inherently disclose the ELF shared libraries that Levine describes, it would have been obvious to a person of ordinary skill in the art to combine Ely and Levine. A person of ordinary skill in the art would have been motivated to combine the teachings of Ely and Levine to dynamically link the Alpine shared library in the manner disclosed in Levine for several reasons, discussed below.

70. First, Ely contains an express teaching, suggestion, or motivation because it calls for implementing Alpine as a dynamically linked library, and Levine discloses how dynamically linked libraries were implemented as discussed above. (¶¶ 43-44, 48-52, above.)

71. Second, Ely sought to achieve shortened development times and easier debugging of networking protocols by using a user-space library. (Ex. 1003 (Ely), 171.) This disclosure would have motivated a person of ordinary skill in the art to look to the dynamically linked shared libraries disclosed in Levine, and Levine explains that “[d]ynamically linked shared libraries are easier to update than statically linked shared libraries.” (Ex. 1005 (Levine), 205.)

72. Third, dynamically linked shared libraries were widely known and supported. In my work, I routinely used dynamically linked shared libraries as a conventional type of library well before the '058 patent's earliest possible priority date. For example, the REINAS and Swift research described in my CV, which I worked on between 1989 and 1997, used shared libraries. Therefore, using dynamically linked shared libraries in place of any other type of library would represent nothing more than a simple substitution of one known element (Levine's dynamically linked library) for another (Ely's dynamically linked library) to obtain predictable results (dynamically linking a library with applications). (Ex. 1005 (Levine), 205, 206, 217.) Moreover, the combination represents using a known technique (Levine's shared library execution) to improve a similar device and method (a computer executing Ely's Alpine library) in the same way. (*Id.*) The combination also applies a known technique (Levine's shared library execution) to a known device and method (a computer executing Ely's Alpine library) that is ready for improvement and yields predictable results. (*Id.*)

73. Fourth, a person of ordinary skill in the art would have been motivated to use the ELF shared library format described in Levine to implement Ely's Alpine library because ELF was one of a finite number of identified, predictable solutions for dynamically linking a library to an application. (See Ex. 1005 (Levine) at 206 (ELF solution), 217 (DLL solution).)

74. Fifth, Ely discloses using a shared library that is dynamically linked to applications, giving a person of ordinary skill in the art a reasonable expectation of success in doing so using the conventional techniques disclosed in Levine. (Ex. 1003 (Ely), 177 (“Alpine exploits dynamic linking” to load “Alpine’s dynamic library before any other dynamic librar[ies]”).) Additionally, Levine discloses that dynamically linked shared libraries were widely supported and already available in most operating systems, further strengthening the person of ordinary skill in the art’s reasonable expectation of success. (Ex. 1005 (Levine), 206 (“by the late 1990s [dynamic linking] had become the standard”), 217; *see also* ¶ 72, above (explaining that dynamically linked shared libraries were routinely used in my own work).)

75. Ely and Levine disclose each limitation of claims 1-6, 9-14, 16, and 18, and a person of ordinary skill in the art would have found it obvious to combine their teachings for the reasons discussed above. Thus, those claims would have been obvious to a person of ordinary skill in the art.

1. Claim 1

76. Claim 1 recites a computing system comprising (a) a processor, (b) an operating system with operating system critical elements (OSCSEs) for running in kernel mode, and (c) a shared library containing shared library critical system elements (SLCSEs) for use by software applications in user mode. (Ex. 1001 at 10:51-59.) The claim further recites several features of the SLCSEs—requiring that

(i) some are “functional replicas” of the OSCSEs, (ii) an application runs its own instance of an SLCSE that is not shared with other applications, and (iii) two instances of the same SLCSE are provided to two different applications. (*Id.* at 10:60-11:14.) Ely discloses each of these limitations, expressly or inherently. The limitations that are inherent in Ely are conventional features of dynamically linked shared libraries, which have been widely used in popular operating systems since the 1980s. Levine expressly discloses these conventional dynamic-linking features on which Ely relies.

a. 1: Preamble

77. The preamble of claim 1 recites “[a] computing system for executing a plurality of software applications.” Ely discloses the preamble. Specifically, Ely discloses that the “Alpine” library runs “on top of a Unix operating system.” (Ex. 1003 (Ely), 171.) A person of ordinary skill in the art would have understood that Unix was a conventional computer operating system for executing a plurality of software applications. Moreover, Ely discloses that “Alpine interfaces with [software] applications” (plural) and was implemented on a computing system. (Ex. 1003 (Ely), 176; *see also id.* at 178 (disclosing implementation on a “test machine”), Fig. 7 (showing the execution of applications on a computer running Alpine), 180 (“Alpine running in a graphical debugger” with “a telnet application,” and a “mail program and web browser.”) Thus, Ely discloses this limitation.

b. 1[a]: Processor

78. Claim 1 recites “a processor.” Ely discloses the Alpine runs on a system with a “CPU,” which a person of ordinary skill in the art would understand is an abbreviation for central processing unit—a type of processor. (Ex. 1003 (Ely), 175 (describing how Alpine optimizes “CPU utilization”).) Ely also discloses a computing system as explained above (§ 77, above (discussing claim 1 preamble)), which a person of ordinary skill in the art would have understood to include a processor as a necessary component. Thus, Ely discloses, or at least suggests, this limitation.

c. 1[b]: Operating System with a Kernel and OSCSEs

79. Claim 1 next recites “an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.” The ’058 patent admits that this limitation was well-known in the prior art. (Ex. 1001, Fig. 2 (“Prior Art” system with OS and kernel 16, and OSCSEs 17), 6:62-7:22 (discussing “conventional architecture where critical system elements execute in kernel mode”).)

80. Ely also discloses this limitation. Specifically, Ely discloses a “networking stack” that runs in the “kernel” of “traditional operating systems.” (Ex. 1003 (Ely), 171.) The services in this networking stack are CSEs. (§§ 63-67, above

(identifying examples of “critical system elements” in the ’058 patent); Ex. 1001, 3:17-30, 5:35-53, 6:11-15 (“[n]etwork services” are CSEs); *id.*, claim 12.) The traditional kernel-based networking stack is shown in Ely’s Figure 1:

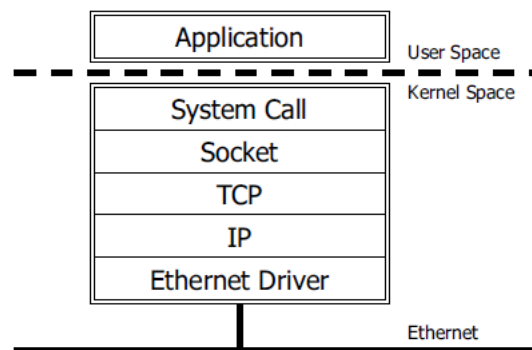


Figure 1: In a traditional network stack, applications interface with the network through a set of system calls (e.g., `socket`, `bind`, `send`), and all network processing is performed in the kernel.

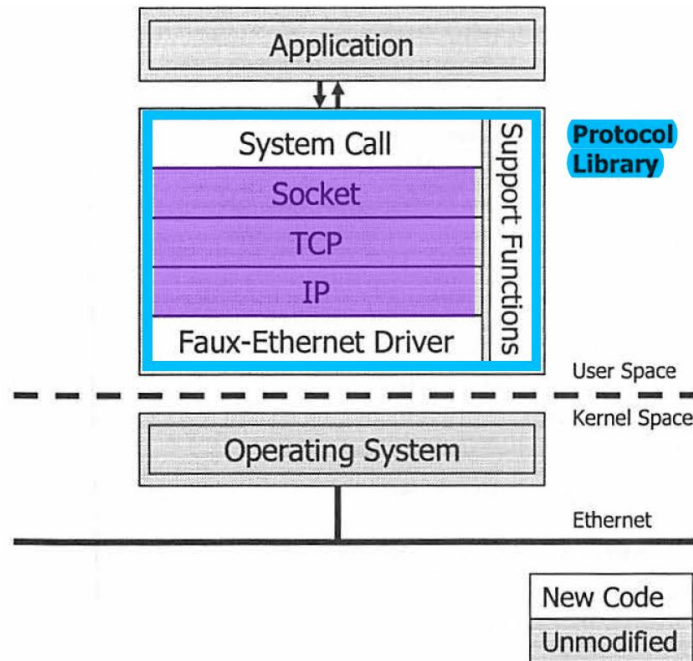
(Ex. 1003 (Ely), Fig. 1.) In this traditional stack, “all network processing is performed in the kernel.” (*Id.*)

81. Although Ely’s Alpine library adds another networking stack outside the kernel, the traditional kernel-based networking stack remains in place. (Ex. 1003 (Ely), 172 (“No kernel changes.”), 175 (“Alpine provides a service that is also provided by the kernel”), 178.) Thus, Ely discloses claim 1’s OSCSEs for running in kernel mode.

d. 1[c]: Shared Library with SLCSEs

82. Claim 1 next recites “a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.” Ely discloses this limitation.

83. Ely discloses placing a networking stack in a “user-level library” called the “[Alpine library](#).” (Ex. 1003 (Ely), 176, Fig. 2.) This is shown in Figure 2:



(*Id.*, Fig. 2.) The Alpine library includes the “[socket layer and the TCP/IP layer](#)” and is “responsible for sending and receiving packets and maintaining state about connections.” (*Id.*, 173.)

84. Ely explains that the Alpine library provides networking functionality “also provided by the kernel.” (Ex. 1003 (Ely), 176.) Ely thus discloses that both OSCSEs (e.g., elements of a network stack in the kernel) and SLCSEs (e.g., elements of the Alpine library, also providing a network stack but outside of the kernel) are present in the same system. (Ex. 1003 (Ely), 176.)

85. Ely further discloses that the Alpine library is a “dynamic library,” which is a well-known type of shared library. (Ex. 1003 (Ely), 177; ¶¶ 40-44, above

(background on shared libraries and dynamic linking).) Ely’s Presentation contains a slide further confirming that Alpine was implemented as a shared library. The slide explains that applications link with Alpine using the command “LD_PRELOAD=libAlpine.so.” (Ex. 1007 (Ely Presentation), 8; *id.* at 3, 6-7.) The “.so” extension of this “libAlpine” file shows that the file is a shared library. (Ex. 1015 (FreeBSD 3.3 Linker Information Page), line 326 (“By convention, a ‘.so’ extension indicates a shared library.”).) Thus, Ely discloses a shared library containing SLCSEs for use by applications in user mode.

i. 1[c][1]: SLCSEs are Functional Replicas of OSCSEs

86. Claim 1 further recites “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs.” The ’058 patent explains that replicas may be exact replicas or slightly differing replicas. (Ex. 1001, 8:27-40; *see also id.*, 5:20-5:34, 9:52-56.) Ely discloses both exact replicas and differing replicas.

87. The exact replicas in Ely include an “unmodified socket layer and the TCP/IP layers” that are “responsible for sending and receiving packets and maintaining state about connections,” as well as “the same networking API as the kernel.” (Ex. 1003 (Ely), 173.) The differing replicas include SLCSEs “that emulate their counterparts in the kernel,” including SLCSEs “dealing with memory allocation,” implementations of kernel “software interrupt functions,” initialization

functions, and functions the “networking stack uses ... to handle synchronous events and ... asynchronous events.” (Ex. 1003 (Ely), 174-75.) Ely, therefore, discloses SLCSEs that are functional replicas of OSCSEs.

ii. 1[c][2]: SLCSEs Form a Part of the Application

88. Claim 1 further recites that some of the SLCSEs “are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.” Ely inherently discloses this limitation and also renders it obvious in view of Levine.

89. Ely discloses the use of dynamic linking to load the Alpine library. (Ex. 1003 (Ely), 177.) The ’058 patent admits that dynamic linking a shared library of SLCSEs (e.g., the Alpine library) results in the SLCSEs forming a part of each application to which the library is linked. (Ex. 1001, 2:14-15 (SLCSEs “form a part of at least some of the software applications by being linked thereto”).)

90. To the extent Ely does not inherently disclose this limitation, it would have been obvious in view of Levine. Levine explains how shared libraries work with dynamic linking. A dynamic linker “finds the names of the libraries required by [an application]” and maps the libraries’ contents into the application. (Ex. 1005

(Levine), 211-12.)² Once a shared library has been mapped into an application, its contents (e.g., executable code and data) are loaded into the application’s address space. (*Id.*, 212-13, 190 (“Each shared library occupies a fixed piece of address space in each program in which it is used.”).)

91. The dynamic linking process causes the shared library to form part of the application as described in the ’058 patent and as the applicant explained during prosecution. (Ex. 1001, 2:12-15 (“CSEs stored in the shared library ... form a part of at least some of the software applications by being linked thereto”); Ex. 1008, 91 (1/15/2010 response) (“In other words, the SLCSEs literally form part of the application. SLCSEs reside in the same address space as application code ...”).) Thus, Ely’s disclosure of dynamic linking inherently satisfies this limitation, and renders the limitation obvious in view of Levine’s explanation of how dynamic linking worked.

² The terms “program” and “application” are synonymous in the context of computer software.

e. 1[d]: CSE Provided to First and Second Applications

i. 1[d][1]: First Instance of SLCSE Provided to First Application

92. Claim 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.” As discussed above, Ely discloses a shared library containing SLCSEs. (¶¶ 82-85, above.) When an application accesses a shared library, an instance of any shared object used by the application (e.g., an instance of a SLCSE in the Alpine library) runs in the context of the linked application. (¶¶ 40-44, above (background on shared libraries); *see also* Ex. 1008, 23 (2/18/2009 response) (“It should be noted that a SLCSE in the context of an application is, for example, TCP IP under the request/control of an application.”).)

93. Although library code is shared between applications, each *instance* of a library object runs “without being shared” because each application gets a unique copy of private data. (¶¶ 40-44 (background on shared libraries); Ex. 1009 (Banerji), 60-61; Ex. 1013 (Sun Linker and Libraries Guide), 100-109; Ex. 1010 (Unix System V Release 3 Programmer’s Guide), 8-10, 8-13, 8-29.) The ’058 patent confirms this understanding of how conventional shared libraries worked and were used in the alleged invention:

In accordance with this invention, the code [is shared] 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 *Each application has its own unique data space.* This indivisible data space ensures that CSEs are unique to an application[.] ... Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. *In this manner CSEs are not shared among applications even though the code is shared.*

(Ex. 1001, 3:30-45 (emphasis added).) Accordingly, a person of ordinary skill in the art would understand that Ely discloses an instance of a SLCSE (e.g., a networking element in the Alpine library) running in the context of an application without that instance being shared.

94. This limitation also would have been obvious in view of Levine, which explains how shared libraries were implemented in conventional operating systems. Levine discloses that shared libraries are separated into segments, as shown in Figure 9.2 below:

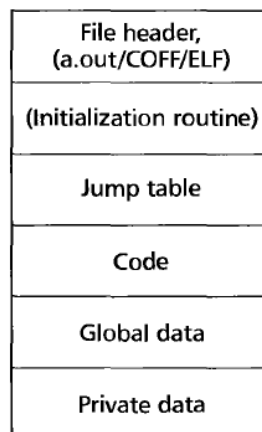


FIGURE 9.2 • Structure of typical shared library.

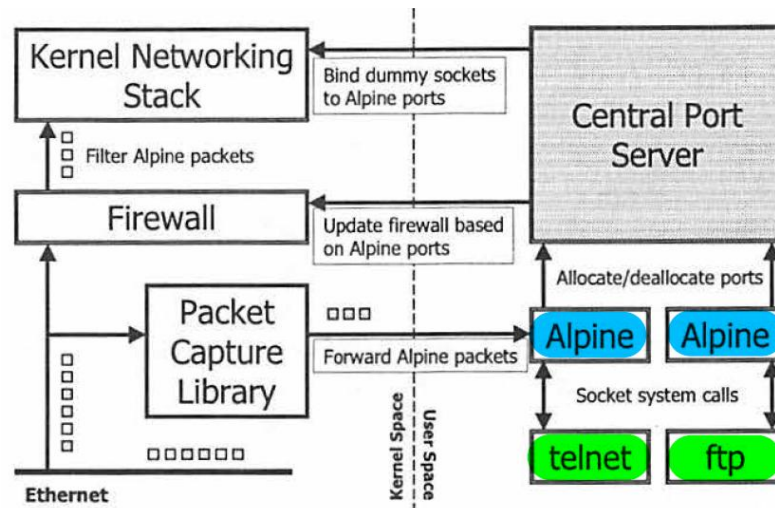
(Ex. 1005 (Levine), Fig. 9.2.) These segments are mapped to the address space of each application that accesses the shared library so that the code is executed in each application's own address space. (Ex. 1005 (Levine), 39, 212-13, 220.) Unique private data segments are provided to each application that links to a shared library. (Ex. 1005 (Levine), 38-39 (private writable data is mapped "copy-on-write"), 40 (data pages of a shared library mapped copy-on-write are not shared), 169-172 (each application has its own copy of private data).)

95. The '058 patent uses shared libraries in the same way Levine describes. (Ex. 1001, 3:30-45 ("In this manner CSEs are not shared among applications even though the code is shared").) Accordingly, Ely discloses this limitation and renders it obvious in view of Levine.

**ii. 1[d][2]: Unique Instance of
Corresponding CSE Provided to Second
Application**

96. Claim 1 recites "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." As discussed above, each application that links to a dynamic library gets a unique instance of any shared object it uses from the library. (¶¶ 92-95, above.) So, when a second application links to the Alpine library, it receives a unique instance of a CSE for performing the same function (e.g., a networking function) as the first application.

This scenario is depicted in Ely’s Figure 3, which shows two applications (“telnet” and “ftp”) both using Alpine:



(Ex. 1003 (Ely), 176 (highlighting added).) Thus, Ely discloses this limitation and renders the limitation obvious in view of Levine.

f. 1[e]: Two Instances of SLCSE Run Simultaneously

i. 1[e][1]: SLCSE Related to a Predetermined Function

97. Claim 1 further recites “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.” Ely discloses providing SLCSEs related to predetermined functions—for example the functions required to implement a networking stack. (¶¶ 82-85, above.) Any networking service that an application accesses via Alpine would qualify as a first instance of an SLCSE provided to a first application. (E.g., Ex. 1003 (Ely), 180 (describing “TCP” service that “telnet”

application accesses via Alpine); *see also* ¶¶ 92-95, above (Ely discloses instances of SLCSEs.) Thus, Ely discloses this limitation.

ii. 1[e][2]: Second Instance of SLCSE

98. Claim 1 further recites “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.” As discussed above, when a second application links to the Alpine library, a SLCSE for performing a same function (e.g., a networking function) is provided to that second application for running a second instance of the SLCSE. (¶ 96, above.) And both applications run Alpine’s networking stack simultaneously as shown in Figure 3 of Ely, for example. (Ex. 1003 (Ely) at 176 (“telnet” and “ftp” applications).) Thus, Ely discloses this limitation or renders it obvious in view of Levine’s explanation of shared libraries. Accordingly, claim 1 would have been obvious a person of ordinary skill in the art.

2. Claim 2

99. Claim 2 depends on claim 1 and further recites “wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.” As discussed above, Ely discloses that multiple instances of an SLCSE run simultaneously within the operating system—e.g., when more than one application uses the Alpine library on one computer. (¶ 97, above.) Thus, Ely discloses claim 2 or at least renders it obvious in view of Levine.

3. Claim 3

100. Claim 3 depends on claim 1 and further recites “wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.” Ely discloses this limitation.

101. Ely describes the use of networking SLCSEs to “provide[] a service that is also provided by the kernel.” (Ex. 1003 (Ely), 176.) Ely’s system is implemented “on an unmodified legacy operating system” that retains the kernel networking stack. (*Id.*, 172, Fig. 1 (showing a “traditional network stack” in the operating system kernel); ¶¶79-81, above.) Thus, Ely discloses OSCSEs (e.g., networking elements in the kernel) corresponding to and capable of performing the same functions as SLCSEs (e.g., networking elements in the Alpine library). (*Id.*) Accordingly, Ely discloses claim 3 or at least renders it obvious in view of Levine.

4. Claim 4

102. Claim 4 depends on claim 1 and further recites “wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.” Ely discloses this additional limitation.

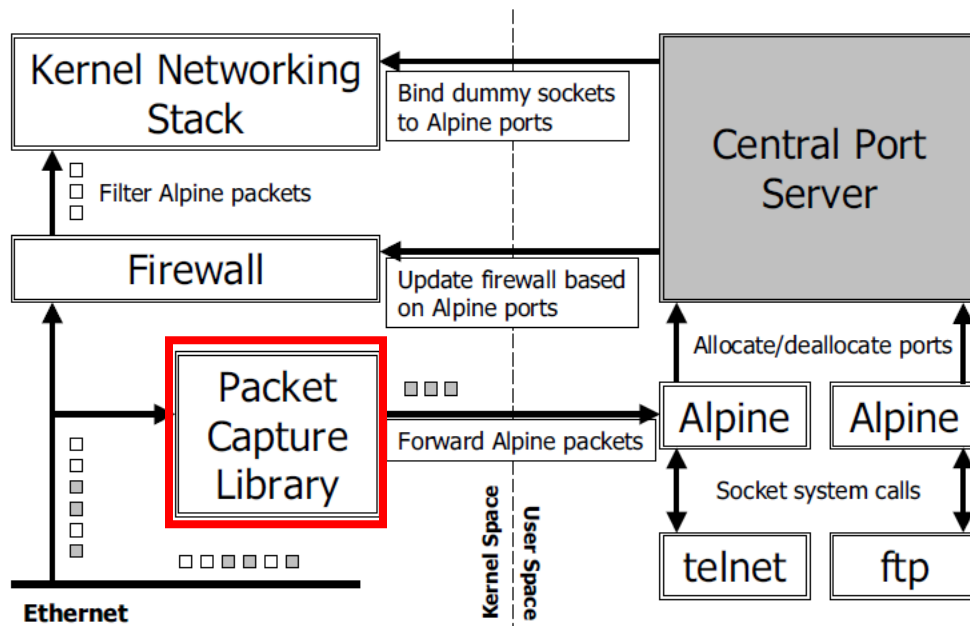
103. As discussed above, the individual instance of an SLCSE that is provided to an application is not shared. (¶¶ 92-95, above.) Thus, the application has “exclusive use thereof” as claim 4 requires.

104. Ely discloses that in some circumstances, its networking SLCSE “must make a *select* call into the kernel” of the operating system. (Ex. 1003 (Ely), 177.) The *select* call is a system call in FreeBSD 3.3 and similar operating systems. (Ex. 1023 (FreeBSD System Calls Manual).) Thus, Ely discloses claim 4 or at least renders it obvious in view of Levine.

5. Claim 5

105. Claim 5 depends on claim 1 and further recites “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.” The ’058 patent defines a “kernel module” as a “set of functions that reside and execute in kernel mode as extensions to the operating system kernel.” (Ex. 1001, 6:56-57.) Kernel modules were well-known and implemented in many conventional systems. The ’058 patent admits that kernel modules were “common” in conventional systems. (*Id.*, 6:57-59.) The ’058 patent discloses a kernel module that allows CSEs in user mode to communicate with devices operating in kernel mode. (Ex. 1001, 9:60-64, 10:35-43.)

106. Ely discloses the kernel module recited in claim 5. Specifically, Ely discloses a kernel-mode “Packet Capture Library” that serves as an interface between an ethernet device driver and Alpine’s SLCSEs:



(Ex. 1003 (Ely), Fig. 3.) The packet capture library is “adapted to serve as an interface” as claim 5 requires because it receives packets from the ethernet driver and forwards them to Alpine’s networking stack. (Ex. 1003 (Ely), Fig. 3; *id.* at 173-74.) Thus, Ely discloses claim 5 or at least renders it obvious in view of Levine.

6. Claim 6

107. Claim 6 depends on claim 5 and further recites two “wherein” clauses.

a. 6[a]: Kernel Provides Notification of an Event.

108. Claim 6 recites: “wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program[.]” Ely discloses that its kernel module (the packet capture library) provides notification of an event (the arrival of incoming packets) to an SLCSE (in Alpine’s networking stack) running in the context of an application program. Specifically,

Alpine uses an “interrupt handler” to respond to incoming packets. (Ex. 1003 (Ely), 174.) The interrupt handler polls “libpcap” (the packet capture library) to see whether a packet has arrived and, if so, sends the packet to Alpine’s networking stack for processing. (*Id.*)

b. 6[b]: Event is Asynchronous and Outside of the Application.

109. Claim 6 further recites: “wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.” Ely discloses this limitation because receiving packets from the network is an asynchronous event and information about a packet’s arrival comes from the kernel—outside the application. (Ex. 1003 (Ely), 174 (“packets arrive asynchronously” and the “kernel receives an interrupt whenever a packet arrives”).) Thus, Ely discloses claim 6 or at least renders it obvious in view of Levine.

110. Ely also discloses an alternative design whereby, rather than polling libpcap, the authors “could have modified the kernel to forward [packet-arrival interrupts] to Alpine using signals.” (Ex. 1003 (Ely), 174.) The authors did not implement this alternative design in Alpine but they disclosed it nonetheless. (Ex. 1003 (Ely), 174.) For this second independent reason, Ely discloses claim 6 or at least renders it obvious in view of Levine.

7. Claim 9

111. Claim 9 depends on claim 2 and further recites “wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.” Ely discloses claim 9’s additional limitation.

112. As described in the ’058 patent, 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” and a function overlay “is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded.” (Ex. 1001, 8:62-9:8.) Ely discloses exactly this:

[B]y loading Alpine’s dynamic library before any other dynamic library, its networking stack is used instead of the kernel’s. ... The LD_PRELOAD environment variable allows the Alpine dynamic library to be loaded before any other library, which implies that Alpine’s networking stack will be used instead of the kernel’s.

(Ex. 1003 (Ely), 177; *see also id.*, 178 (“To integrate with unmodified applications, Alpine ... exploit[s] properties of the linker, which allows Alpine to override any system call or library routine[.]”).) Thus, Ely discloses claim 9 or at least renders it obvious in view of Levine.

8. Claim 10

113. Claim 10 depends on claim 2 and further recites “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.” Ely discloses this limitation.

114. Specifically, Ely discloses that “by linking [an application] with the Alpine library before other libraries, Alpine’s networking stack is used instead of the kernel’s.” (Ex. 1003 (Ely), 177.) Alpine uses “dynamic linking” to accomplish this; the Alpine library is “dynamically linked” with each application that uses it. (Ex. 1003 (Ely), 177.) Dynamic linking provides an application with unique access to a unique instance of a shared library’s contents (e.g., networking SLCSEs in Alpine). (¶¶ 40-44, 92-96, above.) Thus, Ely discloses claim 10 or at least renders it obvious in view of Levine.

9. Claim 11

115. Claim 11 depends on claim 2 and further recites “wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.” Ely discloses claim 11’s additional limitation.

116. In conventional operating systems, user-mode processes rely on kernel services for device access, interrupt delivery, and virtual memory mapping because user-mode processes are not permitted to perform these functions themselves. Ely discloses that Alpine was implemented on such conventional operating systems. (Ex.

1003 (Ely), 171 (“Alpine supports a FreeBSD networking stack on top of a Unix operating system.”), 178 (“Alpine has been fully implemented in the FreeBSD 3.3 operating system.”).) Thus, a person of ordinary skill in the art would understand that Ely’s user-mode SLCSEs relied on kernel services for the recited functions.

117. Specifically, Ely’s SLCSEs relied on kernel services for access to networking devices when packets were received over the network, as discussed above. (¶¶ 105-106, above.) Ely’s SLCSEs also utilized kernel services for interrupt delivery—for example, when the CPU switched between processes. Additionally, because the kernel controls virtual memory, Ely’s SLCSEs relied on kernel services for virtual memory mapping when, for example, they were mapped into an application’s address space during the dynamic linking process. (¶¶ 40-46, above.) Thus, Ely discloses claim 11 or at least renders it obvious in view of Levine.

10. Claim 12

118. Claim 12 depends on claim 1 and further recites “wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.” Ely discloses SLCSEs implementing a user-space networking stack including networking protocol processes. (¶¶ 82-85, above; Ex. 1003 (Ely), 171 (Alpine “moves network protocols into a user-level library”).) Thus, Ely discloses claim 12 or at least renders it obvious in view of Levine.

11. Claim 13

119. Claim 13 depends on claim 10 and further recites “wherein some SLCSEs are modified for a particular one of the plurality of software applications.” Ely discloses that the Alpine user-level networking stack “can be examined and changed just as any other part of the application source code” and that “Alpine provides per-application specialization.” (Ex. 1003 (Ely), 179, 172.) Thus, Ely discloses claim 13 or at least renders it obvious in view of Levine.

12. Claim 14

120. Claim 14 depends on claim 13 and further recites “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.” Ely discloses that its user-space Alpine networking stack (which contains the SLCSEs) can be application-specific. (§ 119, above.) Ely also discloses that while the Alpine networking stack runs in user space, an unmodified networking stack continues to run in the kernel. (Ex. 1003, 171 (“networking stack resides in the kernel”), 172 (no kernel changes), 180 (describing “telnet” application using Alpine while “web browser” used kernel networking stack).) The kernel networking stack contains CSEs. (§§ 79-81, above.) The CSEs in the kernel networking stack are platform-specific. The ’058 patent defines a “[c]ompute platform” as the “combination of computer hardware and a single instance of an operating system” (OS). (Ex. 1001,

6:29-30.) This definition suggests that a component may be “platform-specific” if it is (1) hardware-specific or (2) OS-specific. The patent supports this understanding by distinguishing “platform independent” libraries from code that is specific to a CPU or operating system. (Ex. 1001, 7:1-7.) Ely’s kernel networking stack is platform-specific in both the hardware and OS dimensions. The kernel stack is hardware-specific because it includes an “Ethernet Driver,” which controls network interface hardware. (Ex. 1003, 172.) Second, the kernel stack in Ely comes from FreeBSD 3.3. (*Id.*, 178-79.) This BSD networking stack is OS-specific because it differs from networking stacks in other operating systems such as Linux. (See Ex. 1001, 5:41-47 (distinguishing BSD stack from networking stack used by “Linux platform”).) Thus, Ely discloses claim 14 or at least renders it obvious in view of Levine.

13. Claim 16

121. Claim 16 depends on claim 1 and further recites “wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.” Ely and Levine render obvious claim 16 for the reasons explained above in connection with claim 1. (¶¶ 88-91, above.)

14. Claim 18

122. Claim 18 depends on claim 2 and further recites “wherein SLCSEs are not copies of OSCSEs.” Ely discloses this limitation. A primary purpose of the Alpine library is to support the development of “protocol modifications or new

network protocols.” (Ex. 1003 (Ely), 171.) These modifications and new protocols are necessarily different than the *unmodified* network protocols in the operating system kernel. Thus, Ely discloses or at least suggests that the SLCSEs in Alpine (modified networking services) are not copies of the OSCSEs in the kernel (unmodified networking services). (See Ex. 1003 (Ely), 172 (“Alpine provides per-application specialization”), 179 (“Alpine running TCP Daytona,” which is “a set of modifications” to TCP networking).) Thus, Ely discloses claim 18, or at least renders it obvious in view of Levine.

B. Claims 5-8 and 15 Would Have Been Obvious in View of Ely, Levine, and Thekkath

123. Claims 5-8 and 15 depend (directly or indirectly) from claim 1. As discussed above, Ely and/or Levine disclose each limitation of claim 1. (¶¶ 76-98, above.) Like Ely, Thekkath discloses a networking “protocol library” linked to an application in user space. (Ex. 1006 (Thekkath), 67.) Thekkath further discloses modifying an operating system’s kernel to include a “Network I/O Module,” which facilitates communication between the user-level protocol library and the network. (¶¶ 52-55, above.)

124. A person of ordinary skill in the art would have been motivated to combine the teachings of Thekkath and Ely (with or without Levine). Specifically, a person of ordinary skill in the art would have been motivated to replace Ely’s packet

capture approach (which passes network packets to the user-level networking library) with Thekkath's network I/O module (which performs the same function more efficiently) for the reasons explained below.

125. First, Ely states that its "approach is similar to that proposed in [Thekkath]," and cites Thekkath. (Ex. 1003 (Ely), 176, 183 (citation "[29]").)

126. Second, Second, a person of ordinary skill in the art would look to Thekkath because Ely and Thekkath address the same problems of slow development and debugging of networking protocols implemented in the kernel. (*E.g.*, Ex. 1003 (Ely), 171; Ex. 1006 (Thekkath), 64.)

127. Third, replacing Ely's packet capture with Thekkath's network I/O module would represent nothing more than a simple substitution of one known element (Thekkath's module) for another (Ely's packet capture) to obtain predictable results (accessing the network from a user-level protocol library).

128. Fourth, Ely expressly recognizes the potential benefits of modifying the kernel to facilitate user-level networking, and Thekkath discloses how to make such modifications. Ely avoided kernel modifications only because of a self-imposed "design constraint." (Ex. 1003 (Ely), 174.) The purpose of this constraint was to make deployment of the Alpine library easier for users. (*Id.*, 172.) However, Ely also disclosed that relaxing this constraint and allowing kernel modification could lead to higher performance. (*See id.* ("Since our design constraints include requiring no

kernel ... modifications, Alpine cannot achieve the high-performance of [alternative] systems”).) Based on this disclosure, a person of ordinary skill in the art interested in maximizing performance would have been motivated to modify the kernel instead of relying on Ely’s lower-performance packet-capture approach. Thekkath discloses how to implement such a kernel modification in the form of its Network I/O module. (Ex. 1006 (Thekkath), 67-69.) Thekkath also describes how its disclosed implementation offers improved performance. (Ex. 1006 (Thekkath), 70 (showing that Thekkath’s implementation achieved higher throughput than the earlier “Mach 3.0/UX” user-level networking implementation).)

129. Fifth, the combination represents using a known technique (signaling from a kernel module as in Thekkath) to improve a similar device and method (Ely’s user-level networking library) in the same way. The combination also applies a known technique (signaling from a kernel module as in Thekkath) to a known device and method (Ely’s user-level networking library) that is ready for improvement and yields predictable results.

130. Sixth, because Ely uses an approach similar to Thekkath’s approach to solve the same problems, a person of ordinary skill in the art would have had a reasonable expectation of success in making the combination. (Ex. 1003 (Ely), 176.)

131. The combination of Thekkath and Ely (with or without Levine) renders obvious claims 5-8 and 15 as explained below.

1. Claim 5

132. Claim 5 depends on claim 1 and further recites “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.” Thekkath in combination with Ely discloses this limitation. Thekkath discloses that its protocol library “is linked into [each] application” that uses it and “contains the code that implements the communication protocol” but cannot access device drivers such as the network device drivers directly. (Ex. 1006 (Thekkath), 67.) Instead, the protocol library relies on a network I/O module, which sends and receives data from the protocol library and functions “to provide efficient and protected access to the network by the libraries.” (*Id.*, 67-69, Fig. 2.) Thekkath’s network I/O module “is co-located with the network device driver” in the kernel. (*Id.*) These components are shown in Figure 2 of Thekkath:

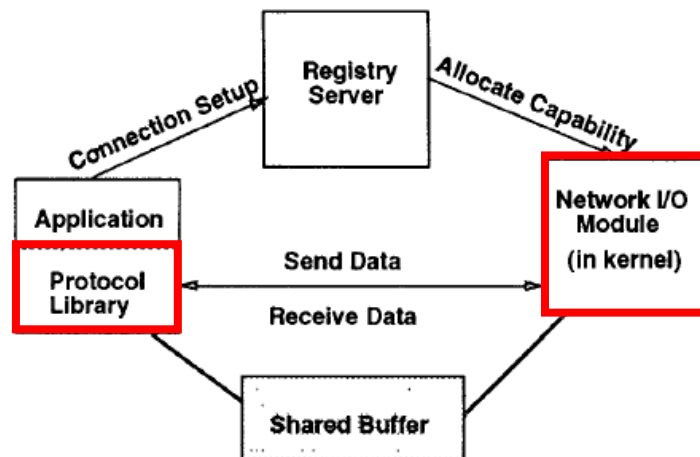


Figure 2: Structure of the Protocol Implementation

(*Id.*, Fig. 2 (annotated).)

133. As explained above, a person of ordinary skill in the art would have been motivated to combine Thekkath’s network I/O module with Ely’s Alpine library, as a replacement for Ely’s packet capture approach. (¶¶ 124-130.) In this combination, Thekkath’s module would serve as an interface to the networking SLCSEs in Ely’s Alpine library. (*Id.*) These SLCSEs run in the context of an application program as explained in connection with claim 1. (¶¶ 92-95, above.) Thus, claim 5 would have been obvious in view of Thekkath and Ely.

2. Claim 6

134. Claim 6 depends on claim 5 and recites two additional “wherein” clauses.

a. 6[a]: Kernel Provides Notification of an Event.

135. Claim 6 recites: “wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program[.]” Thekkath discloses that when the kernel’s network I/O module receives a packet, “a notification is sent to the application library via a lightweight semaphore.” (Ex. 1006 (Thekkath), 69; *see also id.*, 67.) The application library contains SLCSEs and runs in the context of an application program as explained above in connection with Ely. (¶¶ 92-95, above.)

b. 6[b]: Event is Asynchronous and Outside of the Application.

136. Claim 6 further recites: “wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.” The event that triggers the notification in Thekkath is the receipt of a network packet. (Ex. 1006 (Thekkath), 68-69.) Receipt of a network packet is asynchronous. (Ex. 1003 (Ely), 174-75 (“processing of incoming packets is asynchronous”).) Thekkath’s system requires information about this asynchronous event to be passed to the networking stack (SLCSEs) from outside of the application. (Ex. 1006 (Thekkath), 68-69 (“[i]ncoming packets from the network are moved into the shared [memory] region” where the application-level network protocols can access them).) Thus, claim 6 would have been obvious in view of Thekkath and Ely.

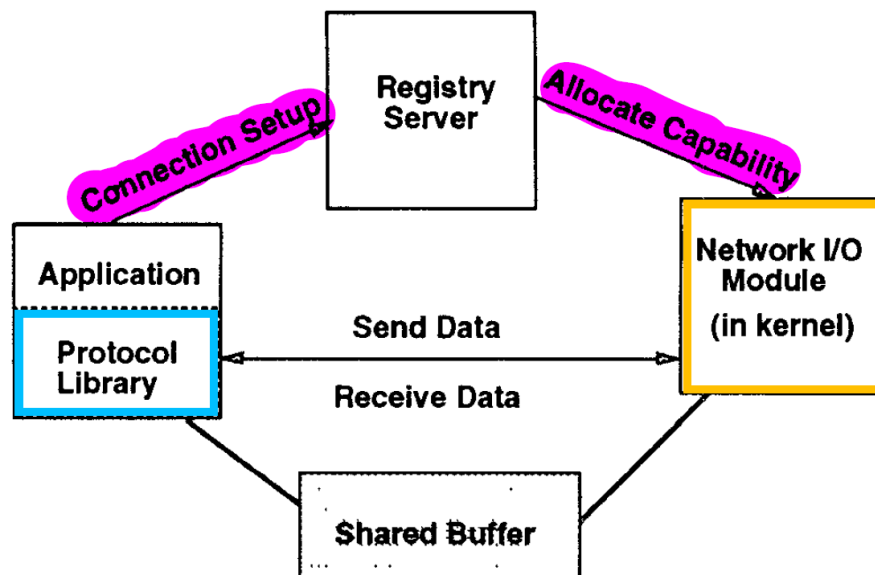
3. Claim 7

137. Claim 7 depends on claim 6 and further recites “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.” The ’058 patent defines an up call mechanism as a “means by which a service in kernel mode executes a function in a user mode application.” (Ex. 1001, 6:60-61.) Thekkath discloses the use of an upcall mechanism in the form of a “semaphore” to provide “network packet arrival notification” from the kernel to the user-level network stack. (Ex. 1006 (Thekkath),

68 (referring to user-level code being “upcalled”).) This notification causes the user-level network stack (running in the context of a user-mode application) to perform the function of responding to the packet. (*Id.*) Thus, claim 7 would have been obvious in view of Thekkath and Ely.

4. Claim 8

138. Claim 8 depends on claim 7 and further recites “wherein the upcall mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.” Thekkath discloses claim 8’s additional limitation. Specifically, Figure 2 of Thekkath shows that networking SLCSEs in the [protocol library](#) provide “[connection setup](#)” instructions to a registry server, which then configures the [kernel’s network I/O module](#):



(Ex. 1006 (Thekkath), Fig. 2; *id.* at 69.) Because the network I/O module is in the kernel, a person of ordinary skill in the art would understand that it executes the

configuration instructions it receives in kernel mode. (Ex. 1006 (Thekkath), 69 (library cannot access incoming packets until it provides connection-setup instructions that are used to configure the network I/O module); *see also id.* at Fig. 2, 67-68.)

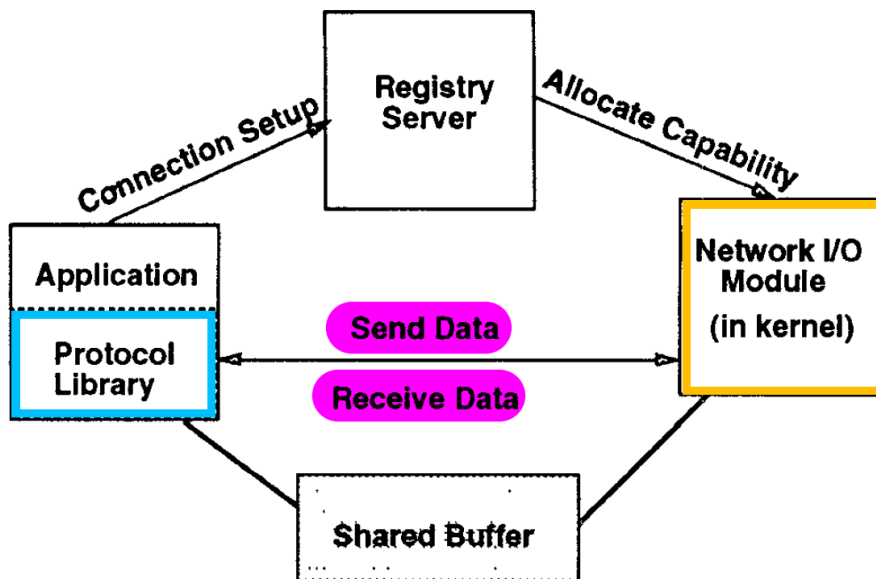
Thus, claim 8 would have been obvious in view of Thekkath and Ely.

5. Claim 15

139. Claim 15 depends on claim 5 and recites two additional “wherein” clauses.

a. 15[a]: Kernel Module Enables Data Exchange

140. Claim 15 recites: “wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode[.]” Thekkath discloses that the **kernel’s network I/O module** and the user-space SLCSEs in the **protocol library** can **send and receive data** to and from each other:



(Ex. 1006 (Thekkath), Fig. 2.) Thekkath’s network I/O module is “located with the in-kernel network device driver” and is used “to provide efficient and protected access to the network” from the user-mode library. (Ex. 1006 (Thekkath), 69.)

b. 15[b]: Data Exchange Uses Virtual Memory Mapping.

141. Claim 15 further recites: “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 SLCSE in user mode.” Thekkath discloses that data is exchanged back and forth using a “virtual memory region in the library [that] is mapped shared with the particular I/O module for the network device that the connection is using.” (Ex. 1006 (Thekkath), 68; *id.* (“This shared memory region is used to convey data between the protocol and the network device.”).) Thus, claim 15 would have been obvious to a person of ordinary skill in the art in view of Ely and Thekkath (with or without Levine).

C. Claims 1-4, 9-12, 16, and 18 Would Have Been Obvious In View of Eggert and Levine

142. Eggert discloses a dynamically linked library that extends a computer’s default file system with added capabilities. (¶ 56, above.) A person of ordinary skill in the art would have understood that Eggert’s dynamically linked library inherently worked in accordance with Levine’s disclosure of how dynamically linked libraries

were conventionally implemented. Indeed, Eggert specifically discloses that its library was “implemented atop SunOS 4.1.x.” (Ex. 1004 (Eggert), 237.) As Levine explains, the “SunOS” operating system “introduced dynamic shared libraries to UNIX in the late 1980s.” (Ex. 1005 (Levine), 206.) Because Eggert was published in 1993, it used a version of SunOS that supported such dynamically linked libraries. (Ex. 1029 (Australian UNIX User Group Newsletter), 37 (SunOS 4 release announcement describing dynamically linked shared libraries).)

143. Even if Eggert did not inherently use the sort of shared libraries that Levine describes, combining these references would have been obvious to a person of ordinary skill in the art. A person of ordinary skill in the art would have been motivated to implement Eggert’s dynamically linked library in accordance with Levine’s disclosure for several reasons, as explained below.

144. First, Eggert expressly teaches linking its library to applications using dynamic linking technology that was widely available. (Ex. 1004 (Eggert), 229 (“The recent rise of dynamic linking provides a new way for users to develop their own file systems”), 235 (“any dynamically linked program automatically incorporates IFS1”).) Based on this teaching, a person of ordinary skill in the art would have looked to Levine for an explanation of how such linking was conventionally implemented.

145. Second, Eggert disclosed a goal of making the work of “[c]hanging file systems” and “writing [] new file system[s]” simpler than previous approaches, which

required kernel modifications. (Ex. 1004 (Eggert), 229.) This disclosure would have motivated a person of ordinary skill in the art to look to Levine, which discloses that its “[d]ynamically linked shared libraries are easier to update than statically linked shared libraries.” (Ex. 1005 (Levine), 205.)

146. Third, because the conventional shared libraries in Levine were widely known and supported (*see* ¶ 72, above (dynamically linked shared libraries were conventional and routinely used in my own work)), using them in place of any other type of library would represent nothing more than a simple substitution of one known element (Levine’s conventional shared library) for another (any other type of library that Eggert could have used) to obtain predictable results (dynamic linking of the shared library). (Ex. 1005 (Levine), 205, 206, 217.) Moreover, the combination uses a known technique (dynamic linking as in Levine) to improve a similar device and method (a computer running Eggert’s library) in the same way. (*Id.*) The combination also applies a known technique (dynamic linking as in Levine) to a known device and method (a computer running Eggert’s library) that is ready for improvement and yields predictable results. (*Id.*)

147. Fourth, because Levine discloses that conventional shared libraries were widely supported and already available in common operating systems, a person of ordinary skill in the art would have had a reasonable expectation of success in using them to implement Eggert’s IFS1 library. (Ex. 1005 (Levine), 206, 217.)

148. Eggert and Levine disclose each limitation of claims 1-4, 9-12, 16, and 18 and render those claims obvious to a person of ordinary skill in the art.

1. Claim 1

a. 1: Preamble

149. The preamble recites “[a] computing system for executing a plurality of software applications.” Eggert discloses a file system extension used by applications running on a computing system. (Ex. 1004 (Eggert), 229, 237.)

b. 1[a]: Processor

150. Claim 1 recites “a processor.” Eggert discloses a system that utilizes “cycles” on “client processors,” and therefore discloses this limitation. (Ex. 1004 (Eggert), 235.) In addition, a person of ordinary skill in the art would have understood Eggert to implicitly disclose a processor as an essential component of conventional computing systems.

c. 1[b]: Operating System with Kernel and OSCSEs

151. Claim 1 next recites “an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.” The ’058 patent admits that this limitation was well known in the art. (¶ 79, above.)

152. Additionally, Eggert discloses that its file system extensions are “implemented atop SunOS 4.1.x,” a Unix-based operating system that includes a

kernel. (Ex. 1004 (Eggert), 237.) Eggert’s file system extensions improve on the traditional file system, located in the kernel. (Ex. 1004 (Eggert), 229 (“Changing [existing] file systems requires kernel modifications or root privileges”).) The kernel’s file system implementation contains OSCSEs. (Ex. 1001, 6:6-8 (defining CSEs), 6:11-22 (CSEs include file system implementations).) Thus, Eggert discloses, or at least suggests, this limitation.

d. 1[c]: Shared Library with SLCSEs

153. Claim 1 recites “a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.” Eggert discloses this limitation.

154. Eggert discloses “user-extensible file systems” that extend the kernel’s conventional file system. (Ex. 1004 (Eggert), 229.) One such file system, IFS1, extends the conventional file system to “permit certain shell-like expressions as pathnames” and to allow for “an escape mechanism for interpreting pathnames that can be modified by the user at any point.” (Ex. 1004 (Eggert), 229.) These file system extensions are within the scope of the ’058 patent’s definition of CSEs. (Ex. 1001, 6:11-15 (“CSEs include: ... File System services that offer extensions to those supplied by the OS[.]”).)

155. Eggert further discloses that these CSEs are implemented using a dynamically linked shared library that “works entirely in user space.” (Ex. 1004

(Eggert), 236 (IFS uses an approach that is “automatically adopted by all dynamically linked programs”), 235 (“any dynamically linked program automatically incorporates IFS1”), 237 (“dynamic linker” is used “to preload IFS1”).) Thus, Eggert discloses this limitation.

i. 1[c][1]: SLCSEs are Functional Replicas of OSCSEs

156. Claim 1 recites “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs.” The ’058 patent defines the term “replica” to mean “a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system (OS).” (Ex. 1001, 1:66-2:1.) The file system extensions in Eggert’s library are functional replicas of the kernel’s file system because they perform a similar function—accessing files using a pathname. (Ex. 1004 (Eggert), 231 (conventional file systems support “[e]xtensional pathnames” and Eggert extends this by also supporting “intensional [*sic*] pathnames, which include shell-evaluable expressions”), 233.) Because Eggert’s SLCSEs provide similar functionality to the kernel’s conventional OSCSEs, Eggert’s SLCSEs are “functional replicas” of the kernel’s OSCSEs. (Ex. 1001, 5:20-34, 8:27-40, 9:52-56.) Thus, Eggert discloses this limitation.

ii. 1[c][2]: SLCSEs Form a Part of the Application

157. Claim 1 further recites that some of the SLCSEs “are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.” Eggert discloses the use of dynamic linking to provide applications with access to the IFS1 library. (Ex. 1004 (Eggert), 229, 235, 237.) Because Eggert’s file system SLCSEs are dynamically linked, they form part of the applications to which they are linked. (See Ex. 1001, 2:14-15 (SLCSEs “form a part of at least some of the software applications *by being linked thereto*”) (emphasis added).)

158. To the extent that Eggert does not disclose this limitation, it is disclosed by Levine for the same reasons discussed above regarding Ely and Levine. (¶¶ 88-91, above.) Accordingly, Eggert and Levine disclose, or at least suggest, this limitation.

e. 1[d]: CSE Provided to First and Second Applications

i. 1[d][1]: First Instance of SLCSE Provided to First Application

159. Claim 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.” As discussed above, Eggert discloses a dynamically linked library containing SLCSEs. (¶¶ 151-155, above.) When an application accesses a dynamically linked library, an instance of each shared object used by the linked application (e.g., an instance of each SLCSE in Eggert’s IFS1 library) runs in the context of that application. (¶¶ 40-46, above (background on shared libraries and dynamic linking); *see also* Ex. 1004 (Eggert), 233-34 (describing “Sample IFS Applications”).) Although library code is shared between applications, each *instance* of a library object runs “without being shared” because each application gets a unique copy of private data for that object. (¶¶ 40-46, above (background on shared libraries and dynamic linking); Ex. 1001, 3:30-45 (explaining how conventional shared libraries worked and were used in the alleged invention).) Accordingly, Eggert discloses, or at least suggests, this limitation.

160. This limitation also would also have been obvious in view of Levine, which explains how shared libraries were implemented in conventional operating systems, for the reasons explained above. (¶¶ 92-95.)

ii. 1[d][2]: Unique Instance of Corresponding CSE Provided to Second Application

161. Claim 1 recites “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.” As discussed

above, each application that links to a dynamic library gets a unique instance of any shared object it uses from the library. (¶¶ 159-160, above.) So, when a second application links to Eggert’s IFS1 library, it receives a unique instance of a CSE for performing the same function (e.g., accessing files) as the first application. (Ex. 1004 (Eggert), 234 (two applications, “diff” and “grep,” each using IFS to access files.) Thus, Eggert discloses this limitation or at least renders it obvious in view of Levine.

f. 1[e]: Two Instances of SLCSE Run Simultaneously

i. 1[e][1]: SLCSE Related to a Predetermined Function

162. Claim 1 further recites “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.” Eggert discloses providing SLCSEs related to predetermined functions—for example the functions required to implement file-system extensions. (¶¶ 153-155, above.) Any file-system extension that an application accesses via IFS would qualify as a first instance of an SLCSE provided to a first application. (*E.g.*, Ex. 1004 (Eggert), 234 (file-versioning extension that “diff” application accesses via IFS); *see also* ¶¶ 159-160, above (explaining how Eggert discloses instances of SLCSEs).) Thus, Eggert discloses this limitation.

ii. 1[e][2]: Second Instance of SLCSE

163. Claim 1 further recites “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.” As discussed above, when a second application links to the IFS library, a SLCSE for performing a same function (e.g., a file-system function) is provided to that second application for running a second instance of the SLCSE. (¶ 161, above.) A person of ordinary skill in the art would know that running two applications at the same time was a very common activity in operating systems like SunOS (which Eggert used). When two applications with the same file-system functionality were in use at the same time, two instances of the SLCSE would run simultaneously as the claim requires.

164. Thus, claim 1 would have been obvious to a person of ordinary skill in the art in view of Eggert by itself, or in combination with Levine.

2. Claim 2

165. Claim 2 depends on claim 1 and further recites “wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.” As discussed above regarding claim 1, multiple instances of Eggert’s SLCSEs run simultaneously when a shared library is dynamically linked to multiple applications that are running at the same time. (¶¶ 163-164, above.) A person of ordinary skill in the art would have understood that running multiple

applications that access a file system at the same time was very common. Thus, claim 2 would have been obvious in view of Eggert by itself, or in combination with Levine.

3. Claim 3

166. Claim 3 depends on claim 1 and further recites “wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.” Eggert discloses that SLCSEs for evaluating pathnames are implemented in a shared library while OSCSEs for performing the same function—evaluating pathnames—remain in the kernel. (Ex. 1004 (Eggert), 233 (pathnames are evaluated first in the kernel and then in IFS1), 236 (“This approach requires no kernel modifications[.]”).) Thus, claim 3 would have been obvious in view of Eggert by itself, or in combination with Levine.

4. Claim 4

167. Claim 4 depends on claim 1 and further recites “wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.” Eggert discloses that a file system extension in IFS1 intercepts a system call, determines a new pathname, and then “retries the system call with the new pathname.” (Ex. 1004 (Eggert), 233, 236 (the user-space file system “operates as a layer between the actual system calls and the virtual [system] call”).) Eggert thus discloses that its

SLCSEs use system calls to access whatever kernel services the linked application requires using the pathnames generated by the SLCSEs.

168. Accordingly, claim 4 would have been obvious in view of Eggert by itself, or in combination with Levine.

5. Claim 9

169. Claim 9 depends on claim 2 and further recites “wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.” The ’058 patent explains 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.” (Ex. 1001, 8:62-64.) Function overlays may be used “to intercept software application accesses to operating system services.” (*Id.*, 8:65-67.) “The overlay is accomplished by [providing] a library to be preloaded before other libraries are loaded.” (*Id.*, 9:1-8.)

170. Eggert discloses that “a user can tell the dynamic linker to preload IFS1 before linking the standard C library[.]” (Ex. 1004 (Eggert), 237.) This preloading allows “redefinition of the basic system call interface in [the standard C library].” (*Id.*, 236.) As a result, IFS1 can intercept system calls from applications and provide extensions that change how the system calls are handled by the operating system. (Ex. 1004 (Eggert), 233, 236, 237.) Thus, claim 9 would have been obvious in view of Eggert by itself, or in combination with Levine.

6. Claim 10

171. Claim 10 depends on claim 2 and further recites “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.” Eggert discloses the use of dynamic linking which, as discussed above, links a shared library with an application as the application is loaded and provides each linked application with unique access to a unique instance of the shared library’s contents (e.g., SLCSEs). (¶¶ 40-46, above (background on shared libraries and dynamic linking); Ex. 1004 (Eggert), 229, 235-37.) Thus, claim 10 would have been obvious in view of Eggert by itself, or in combination with Levine.

7. Claim 11

172. Claim 11 depends on claim 2 and further recites “wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.” Eggert discloses and/or renders obvious this limitation. Eggert’s file system extension is implemented on the Unix-based SunOS operating system. (Ex. 1004 (Eggert), 237.) As with other conventional operating systems, user-mode processes in SunOS rely on kernel services for device

access, interrupt delivery, and virtual memory mapping. (¶¶ 115-119, above (discussing claim 11 in view of Ely and Levine).)

173. Eggert’s SLCSEs rely on kernel services for device access, for example, when using system calls to access files stored on disk. (Ex. 1004 (Eggert), 233.) Eggert’s SLCSEs also utilize kernel services for interrupt delivery and virtual memory mapping, like all user processes in conventional Unix-based operating systems. For example, Eggert’s SLCSEs rely on kernel services for interrupt delivery when the CPU switches between processes and for virtual memory mapping when SLCSEs are mapped into each application’s address space during the dynamic linking process. (¶¶ 115-119, above (discussing claim 11 in view of Ely and Levine).) Thus, claim 11 would have been obvious in view of Eggert by itself, or in combination with Levine.

8. Claim 12

174. Claim 12 depends on claim 1 and further recites “wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.” Eggert discloses SLCSEs related to management of files. (¶¶ 153-155, above; Ex. 1004 (Eggert), 230.) Thus, claim 12 would have been obvious in view of Eggert by itself, or in combination with Levine.

9. Claim 16

175. Claim 16 depends on claim 1 and further recites “wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.” As discussed above, Eggert discloses dynamically linking SLCSEs with applications, which causes SLCSEs to form a part of those applications. (¶¶ 153-158, 40-46 (background on shared libraries and dynamic linking).) Thus, claim 16 would have been obvious in view of Eggert by itself, or in combination with Levine.

10. Claim 18

176. Claim 18 depends on claim 2 and further recites “wherein SLCSEs are not copies of OSCSEs.” As discussed above, Eggert discloses SLCSEs that extend the functionality of the kernel’s file system rather than copying it. (¶¶ 153-156, above.) Thus, claim 18 would have been obvious in view of Eggert by itself, or in combination with Levine.

VII. SECONDARY CONSIDERATIONS OF NONOBVIOUSNESS

177. I am not aware of any secondary considerations of nonobviousness. If Patent Owner identifies any alleged evidence of secondary considerations in the future, I reserve the right to respond to that information.

VIII. CONCLUSION

178. For the foregoing reasons, it is my opinion that claims 1-16 and 18 of the '058 patent would have been obvious to a person of ordinary skill in the art at the time of the alleged invention in view of the prior art discussed above.

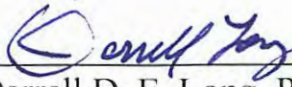
179. I reserve the right to supplement my opinions in the future to address or respond to any issues that the Patent Owner may raise, as well as new information including, but not limited to, any claim constructions advanced by the Patent Owner or adopted by the Board in the Institution Decision, and respond to any alleged secondary considerations as they become available to me.

Amazon.com, Inc. v. VirtaMove, Corp.

Declaration of Dr. Darrell D.E. Long – U.S. Patent No. 7,784,058

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

Executed on January 29, 2025 at Saginaw, California.



Darrell D. E. Long, Ph.D.