

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTERNATIONAL BUSINESS MACHINES CORP.,

Petitioner,

v.

VIRTAMOVE, CORP.,

Patent Owner.

Case IPR2025-00591

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

**DECLARATION OF TODD C. MOWRY, PH.D.
IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 7,784,058**

TABLE OF CONTENTS

I.	Introduction.....	1
II.	Background and Qualifications	2
III.	Documents and Materials Considered.....	6
IV.	Legal Principles	7
V.	Person of Ordinary Skill in the Art.....	9
VI.	Technology Background.....	10
	A. The Operating System	11
	B. CPUs, Kernel Mode, and User Mode.....	11
	C. Applications, Threads, and Processes	13
	D. Linking and Shared Libraries	14
	E. Networking Services	17
VII.	Background of the '058 Patent	18
	A. Prosecution History of the '058 Patent	21
	B. Priority Date of the Challenged Claims	21
VIII.	Background of the Prior Art	22
	A. Deianov.....	22
	B. Ely.....	24
	C. Levine.....	26
	D. Sands.....	29
IX.	Claim Construction.....	30
X.	Invalidity Opinions: '058 Patent.....	33
	A. Ground 1: Claims 1-4 and 18 Are Rendered Obvious By Deianov, Or Alternatively Obvious In View Of Deianov In Combination With Levine And/Or Sands	33
	1. Motivation to Combine	33
	2. Claim 1	41
	3. Claim 2: “A computing system as defined in claim 2, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.”	62
	4. Claim 3: “A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.”	63
	5. Claim 4: “A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use	

	thereof, use system calls to access services in the operating system kernel.”.....	63
6.	Claim 18: “A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.”.....	64
B.	Ground 2: Claims 1-4 and 18 Are Rendered Obvious By Deianov, Or Alternatively Obvious In View Of Deianov In Combination With Levine And/Or Sands	65
1.	Motivation to Combine	65
2.	Claim 1	71
3.	Claim 2: “A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.”.....	86
4.	Claim 3: “A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.”.....	88
5.	Claim 4: “A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.”.....	89
6.	Claim 18: “A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.”.....	89
XI.	Secondary Considerations Of Non-Obviousness	91

I, Dr. Todd C. Mowry, declare as follows:

I. INTRODUCTION

1. My name is Todd C. Mowry.

2. I have been retained as an expert witness on behalf of International Business Machines Corp. (“IBM” or “Petitioner”) for the above-captioned Petition for *inter partes* review (the “Petition”) of U.S. Patent No. 7,784,058 (the “’058 Patent”).

3. I understand that IBM is challenging the validity of claims 1, 2, 3, 4, and 18 (the “Challenged Claims”) of the ’058 Patent in the Petition.

4. I have been asked to consider whether certain references invalidate the Challenged Claims.

5. I am being compensated for my time in connection with the Petition at my standard consulting rate of \$650 per hour. My compensation is not affected by the content of my opinions, my testimony, or the outcome of the Petition.

6. I am not currently an employee of IBM. I was a Visiting Scientist at IBM in Toronto from 1996 to 2004. I have no financial interest in IBM.

7. The prior art references on which my opinions are based are:

- U.S. Patent No. 6,529,985 to Borislav Deianov, *et al.* (“Deianov”)
- *Alpine: A User-Level Infrastructure for Network Protocol Development* by David Ely, *et al.* (“Ely”)
- *Linkers & Loaders* by John R. Levine (“Levine”)

- U.S. Patent No. 5,291,601 to Samuel Sands (“Sands”)

8. I also rely on textbooks and other documents as evidence of the state of the art prior to the priority date of the '058 Patent, including books such as *Modern Operating Systems* (Ex-1011), *An Introduction to Operating Systems* (Ex-1012), and the *Microsoft Computer Dictionary* (Ex-1013).

9. In my opinion, the Challenged Claims are invalid as obvious based on (1) Deianov alone or in combination with Levine and/or Sands; and (2) Ely alone or in combination with Levine and/or Sands.

II. BACKGROUND AND QUALIFICATIONS

10. I am a Professor in the Department of Computer Science at Carnegie Mellon University. I also have a courtesy appointment in the Department of Electrical and Computer Engineering. I have served on the faculty of Carnegie Mellon University for twenty-eight years from 1997 through the present (2025).

11. I also served on the faculty of the University of Toronto for four years between 1993 and 1997, in the Department of Electrical and Computer Engineering and with a courtesy appointment in the Department of Computer Science. Prior to that appointment, I served as a Graduate Research Assistant in the Department of Electrical Engineering at Stanford University for four years between 1989 and 1993.

12. Over the years, I have taught and continue to teach many courses and directed research in computer systems and software, operating systems, distributed and network systems, parallel processing, and compiler optimization.

13. I received a B.S. in Electrical Engineering with Highest Distinction from the University of Virginia in May 1988. I received an M.S. in Electrical Engineering from Stanford University in June 1989, and a Ph.D. in Electrical Engineering from Stanford University in March 1994.

14. I have worked in the computer industry in various capacities over the years. I was a part-time Computer Architect and then Computer Architecture Consultant at Silicon Graphics, Inc. in Mountain View, California (formerly MIPS Computer Systems in Sunnyvale, California) from 1989 to 1993 and 1993 to 1996, respectively. I was a Visiting Scientist at IBM in Toronto from 1996 to 2004. During that same time period (1996 to 2004), I was also a Member of the Technical Advisory Board of SandCraft, Inc. in Santa Clara, California. I was the Director of the Intel Research Pittsburgh Lab at Intel Corporation in Pittsburgh, Pennsylvania from 2004 to 2007.

15. I have authored twenty-one (21) journal articles and seventy-two (72) conference papers, all of which were peer reviewed. I am also a named inventor on five patents.

16. I have published a number of papers in the top research conferences in operating systems and data storage (including a paper that won the Best Paper Award at the USENIX 2nd Symposium on Operating Systems Design and Implementation (OSDI '96)). *See, e.g.,* Olatunji Ruwase, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry, *Guardrail: High Fidelity Correctness Checking of Device Drivers for Safeguarding I/O Operations*, in Proceedings of the Nineteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2014); Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang, *Self-Driving Database Management Systems*, in Proceedings of the 2017 Conference on Innovative Data Systems Research (CIDR) (2017); and Amit Manjhi, Charles Garrod, Bruce M. Maggs, Todd C. Mowry, Anthony Tomasic, *Holistic Query Transformations for Dynamic Web Applications*, in 2009 IEEE 25th International Conference on Data Engineering (ICDE) (2009).

17. I have also been an active member of the Parallel Data Lab at Carnegie Mellon University since 1997, which is often described as “academia’s premiere storage systems research center.” Ex-1032, 1.

18. I am the recipient of several honors and awards: the Arthur Samuel Thesis Award (awarded by the Stanford Computer Science department to the top two Ph.D. theses in a given year), several IBM Faculty Development Awards (1996, 1997, 1998, 2000, 2001, 2002, and 2003), several Best Paper Awards (the Second Symposium on Operating Systems Design and Implementation in 1996; the 20th International Conference on Data Engineering (ICDE) in 2004), the Alfred P. Sloan Research Fellowship (awarded to researchers in recognition of distinguished performance and a unique potential to make substantial contributions to their field), the Most Thought-Provoking Idea Award in 2004 (awarded by the Architectural Support for Programming Languages and Operating Systems (ASPLOS)), the TR100 Award in 1999 (awarded by MIT's Technology Review magazine to the top 100 most promising young innovators in science and technology), and I became an ACM Fellow in 2016.

19. I am a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Association of Computing Machinery (ACM). I was the Editor-in-Chief of ACM Transactions on Computer Systems from 2013 through 2018, which is the premier journal for computer systems research. (I was an Associate Editor for the journal prior to that, since 2001.) I was the Program Chair of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) in 2010. I was the Co-Program Chair of the International

Conference on Parallel Architectures and Compilation Techniques (PACT) in 2001.

I have been on the programming committee in various years for ASPLOS, the International Symposium on Computer Architecture (ISCA), the International Symposium on Microarchitectures, and the Workshop on Architectural and System Support for Improving Format.

20. Overall, I have over thirty years of experience in the field of computer science.

21. My qualifications and publications are set forth more fully in my curriculum vitae, attached as Ex-1004.

III. DOCUMENTS AND MATERIALS CONSIDERED

22. In forming the opinions expressed in this Declaration, I relied upon my education, training, knowledge, and experience in the relevant field of the art, and I have considered the viewpoint of a person having ordinary skill in the art (POSITA) at the time of the alleged invention.

23. In preparing this Declaration, I have reviewed the materials referenced herein, including the '058 Patent (Ex-1001), the file history of the '058 Patent (Ex-1002), the prior art references, the Petition, references which reflect the state of the relevant art at the time of the alleged invention and the background knowledge of a POSITA, and other documents listed in the Exhibit List of the Petition.

IV. LEGAL PRINCIPLES

24. IBM's counsel has informed me of certain legal standards. I have applied these understandings in my analysis as detailed below. I am not an attorney and offer no legal opinions.

25. I have been informed and understand that a patent claim is invalid if it is obvious, and that obviousness requires that the claim be obvious from the perspective of a person of ordinary skill in the relevant art at the time of the alleged invention. I have been informed and understand that a claim may be obvious in view of a single prior art reference, or may be obvious in view of a combination of two or more prior art references.

26. I have been informed and understand that the level of ordinary skill in the art may be determined by considering such factors as the education level of those working in the field, the sophistication of the technology, the types of problems encountered in the art, the prior art solutions to those problems, and the speed at which innovations are made. I have been informed and understand that not all of these factors need be considered and that one or more of these factors may control.

27. I have been informed and understand that an obviousness analysis is based on several factors, including the scope and content of the prior art, what differences, if any, existed between the alleged invention and the prior art, and the level of ordinary skill in the pertinent art at the time of the alleged invention.

28. I have been informed and understand that to determine whether a claim is obvious, it is appropriate to consider, among other factors:

- whether the teachings of the prior art references disclose known concepts combined in familiar ways, and when combined, would yield predictable results;
- whether a person of ordinary skill in the art could implement a predictable variation, and would see the benefit of doing so;
- whether the claimed elements represent one of a limited number of known design choices, and would have a reasonable expectation of success by those skilled in the art;
- whether a person of ordinary skill would have recognized a reason to combine known elements in the manner described in the claim;
- whether there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; and
- whether the innovation applies a known technique that had been used to improve a similar device or method in a similar way.

29. I further have been informed and understand that while it may be helpful to identify a reason for this combination, common sense should guide and no rigid requirement of finding a teaching, suggestion, or motivation to combine is required.

30. I further have been informed and understand that certain factors called secondary considerations may be considered in evaluating the obviousness of a claim. I understand that such secondary considerations include, among other things,

commercial success of the alleged invention, skepticism of those having ordinary skill in the art at the time of the alleged invention, unexpected results of the alleged invention, any long-felt but unsolved need in the art that was satisfied by the alleged invention, the failure of others to make the alleged invention, praise of the alleged invention by those having ordinary skill in the art, and copying of the alleged invention by others in the field. I have been informed and understand that there must be a nexus—a connection—between any such secondary considerations and the alleged invention. I also have been informed and understand that contemporaneous and independent invention by others is a secondary consideration tending to show obviousness.

V. PERSON OF ORDINARY SKILL IN THE ART

31. I understand that there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including the educational level of active workers in the field at the time of the alleged invention, the sophistication of the technology, the type of problems encountered in the art, and the prior art solutions to those problems.

32. In determining the characteristics of a hypothetical person of ordinary skill in the art (“POSITA”) of the ’058 Patent at the time of the alleged invention, I considered several things, including the type of problems encountered in this field and the rapidity with which innovations were made. I also considered the

sophistication of the technology involved, the educational background and experience of those actively working in the field, and the level of education that would be necessary to understand the '058 Patent. I also, based on my own experience, placed myself back in the relevant periods of time and considered the state of the art and level of skill of computer scientists working in this field in those times.

33. In my opinion, a POSITA in the subject matter of the '058 Patent would have had a bachelor's degree in computer science (or a similar degree) and at least two years of experience in computer software systems. Additional education could offset less experience and vice versa, and a POSITA could have also obtained similar knowledge and experience through other means.

34. I have been a POSITA since at least 1991. The opinions I am offering are from the perspective of a POSITA at the time of the alleged invention, which I understand to be the 2002-2003 timeframe. For the purposes of my opinions below, I have been asked to assume that the relevant priority date for the Challenged Claims is September 22, 2003, and have applied the understanding of a POSITA as of that date.

VI. TECHNOLOGY BACKGROUND

35. In this section, I provide a high-level overview of several aspects of computer architecture that underpin the '058 Patent and prior art references. As I

describe below, prior to the alleged invention of the '058 Patent, various technologies existed that disclosed the architecture and functionality claimed in the '058 Patent.

A. The Operating System

36. An operating system (OS) is software which manages computer resources, namely the hardware, and provides a set of abstractions for users to interact with various components of the hardware. *See, e.g.*, Ex-1011, 25; Ex-1012, 3.

37. Some examples of commonly used operating systems are UNIX (and its many variations, such as Linux) and Windows. Ex-1015, 3.

38. At the heart of every operating system is the kernel, which is a privileged operating system program that is responsible for tasks such as control of hardware and networking. Ex-1013, 5; Ex-1012, 8-9; Ex-1015, 6.

B. CPUs, Kernel Mode, and User Mode

39. A processor is a hardware unit in a computer that performs calculations, executes instructions, and manages data to enable a computer to function. Ex-1013, 6. Processors are one of the key resources managed by an operating system. Ex-1012, 4.

40. The CPU, or central processing unit, is the main processing hardware component that controls a computer system. Ex-1013, 4; Ex-1011, 45. The CPU is

controlled by the operating system. As it is hardware, a CPU is a physical processor, as opposed to a form of virtual processor. The CPU controls the computing system by retrieving instructions from memory, decoding them, and then executing those instructions. Ex-1011, 45; Ex-1013, 3. Each CPU has a specific set of instructions that it can execute. Ex-1011, 45.

41. Operating systems and CPUs typically have two modes of operation: kernel mode and user mode. Ex-1011, 46. At a high level, these modes determine the level of access to hardware resources. Ex-1011, 46-47.

42. When the operating system is executing in kernel mode, functions execute with unrestricted access to all features of the hardware. Ex-1011, 47. The operating system and the kernel run in kernel mode, and the protected memory region where the kernel resides is referred to as kernel space. *See, e.g.*, Ex-1011, 400.

43. When the operating system is executing in user mode, applications or functions running generally can execute with only restricted access to hardware. Ex-1011, 47. User mode is the mode in which user applications execute, and user space refers to the memory region outside of the kernel where user-mode programs execute. *See, e.g.*, Ex-1011, 400.

44. If a running program in user mode requires access to a privileged functionality, such as performing I/O operations or managing memory allocation,

the program generally must make a system call, a mechanism provided by the kernel that serves as an interface for user mode programs to request services from the operating system. Ex-1011, 47; Ex-1012, 4, 160; Ex-1015, 10-13. A system call switches the CPU from user mode to kernel mode, starting the operating system to execute the requested function. Ex-1011, 47. Once the requested function has been completed, the operating system returns control to the user program at the instruction immediately following the system call. Ex-1011, 47.

C. Applications, Threads, and Processes

45. Applications are a form of computer program designed to perform specific functions for users, such as word-processing. Ex-1013, 3. The term application is often used interchangeably or together with the term program, but specifically refers to a user mode program. *See, e.g.*, Ex-1013, 3.

46. Computer users often use multiple applications at the same time. *See, e.g.*, Ex-1011, 29. Two techniques that computer systems use to handle multiple applications running at the same time include processes and threads. *See, e.g.*, Ex-1011, 95-182, Ex-1014, 4; Ex-1015, 14. As their name might suggest, processors execute processes and threads. Ex-1015, 14-16.

47. A process is an instance of a running program that includes the program's code, data, and system resources needed for execution. Ex-1015, 14. An

application consists of one or more processes working together. *See, e.g.*, Ex-1015, 14-16.

48. Each process is allocated its own address space by the operating system, which it shares with its respective application. *See, e.g.*, Ex-1011, 95-104. The operating system maintains a separate mapping of virtual address spaces to physical address spaces for each individual process. *See, e.g.*, Ex-1011, 51, 226-229. The virtual address space gives each process the illusion that it has its own memory space and resources, ensuring isolation from other processes, even if the underlying physical memory is limited or fragmented. *See, e.g.*, Ex-1011, 51, 226-229.

49. Because each process operates within its own isolated memory space, one application will not be able to crash or corrupt the memory of others, as they cannot directly access or modify each other's memory. Ex-1011, 541. This ensures greater stability and security for the computer system as a whole. *See, e.g.*, Ex-1011, 541.

50. Each process contains at least one thread. Ex-1011, 105. A thread is the minimal set of register (special memory location) values needed to execute a particular invocation of an application. Ex-1011, 105-124.

D. Linking and Shared Libraries

51. To streamline development and execution, applications rely on linking and shared libraries. *See, e.g.*, Ex-1012, 7. This was the case since the 1980s, long prior to the '058 Patent.

52. Applications typically contain only a fraction of the routines and data that they require to perform a task. *See, e.g.*, Ex-1012, 7. The rest of the routines that they require to perform the task are generally supplied by shared libraries. Ex-1012, 7.

53. Shared libraries are a collection of prewritten routines or functions that are generally designed to be accessible to any application that requires their functionality. *See, e.g.*, Ex-1012, 7. Shared libraries were well-known and used by programmers long before the alleged priority date of the '058 Patent, with earlier forms of shared libraries being used as early as the mid-twentieth century. *See, e.g.*, Ex-1012, 7; Ex-1014, 3; Ex-1015; Ex-1016, 3-4; Ex-1019, 3; Ex-1020, 7 (26:8-16).

54. Library routines offer a way for applications to utilize common functions, such as I/O operations and mathematical calculations, without requiring programmers to write the code for these functions directly into each application. Ex-1012, 7; Ex-1017, 2. Instead, these routines are written once, stored in the shared libraries, and made available to applications through a process called linking. Ex-1017, 1-2.

55. Linking involves resolving library routine calls in an application to ensure that the application accesses the correct instructions for execution. *See generally* Ex-1017; Ex-1012, 7. Put simply, this is done by loading code from the shared libraries into the same address as the application, combining the library code with the application code, and determining the addresses to external functions or variables, to create an executable program. *See, e.g.,* Ex-1017; Ex-1012, 7.

56. By allowing multiple applications to share the same library routines, shared libraries minimize redundancies, as code does not need to be duplicated across multiple applications. *See, e.g.,* Ex-1017, 2. Shared libraries also save memory, since shared libraries can be loaded into memory once and used by several applications simultaneously. *See, e.g.,* Ex-1017, 2. Additionally, shared libraries streamline updates and maintenance, because updating a library automatically benefits all applications relying on it. *See, e.g.,* Ex-1017, 1-2; Ex-1015, 18.

57. In the earlier days of computer systems (before roughly 1989), linking was typically performed through “static linking,” where the actual code from library routines is copied directly into the program’s executable during program compilation, the process of turning human-readable source code into computer-executable object code. Ex-1017, 2. A notable disadvantage of static linking is that it duplicates library code into each executable, wasting system memory and storage. *See, e.g.,* Ex-1017, 2.

58. Starting in the late 1980s and early 1990s, both UNIX and Windows began supporting “dynamic linking,” where library routines are not linked to a particular application until runtime. Ex-1017, 2; *see also, e.g.*, Ex-1016, 4; Ex-1015, 17. More specifically, at runtime, the library routines are loaded into the processes’ virtual address spaces, and the dynamic linker locates the routines and loads them into the application’s memory space. Ex-1015, 17. Advantageously, dynamic linking allows storing a single copy of a library routine in physical memory, while each application runs its own instance of the library routine in user space. Ex-1017, 2; Ex-1015, 17.

59. In UNIX, dynamic shared libraries are commonly called ELF libraries (Executable and Linkable Format); in Windows, they are called dynamic link libraries (DLLs).

E. Networking Services

60. A networking stack, also known as a network stack or protocol stack, is a collection of networking communication services, or network protocols, implemented in software. *See, e.g.*, Ex-1021, 4, 7-8; Ex-1015, 19.

61. These network protocols provide the rules and mechanisms that enable different computers to communicate with one another. *See, e.g.*, Ex-1021, 1-2. Without these protocols, the Internet would not be able to function with its current scale and level of performance. *See, e.g.*, Ex-1021, 7-8.

62. Examples of network protocols include TCP, IP, and UDP. *See, e.g.*, Ex-1021, 7-8.

63. Network protocols are operating system-independent because they follow universal standards regardless of the operating system they are implemented in, ensuring interoperability across these different computers and seamless communications across a network. Ex-1021, 5-6, 10-12. However, their implementation in software may vary across operating systems due to system architecture differences across the operating systems in which they are implemented and the different performance goals of those operating systems. Ex-1021, 5-6, 10-12.

64. The networking stack is typically part of the operating system. Ex-1021, 9. Within a networking stack, protocols are normally developed in layers, with each layer handling a specific aspect of communication. Ex-1021, 7. The layers that handle communication details are usually implemented in the operating system kernel. Ex-1021, 9.

VII. BACKGROUND OF THE '058 PATENT

65. The '058 Patent is titled "Computing system having user mode critical system elements as shared libraries."

66. The '058 Patent relates to replicating operating system services, such as the network stack, into a user mode shared library for applications to utilize. Ex-

1001, Abstract. This means that these operating system services are available to applications outside the kernel. As I describe below, this is the same field of endeavor as the prior art references I have reviewed. *See* Section VIII.

67. The '058 Patent observes that computer system design results in various applications relying upon certain common resources, and that it is essential to safely and effectively provide those resources to applications. Ex-1001, 1:21-24. However, the '058 Patent notes that this centralized control of shared resources by the operating system can cause resource conflict issues if access to those shared resources is not properly managed. Ex-1001, 1:26-28.

68. The '058 Patent emphasizes that this is particularly true of resources or services critical to software applications, which it terms “critical system elements (CSEs).” Ex-1001, 1:25-27. The '058 Patent discloses that CSEs include network services, which are generally provided as a kernel service, and file system services that extend operating system functionality to user mode. Ex-1001, Fig. 1, 5:41-43, 6:6-28.

69. The '058 Patent refers to CSEs that are supplied by the operating system kernel and which run in kernel mode as operating system critical system elements (OSCSEs). Ex-1001, 1:60-61

70. The '058 Patent states that other architectural approaches in the art are inadequate because they continue to rely on centralized control of shared resources or present performance and operational weaknesses. Ex-1001, 1:35-45.

71. As its alleged solution, the '058 Patent describes the storage of CSEs into shared libraries so that applications may link to and utilize the CSEs stored in those shared libraries. Ex-1001, 5:29-34. These shared libraries are located outside of the kernel. The '058 Patent refers to these CSEs stored in shared libraries as “shared library critical system elements (SLCSEs).” Ex-1001, 2:10.

72. The '058 Patent discloses that SLCSEs are provided in user mode. Ex-1001, 3:19-24. As I describe above in the Technology Background, this means that the SLCSEs are located outside of the operating system kernel in user space. See Section VI.B. This architecture is further confirmed and illustrated in Figure 4 of the '058 Patent, shown below.

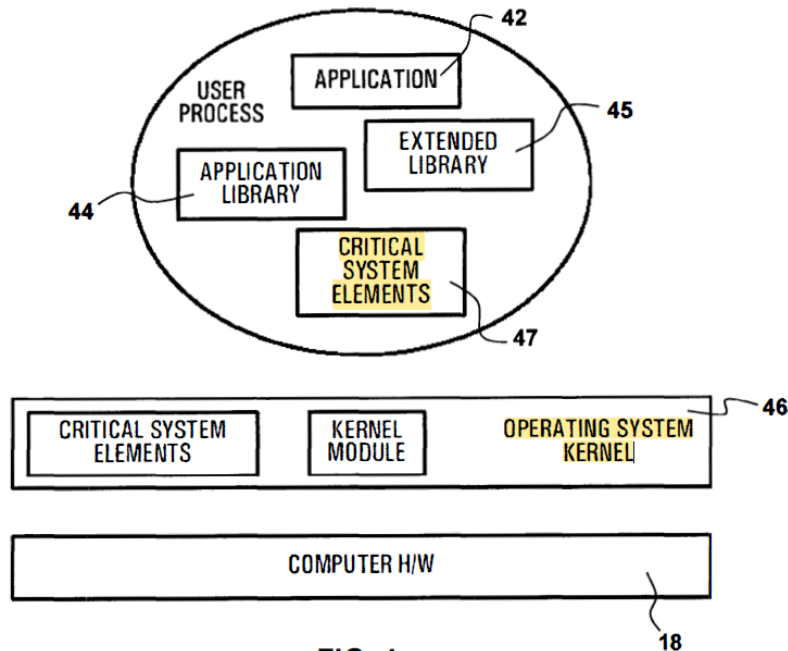


FIG. 4

Ex-1001, Fig. 4 (annotated).

73. The '058 Patent further discloses that SLCSEs form a part of software applications by being linked to them and are run in the context of those applications without the SLCSE being shared with other applications. Ex-1001, 2:14-21.

A. Prosecution History of the '058 Patent

74. I have reviewed the file history of the '058 Patent (Ex-1002). I have also reviewed the summary of the prosecution history in the Petition, and I agree with it.

B. Priority Date of the Challenged Claims

75. For the purposes of my analysis, I have been instructed to apply a priority date of September 22, 2003, for the Challenged Claims. I am not offering an

opinion as to the priority date the Challenged Claims are entitled to in this Declaration.

VIII. BACKGROUND OF THE PRIOR ART

A. Deianov

76. Deianov is a U.S. Patent that issued on March 4, 2003, and it is titled “Selective interception of system calls.” Like the ’058 Patent, it relates to taking operating system services and providing them to applications through a user mode alternative. Ex-1005, 1:22-24.

77. More specifically, Deianov discloses a computer system running a multitasking operating system that extends and expands operating system functionality through the interception of system calls. Ex-1005, 1:28-30, 2:3-5.

78. Deianov explains that system calls are subroutines with object code located in the operating system, such that the system calls run under control of the operating system in kernel mode. Ex-1005, 1:31-44. When a system call is executed, it provides a system operation such as reading a file, managing network connections, or allocating memory. *See id.*

79. Deianov’s system calls are critical to application function as they provide the interface for user mode applications to access the more privileged kernel mode functions that the applications need to execute certain tasks. Examples of

these functions include writing to files, creating new processes, and allocating memory.

80. Deianov explains that when a system call is intercepted, alternative object code, rather than the system call, is executed. Ex-1005, 1:64-2:2; 7:3-4. Deianov states that this alternative object code is known as a system call wrapper. Ex-1005, 2:2. In this way, system call wrappers, like system calls, are critical to application function when used in place of system calls.

81. As system calls exist in operating system space and execute in kernel mode, there are hazards that may arise from modifying system calls, such as for development purposes, given their highly privileged mode of execution. Ex-1005, 3:19-27. To avoid these issues, Deianov discloses that its system call wrappers operate outside of the kernel in user space and are loaded by application programs in user space. Ex-1005, 13:7-11. As I explain above in the Technology Background, because Deianov's system call wrappers are loaded by programs in user space, they form part of the application program during its execution. *See* Section VI.B.

82. As I also explain above in the Technology Background, object code, such as that constituting Deianov's system call wrappers, can be stored in shared libraries. *See* Section VI.D. This allows for the system call wrappers to efficiently be accessed and utilized by a plurality of applications. While Deianov does not describe where it stores its system call wrappers, the most logical and obvious choice

for Deianov would be storing them in a shared library, which was a well-known and widely implemented technology at the time.

83. Deianov also explains that its system call wrappers are accessible to any application program running in Deianov's system through select processes associated with each application program. Ex-1005, 4:37-40, 6:59-66. As I explain above in the Technology Background, because the system call wrapper is loaded into the process address space associated with an application program when that application program is being executed, the instance of that system call wrapper is unique to that particular application program. *See* Section VI.C; Ex-1005, 4:37-40, 6:59-66.

B. Ely

84. Ely is titled "Alpine: A User-level Infrastructure for Network Protocol Development," and relates to network protocol development. Like both Deianov and the '058 Patent, Ely aims to provide a service traditionally provided in operating system space in user space. Ex-1006, 4.

85. Ely emphasizes the importance of network protocols, particularly IP, TCP, and UDP, that govern communications between applications and computing devices on networks. Ex-1006, 1. The '058 Patent explicitly identifies the TCP and IP network protocols as examples of services critical to application functionality and further identifies them as CSEs. Ex-1001, 6:11-13.

86. Ely explains that the networking stack where network protocols reside is traditionally part of the operating system kernel. Ex-1006, 1. Residing there allows them to have the higher level of privileges afforded to kernel-resident services.

87. Ely recognizes, however, that network protocol development is challenging to do in the kernel because it requires system reboots during testing, complicates debugging, and risks crashing the entire system if an error occurs. Ex-1006, 1.

88. Ely's proposed solution to this development challenge, Alpine, moves the networking stack into user space for more seamless use without requiring any modifications to the operating system, the applications using the stack, or the networking stack itself. Ex-1006, 1.

89. Alpine has a user-level dynamic library running on a computer system in which the networking stack is placed. Ex-1006, 1, 7. Ely explains that Alpine's library exploits dynamic linking to manipulate the order in which applications are linked such that Alpine's dynamic library, and therefore, its networking stack, are used instead of the kernel's. Ex-1006, 7.

90. Ely further explains that Alpine's networking stack can be used simultaneously by the applications executing in the system. Ex-1006, 6-8, Fig. 3. This is an important feature of a computing system, as multiple applications may

need to use the networking stack at the same time, so allowing these simultaneously executing applications to use the networking stack at the same time ensures the applications perform as they are expected to by a user. As I explain in the Technology Background above, because these applications are dynamically linked to Alpine's library, each application runs its own instance of Alpine's networking stack when using Alpine. *See* Section VI.D.

91. Ely also notes the importance of Alpine's performance "in order to be a useful tool" for network protocol development. Ex-1006, 8. The purpose of the Alpine system described in Ely is allowing developers to have access to a networking stack, like that found in the kernel, in user-mode so that a developer can easily test changes on the networking stack without risking crashes or other issues inherent to modifying kernel-level networking protocols. Therefore, having Alpine perform in a manner as close to the kernel-level networking stack as possible would benefit any users of Alpine such that applications utilizing Alpine do not compete for resources and that changes to Alpine's user-level networking stack protocols are transferable to the operating system kernel.

C. Levine

92. Levine is a well-known textbook from the late 1990s that discusses the processes of linking and loading in computing systems and how these methods are used to turn code into executable programs. Ex-1007, 10.

93. Levine notes that different programs running on a computer often share a significant amount of common code. Ex-1007, 18. This common code may be stored in a shared library such that it may be utilized by the different applications in the system which require it rather than repeating the code for each application which requires it. Ex-1007, 80. In a shared library, the code is in an archive format that is accessible to and understood by different applications.

94. Levine highlights the well-known fact that shared libraries offer several benefits, including improved runtime performance and memory use. Ex-1007, 18. For example, by having code commonly utilized by multiple applications stored in one place accessible to all applications simultaneously instead of copying that code into each individual application, memory usage is greatly reduced.

95. Levine also explains that in systems that handle multiple programs simultaneously, each program has its own address space, enabling instances of library routines to be used concurrently by multiple programs within their respective address spaces. Ex-1007, 52. When a program, also known as an application in user space, is linked to a particular library routine, the shared library's code is mapped into the address space of that program such that an instance of that shared library routine forms a part of the application.

96. Levine points out that dynamically linked shared libraries have been used since the 1990s to delay binding of code in a shared library with a program

until the program starts running. Ex-1007, 10, 18. Levine then espouses the common knowledge of the time that dynamically linked shared libraries provide “a powerful and high-performance way to extend the functions of programs” because they are easier to create and update than statically linked shared libraries and permit programs to load and unload library routines at runtime, which is otherwise difficult to achieve. Ex-1007, 18, 80-81.

97. Levine gives the examples of Windows dynamic link libraries (DLLs) and UNIX ELF shared libraries as two such dynamically linked shared libraries. Ex-1007, 13, 18, 98-99. In fact, these are two of the best-known types of shared libraries, and I have used them extensively during my career, including prior to 2003. Almost any programming I have done in Windows or UNIX utilizes these two shared libraries, respectively.

98. Levine explains that ELF libraries use position independent code, or PIC, so that multiple processes may utilize the same code stored in a shared library simultaneously. Ex-1007, 100. PIC allows multiple programs to load the same shared library code, at the same time or at different times, at a location in each program’s address space without conflicting with other programs.

99. Levine also discloses that ELF libraries contain “dynamically linked files,” and when these files are linked to a program, they result in a “dynamically

bound program” or application. Ex-1007, 101, 106. This means that the shared library code is loaded only when the program is running.

D. Sands

100. Sands is a U.S. Patent that issued on March 1, 1994, and it is titled “Shared libraries implemented with linking program loader.” As with Levine, Sands is directed towards a computer system for linking to shared libraries.

101. Sands states that the goals of its invention include improving programs’ utilization of shared libraries without affecting run-time performance by enabling rapid run-time linking of programs to shared libraries and avoiding modifications to the operating system, compiler, or assembler in doing so. Ex-1008, 2:24-37.

102. Sands discloses that its disclosed system can run multiple programs simultaneously and that each of these programs has references to shared libraries that they utilize. Ex-1008, 4:12-17. Sands further discloses that this shared library contains items, such as functions and data. Ex-1008, 2:59-60.

103. Sands discloses a specific process through which a program utilizes shared library components, but its broad teachings of using shared libraries to provide shared resources to running multiple programs simultaneously is applicable to different systems, including Deianov. When a user invokes a program, memory space for that program is allocated, and then a “relocatable file” which represents the invoked program is loaded from storage into memory. Ex-1008, Fig. 3A, Fig.

3B, 5:61-66. Sands discloses that the addresses of the library items referenced by the program are then inserted directly into the relocatable file. Ex-1008, Abstract. This results in a program with the necessary library items loaded into it ready for operation. As the '058 Patent explains, through this linking process the shared library component forms a part of the software application linked to it. Ex-1001, 2:10-15.

IX. CLAIM CONSTRUCTION

104. I understand that Petitioner and Patent Owner have exchanged claim constructions for certain terms of the '058 Patent but have not yet submitted claim construction briefings for any terms of the '058 Patent. I am unaware of any claim construction determinations related to the '058 Patent.

105. I further understand that Patent Owner has agreed to certain constructions in its litigation against Petitioner. I also understand that Patent Owner has proposed for certain claim terms a construction other than plain and ordinary meaning. I also consider obviousness under these constructions in my analysis below.

106. For the purposes of this proceeding, I have considered whether the Challenged Claims are invalid under their plain and ordinary meaning, the constructions agreed upon or argued in the Related IBM Litigation (*VirtaMove, Corp. v. Hewlett Packard Enterprise Co.*, 2:24-cv-00093-JRG (E.D. Tex.) (lead

case) and *VirtaMove, Corp. v. International Business Machines Corp.*, 2:24-cv-00064-JRG (E.D. Tex.) (member case)), and the lexicography provided in the specification of the '058 Patent for relevant terms. In my opinion, the Challenged Claims are obvious under any of these constructions. The agreed upon terms and the constructions argued by Patent Owner for disputed constructions are shown below.

Term	Claim(s)	Agreed Upon Construction
Preambles	1	The preambles are limiting.
processor	1	Physical computer processor.
user mode	1	The context in which applications execute.
kernel mode	1	The context in which the kernel portion of an operating system executes. Application code cannot run in kernel mode.
instance / unique instance	1	A single, specific occurrence or copy.
a plurality of software applications / the plurality of software applications a processor / said processor an operating system / the operating system OS critical system elements (OSCSEs) / OSCSEs	1-4, 18	These terms and their corresponding antecedents refer to the same entity, respectively.

shared library critical system elements (SLCSEs) / the SLCSEs one or more of the plurality of software application / the one or more of the plurality of software applications a first of the plurality of software applications / the first of the plurality of software applications / at least first of the plurality of software applications a second of the plurality of software applications / the second of the plurality of software applications a SLCSE related to a predetermined function / a first instance of the SLCSE a SLCSE for performing a same function / a second instance of the SLCSE		
---	--	--

Ex-1023, Appendix A.

Term	Claim(s)	Patent Owner's Construction
critical system element	1	Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application

functional replica	1	Substantial functional equivalents or replacements of kernel functions
shared library	1	An application library whose code space is shared among all user mode applications

Ex-1023, Appendix C.

X. INVALIDITY OPINIONS: '058 PATENT

107. In my opinion, and for the reasons explained below, the Challenged Claims of the '058 Patent are invalid based on the following references and combinations:

A. Ground 1: Claims 1-4 and 18 Are Rendered Obvious By Deianov, Or Alternatively Obvious In View Of Deianov In Combination With Levine And/Or Sands

108. In my opinion, Deianov alone, or in combination with Levine and/or Sands, renders claims 1-4 and 18 obvious.

1. Motivation to Combine

109. In my opinion, as discussed below, the Challenged Claims are obvious in view of Deianov by itself. But they are also obvious in view of either or both of Levine and Sands.

110. Deianov describes the implementation of user-mode system call wrappers for select processes which utilize those system call wrappers rather than kernel-resident system calls. *See* Section VIII.A. Deianov is a system in which

multiple processes simultaneously execute, Ex-1005, 5:60-61, and the disclosed system call wrappers may be simultaneously accessed by applications. Ex-1005, 9:31-37. Deianov further describes the advantages of using user-mode system call wrappers, including extending and customizing operating system functionality, increasing system efficiency and performance, and mitigating the risks associated with using untested or modified system call wrappers in kernel space. Ex-1005: 2:3-3:6. The specific way in which these system call wrappers are stored and accessed by applications, however, is not discussed by Deianov. But, implementations of dynamic libraries, along with the specific features of those dynamic libraries, were well-known and the most widely used type of shared library in the art since at least the 1990s, and it would have been obvious to a POSITA to use dynamic shared libraries with Deianov's system based on Deianov's teachings. *See* Section VI.D; *see, also, e.g.*, Ex-1007; Ex-1008. Use of a dynamic shared library would allow for applications to dynamically link to shared libraries and utilize unique, private instances of the components stored in those shared libraries after mapping those components into their address space.

111. It also would have been obvious in view of Levine and/or Sands. In my opinion, a POSITA would have been motivated to combine Levine's and/or Sands's teachings regarding dynamic shared libraries with Deianov's system in order to store Deianov's user-mode system call wrappers in shared libraries for use by a plurality

of applications with multiple, private instances of shared library components (SLCSEs) provided to those applications for running simultaneously with Deianov's system.

112. Levine teaches ELF dynamic shared libraries. *See* Section VIII.C. These ELF libraries are dynamic shared libraries implemented in the UNIX operating system. Ex-1007, 100. Deianov discusses the UNIX operating system as an example of a multitasking operating system (Deianov is implemented on a multitasking operating system), and the terminology it uses comes from UNIX. Ex-1005, 1:15-15, 6:5-15, 7:50-58. Levine teaches that in multi-tasking operating systems (i.e., systems that can run multiple applications at once), each application has its own address space which allows for shared library components (SLCSEs) to be privately instanced for each application in its respective application space. Ex-1005, 51. Deianov is such a multi-tasking operating system. Ex-1005, 2:42-45. Further, Levine discloses that ELF libraries utilize position-independent code (PIC). Ex-1007, 100. PIC allows for the same shared library component code to be instanced and simultaneously accessed by multiple applications. This allows for simultaneously running applications to have their own instances of a shared library component and simultaneously utilize their respective instances, and through this increasing resource use efficiency and, with it, system speed.

113. Sands provides another example of a shared library implementation. As I describe in the Background of the Prior Art, Sands teaches a method for high performance linking of applications to shared libraries. *See* Section VIII.D. Specifically, Sands teaches a method through which a multi-tasking operating system, such as Deianov's, can utilize Sands's rapid linking method to insert the addresses of shared library components directly into an application's address space at run time. Ex-1008, 2:43-46. Moreover, in Sands's shared library system, while a shared library component is being utilized by an application, it remains available for use by other applications for simultaneous use such that each application has its own unique, private instance of that shared library component. Ex-1008, Claim 1 ("[E]xecutable library routines remain unmodified during execution of the invoked computer program and are available for use by other computer programs in said computer system."). Simultaneous use of shared library components as described in Sands increases efficiency by preventing the need for repeatedly copying shared library component code for separate applications.

114. Therefore, in my opinion, and for the reasons I describe below, a POSITA would have been motivated to modify Deianov's system to incorporate Levine's and/or Sands's shared library teachings to achieve these benefits.

115. Specifically, a POSITA would have been motivated to store Deianov's system call wrappers in a dynamic shared library as taught by Levine and/or Sands

for simultaneous access by the plurality of applications. In my opinion, a POSITA would have been motivated by the teachings of Levine and Sands in this regard, as well as Deianov's teaching that its system call wrappers are intended for use by a plurality of applications, including simultaneous access of the same system call wrapper, *see, e.g.*, Ex-1005, 5:59-62, 9:31-38, to improve such access to system call wrappers for the applications in Deianov's system.

116. A POSITA would have understood that a dynamic shared library is an obvious and effective method through which to provide such improved access. Indeed, the purpose of a shared library is to provide access to shared components stored inside the library to a plurality of applications which utilize them. Storing system call wrappers in dynamic shared libraries for providing instances of shared library components to applications was a known method in the art. In fact, Levine even gives an example of a "system call library" as a popular shared library, indicating that system calls can be stored in a shared library. Ex-1007, 80. A POSITA would have further understood that Levine and Sands both teach dynamic shared libraries that would have been desired by a POSITA, as both references teach libraries which facilitate access to shared resources inside the libraries and which support simultaneous utilization of unique, private instances of those shared components by multiple simultaneously executing applications. *See, e.g.*, Ex-1007, 80-81; Ex-1008, 4:12-14, Claim 1. The use of a dynamic linking at runtime would

be an obvious choice to efficiently utilize common resources such as system call wrappers among the various applications in Deianov's system which utilize those system call wrappers. In my opinion, a POSITA would have understood that storing Deianov's system calls in dynamic shared libraries as taught by Levine and/or Sands would advantageously facilitate applications running in Deianov's system simultaneously accessing and utilizing private, unique instances of the system call wrappers stored inside the library.

117. In my opinion, these modifications would simply amount to combining prior art elements according to known methods to yield predictable results. Levine and Sands both teach a known method for storing files, such as system call wrappers, in a dynamic shared library such that they may be simultaneously utilized by a plurality of applications. *See, e.g.*, Ex-1007, 80-81; Ex-1008, 4:12-14, Claim 1. Further, Deianov discloses system call wrappers, a known prior art element. *See* Ex-1005. A POSITA would have understood that combining Levine's and Sands's teachings with Deianov merely takes its system and combines it with the known shared library methods taught by Levine and Sands.

118. Further in my opinion, a POSITA would find these results predictable. A POSITA would have understood that each of the elements of the combined references merely performs the same function as it does separately. Deianov's system call wrappers still provide a user-mode alternative to system calls, while

Levine's and Sands's shared libraries still store files inside themselves for simultaneous access by a plurality of applications. Indeed, storing system call wrappers in shared libraries was a known combination in the art. *See, e.g.*, Ex-1007, 80. Therefore, a POSITA would find the result of storing these system call wrappers in a shared library predictable, a known technique, to have a reasonable expectation of success.

119. Also, for these same reasons, in my opinion, this combination is merely applying a known technique to improve similar devices in the same way. Indeed, all three of Deianov, Levine, and Sands are in the similar field of computing systems. Each describes the use of computing resources by applications and systems for improving or facilitating access to those resources. Further, in combination, Levine's and Sands's teachings are being used to improve Deianov's system in the same way that Levine and Sands describe the benefits of their systems. Therefore, for the same reasons I discuss above, combining Deianov's system with the teachings of Levine and/or Sands is merely the use of a known technique to improve a similar device in the same way.

120. Further in my opinion, combining Deianov's system with the teachings of Levine and/or Sands would be obvious to try. As I describe above, storing system calls in some form in shared libraries was a known technique in the art. *See, e.g.*, Ex-1007, 80; Ex-1015, 10. Shared libraries were among a finite number of

identified, predictable solutions to Deianov's goal of providing system call wrappers simultaneously to a plurality of applications. Further, because Deianov does not put constraints on the use or storage of its system call wrappers besides them remaining in user-mode, a POSITA would have understood that shared libraries such as those taught by Levine and Sands were obvious techniques to incorporate in order to facilitate simultaneous access to system call wrappers by the plurality of applications in Deianov's computing system. As I explain above, the use of Levine's and Sands's teachings to improve Deianov's system would yield predictable results. A POSITA would have had a reasonable expectation of success because placing system call wrappers in shared libraries was a known technique for storage of system call wrappers, and because Deianov does not place constraints on the storage or access of its system call wrappers beyond keeping them in user-mode. Both Levine and Sands teach shared libraries in user-mode. *See, e.g.*, Ex-1007; Ex-1008. To implement a shared library in Deianov, a programmer would compile code, such as system call wrappers, with a shared library flag, then use a linker command to combine the selected code into a shared library. Implementing a shared library into Deianov in this way would only require basic computer programming skills that a POSITA at the time would have been familiar and proficient in.

121. Therefore, in my opinion, a POSITA would have been motivated to modify Deianov to incorporate Levine's and/or Sands's teachings to achieve these benefits.

2. **Claim 1**

122. In my opinion, claim 1 is obvious in view of Deianov alone or in combination with Levine and/or Sands.

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

123. Deianov discloses or renders obvious a computing system. As I describe above in the Background of the Prior Art, Deianov's invention is directed towards system calls and the selective interception of those system calls. Section VIII.A; Ex-1005, 1:8-11. The background section of Deianov discusses "multitasking operating systems" which are used in computing systems. *See* Ex-1005, 1:13-15. The background section of Deianov continues by describing how system calls are implemented on such computing systems. *Id.*, 1:21-30. A POSITA would understand a computing system to be a computer or other device(s) upon which computing is performed. Such a device constitutes, at its base, computer hardware and an operating system which allow for computing to be performed. The components of Deianov described further support that Deianov is implemented on a computing system, including the disclosure of computer memory and an operating system. *See* Ex-1005, Fig. 1, 3:31-34.

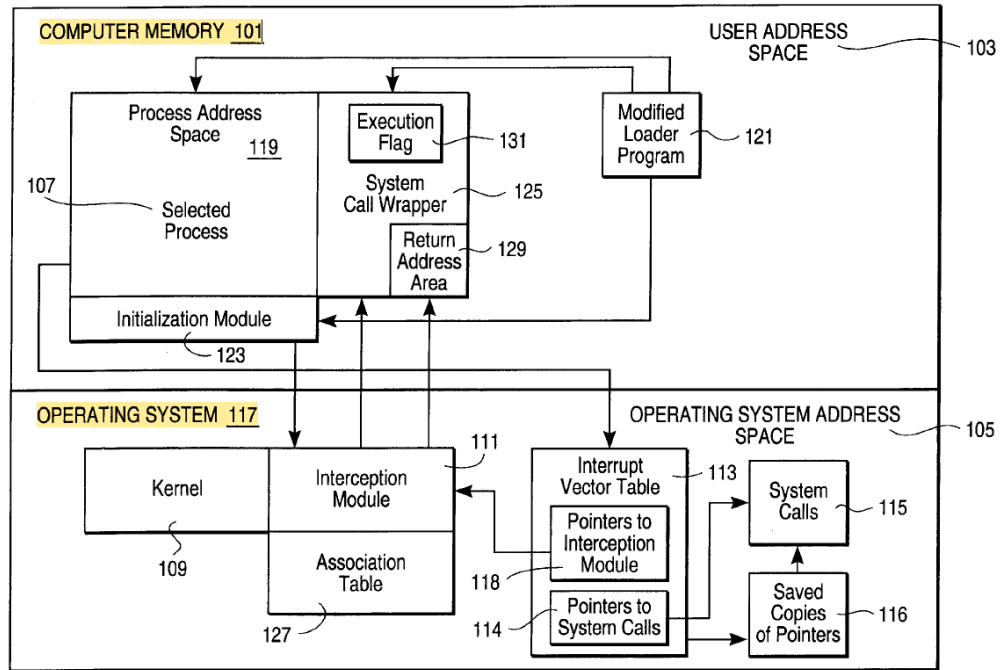


FIG. 1

Ex-1005, Fig. 1 (annotated).

124. Further in my opinion, Deianov discloses or renders obvious that the computing system is for executing a plurality of software applications.

125. Deianov is implemented upon a multitasking operating system. Ex-1005, 1:8-9. A multitasking operating system is a system in which multiple, *i.e.* a plurality, of applications execute at the same time, and a POSITA would have known that. Indeed, Deianov discloses that in multitasking operating systems plural “application programs execute.” Ex-1005, 2:61-62. The term “application” refers to a program that is in user mode. The “tasks” referred to by the term “multitasking” are different processes. Each of these tasks which are running may be associated with a different application, such that multiple tasks running concurrently means that

multiple applications may run concurrently. In this context, “software application” and “application” refer to the same concept, and the word “software” is simply added for clarity. Therefore, the multitasking operating system on which Deianov is implemented is a computing system for executing a plurality of software applications.

b. Claim 1[a]: “a processor;”

126. Deianov discloses or renders obvious a processor. Deianov explains that in making a system call involves a computer’s central processing unit, or CPU. Ex-1005, 1:46-47. As I describe in the Technology Background above, a central processing unit, or “CPU,” is a form of physical computer processor. *See* Section VI.B. Deianov explains that a CPU is utilized to make system calls, and Deianov is a system that executes system calls. Ex-1005, 1:45-48, Abstract.

127. Deianov also specifies that UNIX and Windows are multitasking operating systems on which the disclosed invention may be implemented. Ex-1005, 1:13-14. As a general matter, operating systems are concerned with, among other things, managing hardware such as processors to allow processes to accomplish work. *See* Section VI.A, C; Ex-1012, 4. Because multitasking operating systems need processors to efficiently and concurrently execute tasks, a POSITA would have understood that computing systems running UNIX or Windows include a processor.

See, e.g., Ex-1033. This processor would be a physical computer processor, such as a computer chip.

128. Additionally, Deianov discloses that processes and threads execute in the disclosed Deianov system. Ex-1005, 8:12-14, 9:31-32. The execution of threads and processes requires the computing system to include a processor, as I explain in the Technology Background. *See* Section VI.C.

c. Claim 1[b] “an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,”

129. Deianov teaches an operating system having an operating system kernel. Figure 1 of Deianov actually depicts an operating system with a kernel.

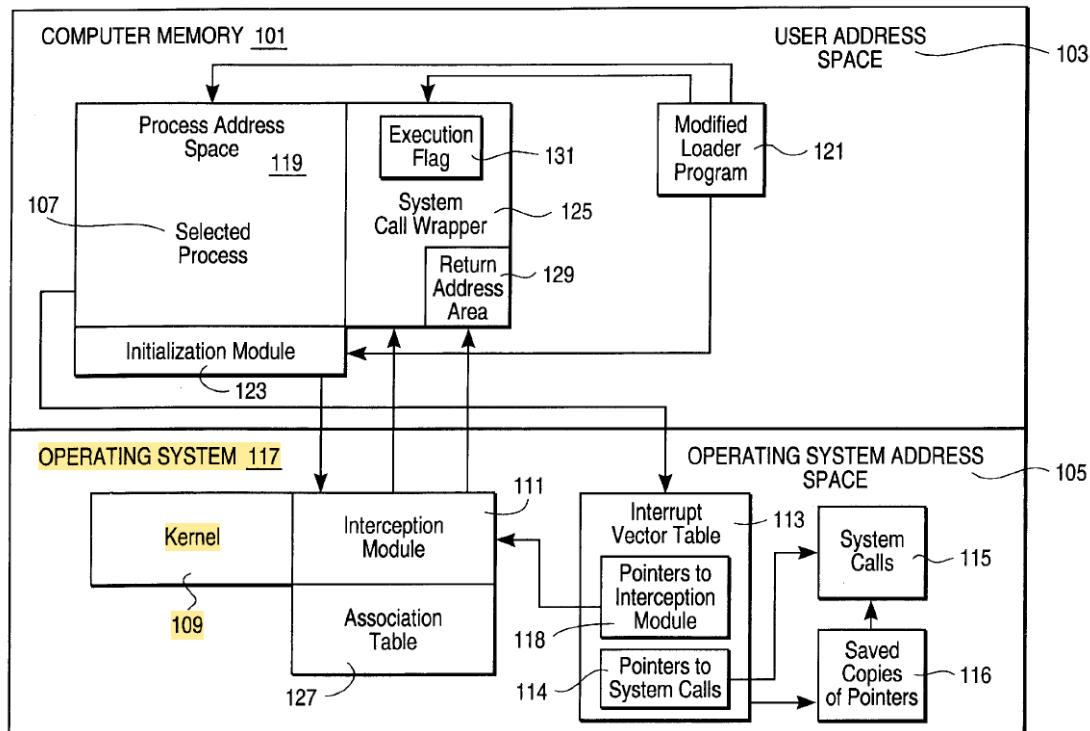


FIG. 1

Ex-1005, Fig. 1 (annotated).

130. As I explain in the Technology Background section, the operating system kernel is a key component of the operating system which manages tasks such as control of hardware and networking. *See* Section VI.A.

131. Deianov also discloses or renders obvious an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor. The specification of the '058 Patent explains that a critical system element is “[a]ny service or any part of a service, ‘normally’ supplied by an operating system, that is critical to the operation of a software application.” Ex-1001, 6:6-8. Further, I understand that Patent Owner has argued this same construction should be applied in the Related IBM Litigation. Ex-1023, Appendix C. The '058 Patent further describes “kernel mode” as “[t]he context in which the kernel portion of an operating system executes.” Ex-1001, 6:32-33. Further, I understand that Patent Owner has agreed in the Related IBM Litigation that “kernel mode” means “[t]he context in which the kernel portion of an operating system executes. Application code cannot run in kernel mode.” Ex-1023, Appendix A.

132. Deianov discloses or renders obvious such OSCSEs through its disclosure of “system calls.” Deianov explains that system calls are “subroutine[s], the object code of which is located in an operating system” which “perform[] some system operation, such as the system hardware or software resource.” Ex-1005,

1:31-37. Although making a “system call” can refer to the act of calling the system call object code in the kernel, Deianov uses the term “system call” to refer to the object code in the kernel itself. Ex-1005, 1:31-37. System calls are critical to the operation of software applications utilizing them because they provide a mechanism for applications through the operating system kernel to access its underlying hardware resources, which user-mode applications need to function as intended. *See* Section VI.B; *see also, e.g.*, Ex-1011, 47.

133. Deianov explains that system calls execute a number of functions, including opening and closing files, managing networking communications, and allocating memory. Ex-1005, 1:37-40. The '058 Patent also explains that OSCSE services include “memory allocation,” an operation that Deianov also attributes to its system calls. Ex-1001, 9:34-35. Deianov’s system calls are also located in and supplied by Deianov’s operating system. Ex-1005, 1:31-32, Fig. 1.

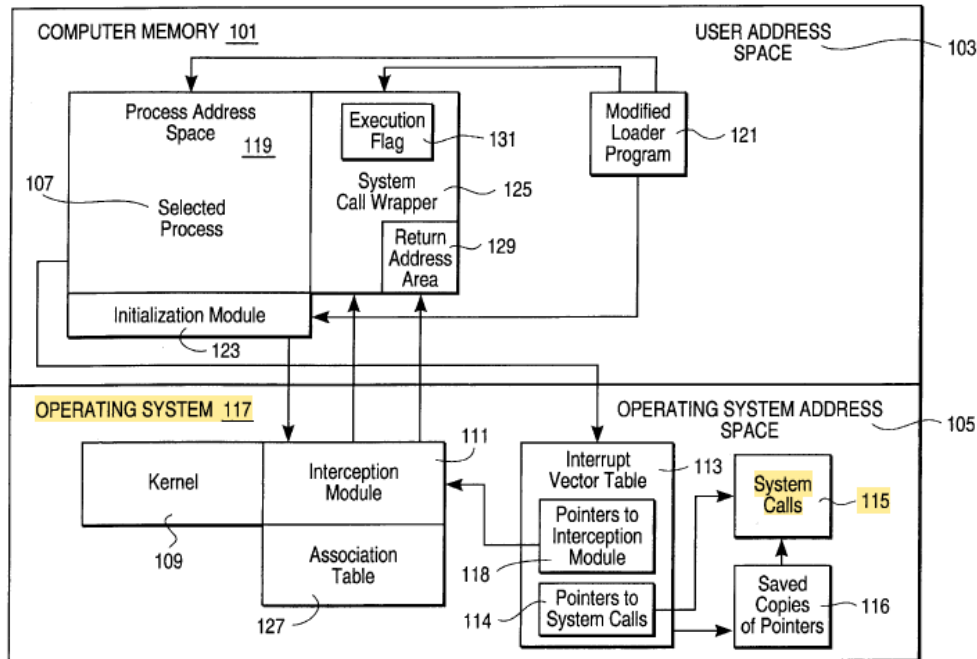


FIG. 1

Ex-1005, Fig. 1 (annotated).

134. These system calls are located in operating system address space, such that they are located in the operating system. Further, they are services that are provided by the base operating system itself rather than services imported by the user. Therefore, Deianov's system calls are objects that provide services normally provided by the operating system which are critical to the operation of a software application, and constitute CSEs as identified by the '058 Patent. Moreover, as they are located in the operating system, they further constitute OSCSEs.

135. Deianov's system calls also run in kernel mode, which is the context in which the kernel runs. Deianov discloses that "an operating system kernel 109 executes in operating system address space 105," Ex-1005, 11:55-56, and that

system calls 115 are also in OS address space. Ex-1005, Fig. 1. As I explain in the Technology Background above, system calls are part of the kernel in that they are controlled by and provide access to privileged functionality exclusive to the kernel. *See* Section VI.B. Since system calls are kernel-exclusive, they execute in kernel mode, meaning that they run in the context in which the kernel executes. *See* Section VI.B. Further, a POSITA would understand that application code cannot run in kernel mode because its code is designed to only run in less privileged user mode and thus is not compiled with the kernel option that would allow it to run in kernel mode. Further to this point, system calls are executed by the CPU, which executes the instructions needed to perform the system operations that the system calls bring about. *See* Section VI.B. Indeed, as I explain in the Technology Background, kernel mode generally refers to one of the modes of operation of a computer's CPU. *See* Section VI.B. A POSITA would have therefore understood and found obvious that the system calls (OSCSEs) execute using Deianov's CPU (i.e., "said processor"). Ex-1005, 1:45-54.

d. Claim 1[c]: "a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

136. Deianov discloses the use of system call wrappers, which Deianov describes as alternative object code that applications running on Deianov's system can execute instead of system calls. Ex-1005, 1:64-2:2, 3:30-34, 4:5-12, 7:3-4. As

Deianov itself indicates, because applications can use system call wrappers in the place of system calls to produce the same results as system calls, system call wrappers replicate the functionality of the system calls. *See* Section VI.B. By performing the same critical functions as system calls (which are CSEs), a POSITA would have understood and found obvious that system call wrappers are also CSEs.

137. Deianov also discloses that at the runtime of an application, a system call wrapper is loaded into user-mode process space. Ex-1005, 4:34-37. Deianov uses the term “load” to indicate that the system call wrapper is loaded from memory. A POSITA would understand and find obvious that the system call wrapper is stored in a shared memory space accessible to all applications, specifically within a shared library. This is because system calls are used by all applications, as they serve as an essential interface between the user mode applications and the functions provided by the operating system kernel. *See* Section VI.B. System calls are not limited to use by only a subset of applications. *See* Section VI.B. Shared libraries would therefore be an obvious choice to store the system call wrappers, because they are designed as a general matter to be accessible to any application that requires their functionality. *See* Section VI.D.

138. It would also be obvious to store Deianov’s system call wrappers in a shared library—a well-known and widely used existing technology—to allow the plurality of software applications to dynamically link and utilize the same system

call wrappers, in view of Levine and/or Sands, for all the same reasons I discuss in Section X.A.1. above. As I emphasize in the Motivation to Combine section above, incorporation of Levine's and/or Sands's teachings of dynamic libraries, which are shared libraries as claimed in the '058 Patent, would merely be a combination of prior art elements with a predictable result, would further use a known technique to improve a known device ready for improvement to yield predictable results, and would be an obvious combination to try. *See* Section X.A.1.

139. The '058 Patent defines a shared library as “[a]n application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.” Ex-1001, 6:49-53. An application library is defined as “[a] collection of functions in an archive format that is combined with an application to export system elements.” Ex-1001, 6:46-48. I understand that IBM proposes the lexicographical definition of shared library, and the term application library therein, identified by the '058 Patent at 6:46-53 as its construction in the Related IBM Litigation. I further understand that in the Related IBM Litigation, Patent Owner has argued that a shared library is “an application library whose code space is shared among all user mode applications.” *Id.*, Appendix C.

140. Levine and Sands disclose that in their shared libraries, in which SLCSEs are stored, components stored therein (such as SLCSEs) are accessible to either one or a plurality of applications. For instance, Levine discloses that “ELF shared libraries . . . can be shared among multiple processes.” Ex-1007, 100. The components of these libraries, *i.e.*, a collection of functions, are thus shared among multiple processes, and each of these processes may be associated with a different application. *See* Section VI.D. Sands similarly discloses that the shared “library 24 is shared by the programs 16, 18, 20, 22 and any other programs which may be run on computer 10.” Ex-1008, 4:65-67. Shared libraries store these components in an archive format, which is a library format understandable to multiple applications, and which allows for the efficient exporting of components stored inside to applications which access the shared library. These shared libraries do not describe any limits placed on which applications they may be shared with, and indeed a POSITA would understand that they can be shared with all user mode applications, as that is how shared libraries are intended for implementation. This shared library code space is thus shared among all user mode applications, and by being in user mode, is in a code space different than that occupied by the kernel and its associated files. This is illustrated in Figure 1 of the '058 Patent, which shows SLCSEs and the code space they reside in as being separate from the kernel code space. *See* Ex-

1001, Fig. 1. Therefore, the shared libraries disclosed by Levine and Sands are “shared libraries” as claimed by the ’058 Patent.

141. Further, the disclosed SLCSEs are used by the plurality of applications in user mode. Deianov discloses that its system call wrappers (SLCSEs) are located in “user address space,” and that applications execute in that space. Ex-1005, Fig. 1. As I explain earlier in the Technology Background, applications as a general matter execute in user space. *See* Section VI.B. Accordingly, Deianov’s applications execute in user mode, which is the “context in which applications execute.” Ex-1001, 6:31. I understand that Patent Owner also agreed to this definition of user mode in the Related IBM Litigation. *See* Ex-1023, Appendix A. Additionally, Deianov teaches that system call wrappers are accessible to multiple applications. Ex-1005, 1:64-2:2, 5:55-62, 9:33-38.

142. As I also explain above in the Technology Background, applications are a form of program, and thus each of these disclosed applications may access the shared library and consequently its components, that is, SLCSEs. *See* Section VI.C. As all applications execute in user-mode, the SLCSEs taught by Deianov in view of Levine and/or Sands are accessible to a plurality of software applications in user-mode.

- e. **Claim 1[c][i] “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,”**

143. The '058 Patent describes the term “replica” as used to “denote a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system.” Ex-1001, 1:66-2:1. During prosecution, Patent Owner explained that SLCSEs are “functional replicas” and are, “in other words, replacements” for OSCSEs. Ex-1002, 288. Further, I understand that Patent Owner has argued in the Related IBM Litigation that “functional replicas” means “substantial functional equivalents or replacements of kernel functions.” Ex-1023, Appendix C.

144. As I mention above, Deianov’s system call wrappers are code that serve as an “alternative” to their corresponding system calls. Ex-1005, 1:64-2:2. Deianov explains that these system call wrappers are alternatives because for processes selected for interception by the interception module, the system call wrappers are executed in place of system calls. Ex-1005, Abstract. By doing so, the system calls provide an alternative to the system calls residing in the kernel, providing their functionality in an alternative manner. Deianov does not put any limits on which processes can be selected for interception, and given the storage of system call

wrappers in a commonly accessible location, a POSITA would have understood that Deianov's system calls are available to processes and the applications those processes are associated with. Further, system calls and system call wrappers perform the same tasks but differ only in their method of execution. *See* Section VI.B. While system calls directly communicate with the kernel, system call wrappers provide a user-mode alternative, making them functional replicas of each other. By executing the system call wrapper in place of the system call, the system call wrapper replaces it. Moreover, a POSITA would have understood and found obvious that these SLCSEs stored within a shared library are accessible to all (which includes some) applications, including the plurality of software applications, for the same reasons I discuss in Section X.A.2.d. Therefore, Deianov's system call wrappers (SLCSEs) are functional replicas of system calls (OSCSEs).

145. Further in my opinion, Deianov alone or in view of Levine and/or Sands renders obvious that "when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software application." Patent Owner argued during prosecution of the '058 Patent that "forms a part" means that SLCSEs "reside in the same address space as application code." Ex-1002, 329. The '058 Patent's specification also explains this. Ex-1001, 2:10-15. Deianov discloses that when a system call wrapper (SLCSE) is called by a process, the system call wrapper is "loaded into their process address

space.” Ex-1005, 4:32-45. As I explain above in the Technology Background, a process associated with an application shares an address space, so when executed, the system file wrapper resides in the same address space as application code. *See* Section VI.C. Thus, in my opinion, Deianov discloses that when an SLCSE is accessed by one or more of the plurality of software applications, it forms a part of the one or more of the plurality of software applications. In fact, Figure 1 shows the system call wrapper residing as a part of the process.

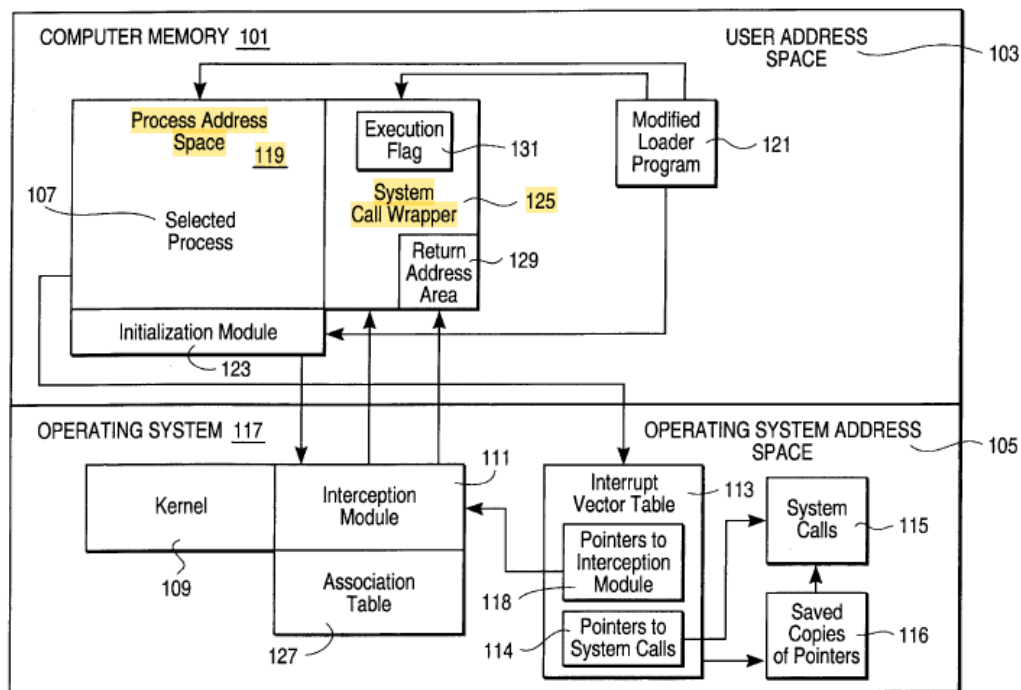


FIG. 1

Ex-1005, Fig. 1 (annotated).

146. Furthermore, the '058 Patent discloses that SLCSEs may “form a part of at least some of the software applications by being linked thereto.” Ex-1001, 2:14-15. I understand that in the related litigation brought by Patent Owner against

Google, Patent Owner agreed that “[an SLCSE] forms a part of the one or more of the plurality of software applications” means “it is linked to one or more of the plurality of software applications.” Ex-1024, 1. I further understand that in the Related IBM Litigation, IBM has proposed that forming a part means to “literally form a part of the application such that it resides in the same address space as application code in contrast to a proxy that is exclusive of the application.” Ex-1023, Appendix C. As I mention in the Technology Background, when an application links to a shared library, the code from the library is loaded into the same address space as the application, allowing the application to directly access and execute the library code as part of its own code rather than do so through an intermediary or proxy. Section VI.D. Through this, the shared library component (SLCSE) literally forms a part of the application such that it resides in the same address space as application code. A POSITA would have understood that this is different from a proxy that is exclusive of an application, as a proxy does not reside in application address space. Therefore, as applications link to components of a shared library, both constructions of “forms a part” are met.

147. Deianov’s SLCSEs are linked to and accessed by one or more of the plurality of software applications. As I mention in the Background of the Prior Art, both Levine and Sands disclose dynamically linked shared libraries. *See* Section VIII.C, D. Levine discloses that ELF libraries contain “dynamically linked files,”

and when these files are linked to a program, result in a “dynamically bound program” or application. *See* Section VIII.C; Ex-1007, 101, 106. I also mention in the Background of the Prior Art that Sands discloses a system in which a linker is utilized to link programs or applications to library components directly at run-time. *See* Section VIII.D; Ex-1008, Abstract; claim 1. Thus, in my opinion, one or more of the plurality of applications access and link to the components of the shared libraries in Levine and Sands.

148. As I discuss in the motivation to combine section, a POSITA would have been motivated to incorporate the teachings of Levine and/or Sands regarding dynamically linked shared libraries into Deianov’s system. *See* Section X.A.1. As I describe in both that section and the Technology Background, it was well known that shared libraries allow for a commonly utilized file stored inside to be accessed by a plurality of applications, rather than requiring that the code in that file be rewritten in each of the applications. *See* Section X.A.1; Section VI.D; Ex-1012, 7; Ex-1017, 1-2. As I detail in the Technology Background, this occurs through the process of linking applications to routines stored in the shared library, allowing applications to access necessary routines in the shared library in a more streamlined, efficient way. *See* Section VI.D.

- f. **Claim 1[c][ii]: “wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and”**

149. Deianov teaches that instances of system call wrappers (SLCSEs) provided to applications from the shared library are run in the context of those applications. Deianov discloses that “the system call wrapper 125 executes in the process address space 119 of the selected process 107. Therefore, the system call wrapper 125 is actually a part of the process 107.” Ex-1005, 8:34-37. As I describe in the Technology Background, processes are associated with a respective application. *See* Section VI.C. I explain earlier that it would be obvious to a POSITA that Deianov’s system call wrappers would be stored in a shared library. *See* Section X.A.2.d. Therefore, a POSITA would understand and find obvious that Deianov’s SLCSEs from the shared library run in the context of applications as the disclosed processes are associated with an application. *See* Section VI.C.

150. Deianov also discloses or renders obvious that SLCSE instances that run in the context of applications are not shared with other applications from the plurality of applications. As I describe in the Technology Background, each process address space is separate from other process address spaces, which is accomplished

by the operating system allocating each process its own virtual address space by the operating system. *See* Section VI.C. Moreover, the process address space is part of the associated application's address space. Section VI.C. Therefore, Deianov's SLCSE instances run in the context of an application are not shared with other applications.

151. Further, Deianov teaches that instances of an SLCSE may be provided to a first of the plurality of applications while at least a second application of the plurality of applications running under the operating system has unique instance of the SLCSE provided to it for performing the same function. Deianov discloses that "multiple processes 107 can execute simultaneously." Ex-1005, 5:60-61. As each process may be associated with a different application, the disclosure of simultaneously running processes indicates that multiple applications may execute simultaneously. *See* Section VI.C. By loading the SLCSE into the application's address space, each application may access and utilize a unique instance of an SLCSE with the same function. Indeed, different instances of the same SLCSE will have the same function as they are merely instances of the same code loaded in different locations. Moreover, Deianov teaches that all processes are run "under control of the operating system," which a POSITA would understand to mean that because processes are associated with applications, the plurality of applications are running under the operating system. Ex-1005, 8:12-14.

152. Further, in my opinion, a POSITA would have been motivated to incorporate the shared library approach of Levine and/or Sands into Deianov's system for the reasons I discuss in Sections X.A.1. and X.A.2.d. *See* Section X.A.1; Section X.A.2.d. A POSITA would have been motivated to utilize Levine's and Sands's concurrently running unique SLCSE instances so that Deianov's system call wrappers, which are SLCSEs when stored in a shared library, are accessed simultaneously by multiple applications without resulting in resource competition or unwanted data sharing. Ex-1005, 5:55-63. For the reasons explain in these sections, Deianov in view of Levine and/or Sands renders this limitation obvious.

- g. Claim 1[c][iii]: “wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.”**

153. Deianov discloses that system calls have predetermined functions, such as writing to a file or creating a new process, and that system call wrappers provide an alternative object code which may be utilized to implement this functionality in user-mode. Ex-1005, 1:34-39; 1:64-2:2. As I previously discuss, when stored in shared libraries as components, these system call wrappers constitute SLCSEs related to a predetermined function. *See* Section X.A.2.d. In my opinion, Deianov alone or in view of Levine and/or Sands also renders obvious that a SLCSE may be

provided to a first of the plurality of software applications for running that first instance of the SLCSE while a SLCSE for performing the same function may be provided to a second of the plurality of software applications for running the second instance of the SLCSE. *See* Section X.A.2.f. In fact, using dynamic libraries to store system call wrappers carries out the purpose of using dynamic libraries, as taught by both Levine and Sands. *See* Section VIII.C, D.

154. Deianov is implemented in a multi-tasking operating system, *i.e.*, a system which can run multiple applications simultaneously. Evidencing this, Deianov discloses that “multiple processes 107 can execute simultaneously.” Ex-1005, 5:60-61. Each process may be associated with a unique application. Therefore, Deianov teaching that multiple processes can execute simultaneously teaches that multiple different applications associated with those processes can run simultaneously. *See* Section VI.C. It would be obvious to a POSITA that when an instance of a system call wrapper is provided to an application while another instance of that same SLCSE is provided to another application, the multiple instances of an SLCSE stored in a shared library run simultaneously. *See* Section X.A.2.f.

155. In my opinion, this limitation would also be obvious in view of Levine and/or Sands for the reasons I discuss in Section X.A.1. Moreover, a POSITA would have been motivated to incorporate the teachings of Levine and/or Sands for the reasons I explain in Section X.A.1. Both Levine and Sands teach shared library

implementations in which applications linked to the same shared library may simultaneously run unique, private instances of the same component in that shared library.

3. **Claim 2: “A computing system as defined in claim 2, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.”**

156. As I explain in the Motivation to Combine section and in the previous sections, a POSITA would have been motivated to combine Deianov’s system with the shared library teachings of Levine and/or Sands such that multiple instances of the same SLCSE run simultaneously within the operating system. *See* Section X.A.1. As I explain above, Deianov, alone or in view of Levine and/or Sands, renders obvious multiple instances of an SLCSE stored in the shared library running simultaneously. *See* Section X.A.2.g. Further, these SLCSE instances run within the operating system in user mode. Programs running with the operating system execute in either kernel-mode, a highly privileged mode of operating, or user-mode, a less privileged mode of operation which applications execute in. Section VI.B. As Deianov discloses, its system call wrappers exist in and run in user-mode. Ex-1005, 8:12-14. Thus, Deianov’s SLCSEs run within the operating system. Indeed, as Deianov discloses, “processes execute in user address space 103 ***under control of the operating system.***” *See id.*

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

157. In Deianov’s system, system call wrappers co-exist with system calls, they do not replace them. Non-selected processes still make system calls to the system calls in the kernel (OSCSEs). Ex-1005, 4: 27-21. Thus, Deianov’s corresponding system calls (OSCSEs) remain in the operating system kernel. Indeed, Deianov at no point discusses removing the system calls that are in the operating system kernel. Instead, its system provides an alternative to kernel-resident system calls and leaves kernel system calls (OSCSEs) in the operating system kernel for those processes which are not selected for interception. These OSCSEs correspond to and are capable of performing the same function as SLCSEs because the SLCSEs are functional replicas of the OSCSEs. *See* Section X.A.2.e.

5. **Claim 4: “A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.”**

158. Deianov discloses that system call wrappers which constitute SLCSEs when stored in user-mode shared libraries, make system calls. *See* Ex-1005, 8:49-52 (“It is commonly desirable for the system call wrapper 125 to utilize OS resources. Like any process, the system call wrapper 125 utilizes such resources by making a system call 115.”). These SLCSEs constitute “the one or more SLCSEs

provided to one of the plurality of software applications having exclusive use thereof” because, as I explain in Section X.A.2., Deianov’s system call wrappers are provided to the plurality of software applications, where each application has exclusive use of a particular instance of a system call wrapper. These system calls allow the software applications to utilize OS resources, *i.e.* services, that are located in the operating system kernel. *See id.*

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

159. Deianov discloses or renders obvious that its system call wrappers are not copies of OSCSEs. Deianov’s system call wrappers (SLCSEs) are different from Deianov’s system calls (OSCSEs), and therefore are not copies of them. Indeed, system calls are designed for and implemented in kernel space while system call wrappers abstract this functionality in such a way as to implement system call functionality in user space. *See* Section VI.B. While they may have the same functionality, or slightly different functionality should a developer so choose, the code is different to account for the differences in how each is implemented. *See* Section VI.B. Code is designed to function in either kernel-mode or user-mode, and as the system call wrappers are implemented in user-mode, the code used would differ to account for this difference. Indeed, as Deianov discloses, system call wrappers are alternative object code. Ex-1005, 1:64-2:2. It would also have been

obvious to a POSITA that different code would be used for SLCSEs, since they do not operate in the same OS space.

B. Ground 2: Claims 1-4 and 18 Are Rendered Obvious By Deianov, Or Alternatively Obvious In View Of Deianov In Combination With Levine And/Or Sands

160. Ely alone, or in combination with Levine and/or Sands, renders claims 1-4 and 18 obvious.

1. Motivation to Combine

161. In my opinion, to the extent not already taught by Ely, a POSITA would have been motivated to combine Levine's and/or Sands's teachings of multiple, private instances of shared library components (SLCSEs) provided to applications for running simultaneously with Ely's computing system.

162. Ely discusses Alpine, a computer system with an infrastructure that implements a dynamic user-mode shared library which contains a networking stack therein. Ex-1006, 1, 7. While the specific implementation of this dynamic library is not discussed in Ely, dynamic libraries, along with the specific features of dynamic libraries, were well-known in the art since at least the 1990s. *See, e.g.*, Ex-1007; Ex-1008; Section VI.D.

163. Levine provides one such example. As I describe above, Levine teaches ELF dynamic shared libraries which utilize PIC to allow for the same shared library code to be privately instanced for simultaneous use by multiple

simultaneously running applications. *See* Section VIII.C, X.A.1. Ely is a multitasking operating system able to take advantage of these benefits. Ex-1006, 6. Further, these ELF libraries are dynamic shared libraries implemented in the UNIX operating system, which is the operating system that Ely's system is implemented on. Ex-1007, 100; Ex-1006, 1.

164. Sands provides another example of a shared library implementation. As I also describe above, Sands teaches a method for high performance shared libraries through an improved rapid linking system and in which multiple applications can simultaneously utilize the same shared library components. *See* Section VIII.D, X.A.1. Ely is a multitasking operating system able to take advantage of these benefits. Further, Sands does not disclose any limitations on which operating system it may be implemented. *See, e.g.*, Ex-1008. Instead, it discloses that its system requires no "modification of the operating system, the compiler, or the assembler." Ex-1008, 2:34-37.

165. In my opinion, and for the reasons I describe below, a POSITA would have been motivated to combine Levine's and/or Sands's teachings regarding private instances of SLCSEs provided to applications for running simultaneously with Ely's computing system so as to ensure that Ely's user-mode shared library networking protocols would be simultaneously available to Ely's applications without resource competition issues arising.

166. Ely explains that Alpine having reasonable performance to accurately replicate the kernel-resident networking stack is a critical component of Alpine's usefulness as a tool. Ex-1006, 8 ("Alpine must have a reasonable performance in order to be a useful tool."). In any computer system, a developer will generally look for ways to improve the performance of that system. These considerations include components including speed and efficiency. This is also true of Ely's Alpine system, which recognizes that speed is a key consideration in ensuring Alpine is usable to applications in the same manner the operating system kernel would be. Ex-1006, 8-11. Ely further describes the need for efficiently providing resources to simultaneously executing applications. Ex-1006, 6. Should Alpine not provide a high-performance alternative networking stack which replicates the functionality of the kernel networking stack, it would not be a useful tool for a developer. Ex-1006, 1. Indeed, a POSITA would have recognized that improving the performance to resemble the functionality of a kernel networking stack when accessed by multiple concurrently running applications would make Alpine a more useful tool. This is because with better performance and functionality, Alpine would be able to accommodate a greater number of application's needs and would also more quickly perform the tasks and functions that are required of it. Indeed, increasing system performance is a universal goal in the design of all computer systems.

167. Further, a POSITA would have recognized that in a multi-tasking computer system such as that disclosed by Ely, the computing system would benefit from ensuring lack of conflict between applications competing for resources. Ex-1006, 6. Networking stack protocols such as TCP, IP, and UDP are commonly used by the applications in an operating system. Should a computing system not account for these simultaneously executing applications needing these common resources, the applications' operations could fail or have their performance significantly degraded. As such, a computing system that provides support for simultaneously executing library components would also prevent simultaneously executing applications from competing for the same library component and the performance drawbacks associated with this competition such as application malfunction caused by lacking access to the critical shared library resources it calls.

168. In my opinion, a POSITA would have recognized that, to the extent Alpine does not already do so, it would be an obvious improvement to Alpine's shared library to simultaneously provide private instances of shared library components to applications. Indeed, a POSITA would have recognized that these modifications to the shared library (to the extent not already present) would greatly increase Alpine's performance. Recognizing this, a POSITA would have been motivated to add these teachings from Levine and Sands to Alpine. I discuss above

and in Ground I how each system discloses this improved performance through their shared library systems. *See* Section X.A.1.

169. In addition, Ely, Levine, and Sands occupy a similar field. All three references concern shared libraries, and their utilization in the context of applications linking to and utilizing shared resources stored inside them. A POSITA would recognize the similarity of the field occupied by these references, and this similarity would have motivated a POSITA to combine Levine's and Sands's teachings regarding improved shared libraries with Ely's system to realize Ely's goal of providing a high-performance user-mode networking stack stored in a shared library.

170. In my opinion, the modifications required to implement this functionality in Ely would simply amount to combining prior art elements according to known methods to yield predictable results. Levine and Sands both disclose a dynamically linked shared library where multiple instances of library components can be used at the same time to increase system performance. *See, e.g.*, Ex-1007; Ex-1008. Indeed, the use of shared libraries was known in the art before the time of the alleged invention claimed in the '058 Patent. *See* Section VI.D. Ely discloses that Alpine is a dynamic shared library. Ex-1006, 7. Using simultaneously accessible shared library components as taught by Levine and Sands to improve Alpine's shared library would require only routine computer programming techniques well within the level of skill of a POSITA at the time, and no hardware

modifications would be required. Specifically, minor changes to Ely's shared library source code would implement these changes. Ely itself discusses how minor code changes, which it gives examples of, can modify the shared library's functionality. Ex-1006, 9-11. Therefore, a POSITA would understand that Ely's combination with the teachings of Levine and Sands would yield predictable results—improving Ely's dynamic shared library with the shared library performance benefits taught by Levine and Sands.

171. A POSITA would have also found these results predictable. Levine and Sands both teach a dynamic shared library system which allows for simultaneously executing applications to utilize private, unique instances of shared library components. Incorporating these teachings into Ely's Alpine system, which utilizes a dynamic shared library, would improve it in the same way, thus yielding predictable results.

172. For the same reasons as discussed above, in my opinion, the combination of the teachings of Levine and Sands with Ely's computing system would be merely applying a known technique to improve similar devices in the same way, and the application of a known technique to a known device ready for improvement to yield predictable results.

173. Further in my opinion, combining Ely's system with the teachings of Levine and Sands would have been obvious to try for their teachings of providing

simultaneous access to private instances of shared library resources. A POSITA would have identified the teachings of Levine and Sands as among the limited solutions available to improve Alpine. A POSITA would have had a reasonable expectation of success because modifying Ely's library to allow for simultaneous instances of the same library component would require only minor changes to the library source code used in Ely's system, as exemplified in Ely itself. Ex-1006, 9-11.

2. **Claim 1**

174. In my opinion, claim 1 is obvious in view of Ely alone or in combination with Levine and/or Sands.

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

175. Ely discloses or renders obvious a computing system. A POSITA would understand a computing system to be a computer or other device(s) upon which computing is performed. Such a device(s) constitutes, at its base, computer hardware and an operating system which allow for computing to be performed. Ely explains that Alpine can be implemented on UNIX operating systems such as FreeBSD and Linux. Ex-1006, 1, 8, 12. Ely further discloses that the invention has a hardware interface, thus indicating the implementation of computer hardware. Ex-1006, 4. A POSITA would have understood that a system which combines an

operating system, software that supports basic computing functions, and hardware, the physical parts of the computer, constitutes a “computing system.”

176. Moreover, Ely discloses that Alpine was tested on 200MHz Pentium-Pro PCs. Ex-1006, 8-9. PCs, or “personal computers,” are computer systems designed for use by an individual. Therefore, PCs are a type of “computing system.” The testing of Alpine on such devices discloses that Alpine may be implemented on a computing system such as the disclosed 200MHz Pentium-Pro PCs, in addition to the UNIX-based computing system it is intended for implementation upon.

177. Further in my opinion, Ely discloses or renders obvious that the computing system is for executing a plurality of software applications. Ely discloses that its system supports an “application [] executing.” Ex-1006, 8. Further, Ely discloses that plural “applications us[e] Alpine.” Ex-1006, 3. In this context, “software application” and “application” refer to the same concept, and the word “software” is simply added for clarity. A POSITA would have understood that in such a system in which multiple applications are utilizing the networking stack, and which an application may execute, multiple applications may execute.

178. Ely discloses that Alpine includes a dynamically linked shared library. Ex-1006, 7. Dynamic linking operates such that the binding of function calls to applications is delayed until an application executes and therefore needs to utilize that function. *See* Section VI.D. A POSITA would have thus understood that in a

system in which dynamic linking is utilized, applications execute. Ely's computing system is therefore for executing multiple, *i.e.* a plurality, of software applications.

b. Claim 1[a]: "a processor;"

179. Ely discloses that its system utilizes a "CPU." Ex-1006, 5. "CPU" stands for central processing unit, which is the most important processor in a computer. Section VI.B. Moreover, it is a physical computer processor.

180. Further, Ely discloses its invention, Alpine, was tested on 200 MHz Pentium-Pro PCs. Ex-1006, 8-9. A "Pentium-Pro" is a type of physical computer processor developed by Intel, thus disclosing that Alpine is implemented on computing systems incorporating processors. Ex-1022.

181. Moreover, Ely discloses that its invention conducts operations including the processing of data packets, the execution of functions, and the execution of applications. Ex-1006, 3, 5, 7. These operations are conducted through the utilization of a processor. *See* Section VI.B. Without a processor, Ely's system would not be able to conduct such operations.

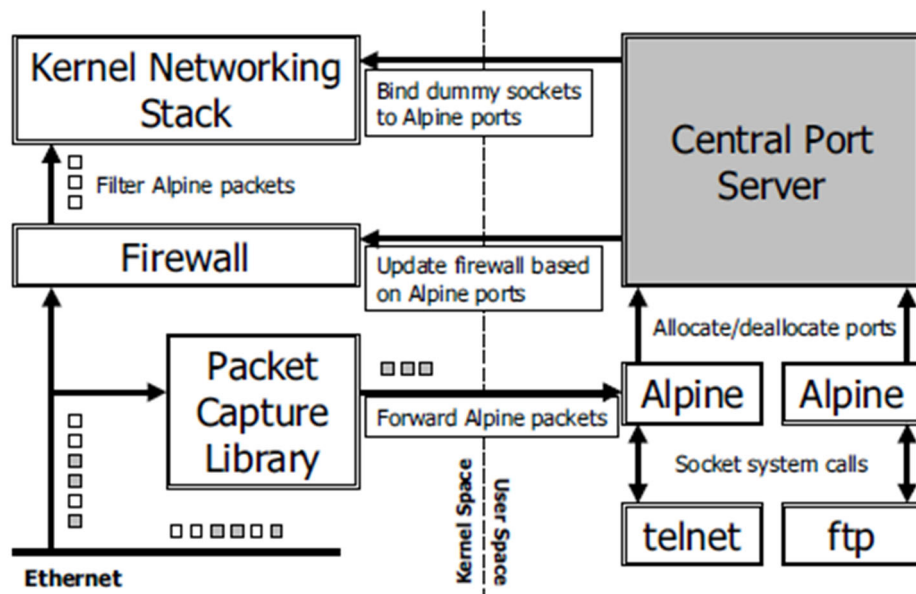
c. Claim 1[b] "an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,"

182. Ely teaches the use of an operating system in its system. Ely explains that its Alpine library is intended for implementation on a form of UNIX operating system, Ex-1006, 1, and that it was implemented on the FreeBSD 3.3 operating

system. Ex-1006, 8. FreeBSD is a type of UNIX operating system. A POSITA would have understood that in order to accommodate the higher-privileged operations conducted by the UNIX operating system, UNIX computing systems (such as FreeBSD 3.3) have and utilize an operating system kernel. *See, e.g.*, Ex-1018, 230 (explaining the use of a kernel in the FreeBSD operating system, referring to it as the “core” of the OS).

183. Ely also teaches that its system has an operating system kernel having OS critical system elements (OSCSEs). The specification of the '058 explains that “critical system elements” (CSEs) are “[a]ny service or part of a service, ‘normally’ supplied by an operating system, that is critical to the operation of a software application.” Ex-1001, 6-8. Further, I understand that Patent Owner has argued this same construction should be applied in the Related IBM Litigation. Ex-1023, Appendix C. These CSEs, when provided by the operating system are operating system CSEs (OSCSEs). Ely discloses such OSCSEs in the form of the protocols contained in the networking stack. The networking stack is a suite of services called “protocols” which enable applications to connect with a network. This is a critical function for the operation of software applications, because network access is a critical requirement for applications and the networking stack is what facilitates and enables such network access. *See* Section VI.E. Without such network access, applications would cease to operate as intended. Further, as Ely explains, the

networking stack is normally supplied by the operating system. Ex-1006, 1 (“[T]he networking stack is traditionally part of the operating system.”). In this implementation, the operating system used (and which provides the networking stack) is FreeBSD, a form of UNIX operating system. Ex-1006, 1. And as Ely also explains, the networking stack exists within the kernel, or in other words, is “kernel-resident.” Ex-1006, 1 (“[T]he networking stack resides in the kernel.”). This is depicted in Figure 3 below.



Ex-1006, Fig.3

184. Because the operating system kernel has a networking stack contained therein, the kernel “has” the networking stack protocols. Indeed, kernel-resident protocols such as the networking stack OSCSEs are controlled by the kernel, such

that the networking protocol functionality is implemented within kernel space as shown by Figure 3. Ex-1006, Fig. 3.

185. Additionally, because Ely's OSCSEs are a kernel-resident suite, they run in kernel mode. Kernel mode, as the '058 Patent explains, is the "context in which the kernel portion of an operating system executes ... Application code cannot run in kernel mode." Ex-1001, 6:32-36. Further, I understand that Patent Owner has agreed in the Related IBM Litigation that "kernel mode" means "[t]he context in which the kernel portion of an operating system executes. Application code cannot run in kernel mode." Ex-1023, Appendix A. The networking stack is run by the kernel, and thus operations in the context in which the kernel portion of the operating system executes. Further, a POSITA would have understood that application code cannot run in kernel mode because its code is designed to only run in the less privileged user mode, and thus is not compiled with the kernel option that would allow it to run in kernel mode. Section VI.B. As kernel-resident services, the networking stack suite instead runs in kernel mode with increased privileges. Ely's discussion of the separation of user mode and kernel mode and the privileges associated with each is consistent with this understanding. *Id.* Therefore, the networking stack OSCSEs are for running in kernel mode.

186. Finally, as I explain, Ely discloses a processor. *See* Section X.B.2.b. Running OSCSEs requires the use of a processor so that the computing system may

interpret and execute the instructions needed to run the OSCSEs, so the OSCSEs in Ely run using Ely's CPU, *i.e.*, "said processor."

d. Claim 1[c]: "a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

187. Ely's system is focused on Alpine, which includes a shared library having shared library critical system elements (SLCSEs) stored therein. As I describe in the Technology Background, at a high level, a shared library is a file that contains executable code designed to be used by the different applications in the computing system. Section VI.D. The code within the shared library is typically code which is utilized by a number of applications, thus making the "sharing" of the code advantageous in that it increases performance and saves space. *Id.* In its lexicography section, the '058 Patent defines a shared library as "[a]n application library code space shared among all user-mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications." Ex-1001, 6:49-53. An application library is defined as "[a] collection of functions in an archive format that is combined with an application to export system elements." I understand that in the Related IBM Litigation, Petitioner argued for this same lexicography of "shared library" with the lexicography of "application library" therein. Ex-1023, Appendix C. A POSITA would have understood and found

obvious that Alpine is such a shared library. Alpine is a collection of protocols, i.e. functions, that are stored in an archive format and are accessed by applications. See Ex-1006, 3, 7. “Archive format” refers simply to the format in which functions within a shared library are stored such that they may be accessed and used by multiple applications at run time. Further, I understand that in the Related IBM Litigation, Patent Owner has argued that “shared library” means “an application library whose code space is shared among all user mode applications.” Ex-1023, Appendix C. Ely emphasizes that Alpine is intended to serve as a replacement for the kernel networking stack and is intended to be integrated into the computing system for use by the system’s “existing applications,” i.e., all user mode applications. Ex-1006, 1, 7. To implement this functionality, Alpine and its shared library files therein are placed in an address space (*i.e.*, the code space) accessible to all applications in user mode. An “application library” is simply a library available to applications. Ely emphasizes that one of the key benefits of Alpine is that it is in user space rather than kernel space, and is thus in a code space different from that occupied by the kernel and its associated files. Ex-1006, 1 (“In traditional operating systems, modifying the network protocol code is a tedious and error-prone task, largely because the networking stack resides in the kernel ... In this paper, we present Alpine, a user-level networking infrastructure.”). Therefore, in my opinion, Alpine includes a shared library as claimed by the ’058 Patent.

188. I further note that the '058 Patent equates shared libraries to dynamic link libraries. Ex-1001, 2:47-48 (“A shared library or dynamic link library (DLL) refers to...”). Ely discloses that Alpine is a shared library that “exploits dynamic linking,” which therefore constitutes a dynamic link library, a term used to refer to shared libraries which utilize dynamic linking. Ex-1006, 7; Section VI.D.

189. Further in my opinion, the components stored within Alpine are CSEs such that when stored in the Alpine shared library, they are shared library CSEs (SLCSEs). Ely’s system takes a networking stack and moves it into a user mode shared library. Ex-1006, 1. The protocols, *i.e.* services, contained in the networking stack are CSEs. Indeed, the networking stack protocols are listed as examples of CSEs by the '058 Patent—the '058 Patent states that CSEs include “[n]etwork services including TCP/IP.” Ex-1001, 6:11-12. Ely explains that the networking stack contained within Alpine includes network services such as TCP and IP. Ex-1006, 3, Fig. 1. Therefore, the CSEs stored within the Alpine shared library constitute SLCSEs.

190. The SLCSEs stored within Alpine are also for use by the plurality of applications in user mode, because they run when applications run, *i.e.*, they run in the context in which applications execute. Ely’s use of the term “user mode” is consistent with the '058 Patent, which defines user mode as the “context in which applications execute.” Ex-1001, 6:31. I understand that Patent Owner also agreed

to this definition of user mode in the Related IBM Litigation. *See* Ex-1023, Appendix A. Dynamic shared libraries, such as Alpine, are designed to be accessible to and used by the plurality of applications in a computing system. Shared libraries are intended to provide resources commonly utilized by multiple applications to those multiple applications, so the shared library is designed to accommodate this by making the contents of the shared library accessible for use by the plurality of applications. Section VI.D. This is supported by Ely, which explains that Alpine's SLCSEs are used by the existing plural applications within the computing system in place of the kernel's networking stack. Ex-1006, 7. Further, Ely does not disclose any restrictions on which applications within the computing system may access Alpine. Applications themselves operate in user-mode by their nature as programs with limited privileges, and Ely explains that the Alpine library too is in user-mode. Ex-1006, 1. Therefore, the SLCSEs stored within Alpine are for use by the plurality of applications in user-mode.

- e. **Claim 1[c][i] “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,”**

191. Ely discloses that SLCSEs stored in Alpine's shared library (*i.e.*, “some of the SLCSEs stored in the shared library) are functional replicas of OSCSEs. The

'058 Patent explains that the term “replica” is used to “denote a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system.” Ex-1001, 1:66-2:1. Further, Patent Owner explained during prosecution of the '058 Patent that SLCSEs are “functional replicas” and “in other words, replacements” for OSCSEs. Ex-1002, 288. Further, I understand that Patent Owner has argued in the Related IBM Litigation that “functional replicas” means “substantial functional equivalents or replacements of kernel functions.” Ex-1023, Appendix C. Ely explains that it takes a FreeBSD kernel networking stack and places it inside the Alpine shared library. Ex-1006, 1, 4. Networking stacks have the same standardized, operating system-independent, protocols, each of which maintains its same function across operating systems, such as TCP, IP, and UDP. *See* Section VI.E. Ely explains that as the Alpine library provides the same function as the OSCSEs, Alpine is linked to by applications before other libraries such that Alpine’s SLCSEs are used instead of the kernel’s OSCSEs, thereby replacing them. Ex-1006, 7. Therefore, the FreeBSD networking stack protocol SLCSEs stored within Alpine are functional replicas of OSCSEs.

192. Further, as I previously explain, the SLCSEs contained within Alpine are accessible to the plurality of applications in Ely’s computing system. *See* Section X.B.2.d.

193. When one of the SLCSEs stored in Alpine is accessed by one or more of the plurality of software applications in Ely's computing system, it forms a part of the one or more of the plurality of applications. The '058 Patent explains that SLCSEs may "form a part of at least some of the software applications by being linked thereto." Ex-1001, 2:14-15. Ely discloses such linking of SLCSEs to software applications as the method through which Ely's plurality of applications access Ely's SLCSEs. Ely explicitly notes that applications link to Alpine using dynamic linking. Ex-1006, 7. As I explain in the Technology Background, when an application links to a shared library, the library's code is loaded to the same address space, becoming a part of the application such that the application can access and execute the library code without a proxy. Section VI.D. By dynamically linking to Alpine, applications are linked thereto to both the library and its components (SLCSEs) therein. For these same reasons, Ely also meets Petitioner's construction of forming a part in the Related IBM Litigation—linked shared library parts are loaded into application address space, and thus reside there, as opposed to simply being a proxy exclusive to the application. *See* Ex-1023, Appendix C. Therefore, when one of the SLCSEs is accessed by one or more of the plurality of applications it forms a part of the one or more of the plurality of software applications.

194. Furthermore, Ely also describes an embodiment of the Alpine infrastructure implemented on the Linux operating system. Ex-1006, 12. Because

Alpine is designed to be portable and rely on minimal platform-specific code, it can be adapted to different operating systems while maintaining its intended functionality. Ex-1006, 12. In my opinion, this implementation of Alpine, which utilizes a FreeBSD kernel, on the Linux operating system would also thus meet this limitation, as well as all the other limitations of the Challenged Claims, for the same reasons as its embodiment on FreeBSD does—how Alpine operates does not change based on the operating system it is implemented on.

- f. **Claim 1[c][ii]: “wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and”**

195. As I previously explain, Ely discloses SLCSEs provided to the plurality of applications in the computing system. *See* Section X.B.2.d. Further, in my opinion, a POSITA would have understood and found obvious that in Ely’s system, an instance of a SLCSE provided to a software application from the shared library is run in a context of that application without being shared with software applications, and where a second software application running under the operating system has its own unique instance of the same SLCSE.

196. Generally, in order to prevent conflict between application code, each application has its own address space. When an application links to a shared library component, *i.e.* SLCSE, that component is memory-mapped into the application's address space. At run time, an instance of the SLCSE is loaded into the calling application's address space. However, the SLCSE stored in the shared library is still available in the shared library to be linked to and called by other applications. Each of these instances is an instance of the same SLCSE, so each instance has the same function—the function of the SLCSE which is instanced. Through an instance of an SLCSE being memory-mapped to and loaded at run time into the application's address space, a POSITA would have understood that an instance of an SLCSE provided to an application is run in the context of that application. A POSITA would have understood that in Ely's Alpine system, which has a dynamic shared library, this system of unique application address spaces and dynamic shared library linking would be present. Ex-1006, 7. Ely discloses that an application and its SLCSEs run in the same process, and therefore in the context of the application associated with that process. Ex-1006, 5. The application address space of each application is private to the application and its related processes, so each of these SLCSE instances provided to an application are equally private and not shared with other applications. All of these applications are running under Ely's UNIX-based operating system. Finally, Ely discloses that its system is a multi-tasking operating system in which

multiple applications can run simultaneously. Ex-1006, 6. Therefore, in my opinion, a POSITA would have understood that Ely alone discloses or renders obvious an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.

197. This limitation is also obvious in view of Levine and/or Sands for the same reasons I explain in Ground I, which are equally applicable to Ely, and for the reasons I explain in the Motivation to Combine section above. *See* Section X.A.2.f, X.B.1. Therefore, in my opinion, Ely in view of Levine and/or Sands renders obvious this limitation.

- g. Claim 1[c][iii]: “wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.”**

198. As I previously explain, Ely discloses or renders obvious SLCSEs stored in a shared library. *See* Section X.B.2.d. These SLCSEs are provided to the plurality of applications in Ely’s system. *See id.* Ely further discloses that its system

is one in which multiple applications may run at the same time. *See* Ex-1006, 6 (describing how “multiple applications try to update the firewall concurrently.”), Fig. 3 (showing two applications, telnet and ftp, simultaneously utilizing Alpine). Each application has its own address space, and each application has a same-function instance of an SLCSE provided to it in its address space. *See* Section X.B.2.f.

199. Each of these simultaneously running applications uses its respective instance of the SLCSE simultaneously. Dynamic libraries, such as Alpine, are designed such that multiple applications may access the shared resources stored therein and such that no conflicts will arise from simultaneously running applications simultaneously utilizing instances of the same shared component. This is because the shared component (SLCSE) is not exclusively utilized by any application; rather, an instance private to the application, due to being within that application’s private address space, is utilized while the SLCSE remains available to other applications.

200. This limitation is also obvious in view of Levine and/or Sands for the same reasons I explain in Ground I, which are equally applicable to Ely, and for the reasons explained in the Motivation to Combine section above. *See* Section X.A.2.g, X.B.1. Therefore, in my opinion, Ely in view of Levine and/or Sands renders obvious this limitation.

3. Claim 2: “A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE

stored in the shared library run simultaneously within the operating system.”

201. As I describe above in the Technology Background, there are two modes in which programs run “within the operating system:” user-mode and kernel-mode. Section VI.B. These provide different levels of privileges to the programs depending on which mode they are running under. Applications run in user-mode, and thus when applications run, they are running “within the operating system.”

202. As I previously discuss, multiple instances of an SLCSE stored in the shared library (Alpine) may run simultaneously within the operating system. Section X.B.2.g. Ely discloses that Alpine and its components are in user-mode, one of the operating system modes. Ex-1006, 1; Section VI.B.

203. Further in my opinion, this limitation is also obvious in view of Levine and/or Sands. As I previously discuss, Levine and/or Sands teach multiple instances of an SLCSE stored in the shared library running simultaneously within the operating system. *See* Section X.A.3. As I also previously discuss, a POSITA would have been motivated to combine Ely with the teachings of Levine and/or Sands. *See* Section X.B.1.

204. These teachings of Ely and Ely in view of Levine and/or Sands are equally applicable to the implementation of Alpine on the Linux operating system. As I previously discuss, Alpine’s functionality does not change depending on which operating system it is implemented on. *See* Section X.B.2.e.

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

205. Ely explains that it is one of the notable design elements of its system is that there are no changes to the kernel. Ex-1006, 1-2. The kernel networking stack, *i.e.* OSCSEs, are kernel resident. Ex-1006, 1. Rather than make changes to the UNIX operating system’s networking stack in the kernel, *i.e.* OSCSEs, Ely utilizes FreeBSD network stack protocols, *i.e.* SLCSEs, that are stored in the Alpine shared library. Ex-1006, 1. As I previously discuss, these SLCSEs are functional replicas of the OSCSEs, and therefore the OSCSEs which remain in the operating system kernel correspond to and are capable of performing the same function as the SLCSEs. *See* Section X.B.2.e. Indeed, the protocols such as IP, TCP, and UDP remain the same across the OSCSEs and SLCSEs. *See id.*

206. Further, Ely makes clear that the entire purpose of the Alpine system is to provide a user-mode networking stack of the kernel-level networking stack. *See* Ex-1006, 1. A POSITA would understand doing so would require no changes to the UNIX kernel Ely’s system is implemented upon.

207. This design is consistent with Ely’s goal—providing an infrastructure which facilitates networking stack development by providing a user-mode networking stack to modify while leaving the kernel networking stack unmodified

in order to avoid the drawbacks associated with modifying a live kernel. Ex-1006, 1.

5. **Claim 4: “A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.”**

208. Ely discloses or renders obvious one or more SLCSEs provided to the plurality of software applications for their exclusive use. *See* Section X.B.2.d.

209. Further, Ely discloses that some SLCSEs make system calls to access services in the operating system kernel to determine if there anything to read out of a socket buffer. Ex-1006, 7 (“Alpine can determine locally if there is anything to read out of the socket buffers, but it must make a select call into the kernel to determine if there is anything to read out of the files.”).

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

210. Ely’s system is implemented on a specific UNIX variant operating system, FreeBSD, and utilizes a FreeBSD networking stack. Ex-1006, 1, 8. However, Ely discloses that Alpine may easily be ported to a different UNIX variant operating system, Linux. Ex-1006, 12. In this Linux embodiment, the kernel networking stack (OSCSEs) has a Linux networking stack while Alpine (SLCSEs) has a FreeBSD networking stack. *See id.* As I previously explain, this Linux embodiment meets the limitations of Claims 1 and 2 for the same reasons as the

FreeBSD embodiment Ely describes does. Section X.B.2.e. Each operating system (including variants) has its own, specific implementation of the networking stack due to the architecture differences and performance needs inherent to each of the various operating systems. *See* Section VI.E. The core functionalities remain the same; indeed, they use the same standardized protocols. However, the way in which the networking stack is implemented across operating system differs. *See id.* Therefore, the networking stack component services are not copies of networking stack components in other operating systems. Instead, they have differences related to how they are implemented and optimized for their respective operating system. *Id.* Therefore, a POSITA would have understood that the FreeBSD networking stack (SLCSEs) are not copies of the Linux kernel networking stack (OSCSEs).

211. Further, Alpine is by design intended to have its networking protocols stored therein modified by developers and tested thereafter. *See* Ex-1006, 1. They are therefore different, and not copies of each other. As Ely explains, providing an infrastructure for developers to modify a kernel networking stack in user-mode is the point of the Alpine system. *See id.* Therefore, a POSITA would understand and find it obvious that these user-mode shared library networking protocols, which are SLCSEs, will be modified versions of what is in the kernel. A POSITA would further understand that these protocols (SLCSEs) still perform the same function of enabling network communications after modification, despite differing from the

original code from the kernel networking stack (OSCSEs). While maintaining the same function, however, they have their code changed such that they are not copies of the OSCSEs they are based off.

XI. SECONDARY CONSIDERATIONS OF NON-OBVIOUSNESS

212. I have been informed that Patent Owner asserts that two secondary considerations of non-obviousness support a finding of non-obviousness of the '058 Patent: commercial success and a long-felt need for the alleged invention. *See* Ex-1028, 11-15. However, Patent Owner offers only conclusory arguments without showing any nexus between the alleged commercial success and long-felt need and the alleged invention of the '058 Patent. I am not aware of any evidence of commercial success of the claimed invention. Moreover, for the reasons I explain in this Declaration, there was no long-felt need for the alleged invention because it had already been done by others, as shown by the Deianov and Ely references. To the extent Patent Owner argues in this proceeding that there are secondary considerations of non-obviousness, I reserve the right to respond.

Declaration of Dr. Mowry for *Inter Partes* Review of U.S. Patent No. 7,784,058

I, Todd C. Mowry, 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 the Title 18 of the United States Code.

Dated: February 5, 2025

Respectfully submitted,



Todd C. Mowry, Ph.D.