

202

COPY

Brent O. Hatch (5715)
HATCH, JAMES & DODGE, PC
10 West Broadway, Suite 400
Salt Lake City, Utah 84101
Telephone: (801) 363-6363
Facsimile: (801) 363-6666

Robert Silver, Esq. (admitted pro hac vice)
BOIES, SCHILLER & FLEXNER LLP
333 Main Street
Armonk, New York 10504
Telephone: (914) 749-8200
Facsimile: (914) 749-8300

Stephen N. Zack (admitted pro hac vice)
Mark J. Heise (admitted pro hac vice)
BOIES, SCHILLER & FLEXNER LLP
Bank of America Tower – Suite 2800
100 Southeast Second Street
Miami, Florida 33131
Telephone: (305) 539-8400
Facsimile: (305) 539-1307

Frederick S. Frei (admitted pro hac vice)
Aldo Noto (admitted pro hac vice)
John K. Harrop (admitted pro hac vice)
ANDREWS KURTH LLP
1701 Pennsylvania Avenue, Ste. 300
Washington, DC 20006
Telephone: (202) 662-2700
Facsimile: (202) 662-2739

Attorneys for Plaintiff The SCO Group, Inc.

IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF UTAH

THE SCO GROUP, INC.

Plaintiff/Counterclaim Defendant

vs.

INTERNATIONAL BUSINESS
MACHINES CORPORATION

Defendant/Counterclaim Plaintiff

**DECLARATION OF SANDEEP
GUPTA IN SUPPORT OF SCO'S
OPPOSITION TO IBM'S MOTION
FOR PARTIAL SUMMARY
JUDGMENT**

Case No. 2:03-CV-0294 DAK

Honorable Dale A. Kimball
Magistrate Judge Brooke C. Wells

DECLARATION OF SANDEEP GUPTA

1. My name is Sandeep Gupta and I am employed by The SCO Group, Inc. My office is located at 430 Mountain Avenue, Murray Hill, NJ 07974. Unless otherwise noted or evident from the context, this declaration is based on my personal knowledge and information available to me from reliable sources. To the best of my knowledge, information and belief, the facts set forth herein are true and correct.
2. I submit this Declaration in support of the SCO's Opposition to IBM's Cross-Motion for Partial Summary Judgment, in the lawsuit entitled The SCO Group, Inc. v. IBM, Civil No. 2:03-CV-0294 DAK (D. Utah 2003).
3. In this declaration, I will explain why I believe that several routines and several groupings of code for which SCO has copyright protection were copied into the Linux operating system. Specifically, this declaration will (1) describe how the Read-Copy-Update routine in Linux is substantially similar to a routine in UNIX; (2) describe how the user level synchronization (ULS) routines in Linux are substantially similar to routines in UNIX; (3) describe how Linux version 2.4.20 contains code that is either an identical or substantially similar copy of SCO's UNIX System V IPC code; (4) identify identical and substantially similar copying of SCO's copyrighted UNIX "header and interfaces" in Linux; (5) describe how Linux version 2.6 contains code that is an identical copy of SCO's UNIX System V init (SYS V init) code; and (6) identify identical copying of SCO's UNIX Executable and Linking Format (ELF) code in Linux.

The declaration will discuss these topics of copyright infringement in the order just described. Although I am not an expert in copyright law, I believe these topics either show copying of code or raise significant factual issues that need to be explored further.

4. SCO claims ownership of copyrights to UNIX software, source code, object code, programming tools, documentation related to UNIX operating system technology, and derivative works thereof. These materials are covered by numerous copyright registrations issued by the United States Copyright Office (the "Copyright Registrations"). These registrations have been obtained by SCO and its predecessors in interest and are owned by SCO. Included among such registrations are:

	TITLE	REG. NO.	REG. DATE
1	UNIX Operating System Edition 5 and Instruction Manual	TXu-510-028	03/25/92
2	UNIX Operating System Edition 6 and Instruction Manual	TXu-511-236	04/07/92
3	UNIX Operating System Edition 32V and Instruction Manual	TXu-516-704	05/15/92
4	UNIX Operating System Edition 7 and Instruction Manual	TXu-516-705	05/15/92
5	Operating System Utility Programs	TXu-301-868	11/25/87
6	UNIXWARE 7.1.3	TX-5-787-679	06/11/03
7	UNIX SYSTEM V RELEASE 3.0	TX-5-750-270	07/07/03
8	UNIX SYSTEM V RELEASE 3.1	TX-5-750-269	07/07/03
9	UNIX SYSTEM V RELEASE 3.2	TX-5-750-271	07/07/03
10	UNIX SYSTEM V RELEASE 4.0	TX-5-776-217	07/16/03
11	UNIX SYSTEM V RELEASE 4.1IES	TX-5-705-356	06/30/03
12	UNIX SYSTEM V RELEASE 4.2	TX-5-762-235	07/03/03
13	UNIX SYSTEM V RELEASE 4.1	TX-5-762-234	07/03/03
14	UNIX SYSTEM V RELEASE 3.2	TX-5-750-268	07/09/03
15	UNIX SYSTEM V RELEASE 4.2 MP	TX 5-972-097	6/29/2004

5. I believe that the Read-Copy-Update ("RCU") routine in Linux version 2.6.5 and in patches to Linux (hereinafter referred to as Linux RCU) are substantially similar to the RCU version in SCO's copyrighted work, specifically UNIX SVR4.2 MP. UNIX SVR4.2 MP is part of a registered copyright of SCO entitled UNIX System V Release 4.2 MP (also known as 4.2 ES/MP) (Registration No. TX 5-972-097).
6. A problem may occur in a multiprocessing computer environment when multiple entities such as multiple processors attempt to access data in a shared memory. For example, a user of the data, such as a "process" in the operating system, may attempt to update a particular piece of data at the same time that another process is attempting to read the same data. In this scenario, the process attempting to read the data will see the data in a partially updated state, or in a non-updated state, rather than in the updated state being supplied by the other process. To address this situation, programmers have devised various synchronization methods that allow access to data by multiple processors at the same time, and that maintain synchronization by directing all processes -- at the appropriate time -- to the updated data.
7. RCU (Read-Copy-Update) is one of the methods used to synchronize access to shared data in a multiprocessing environment.
8. Because RCU provides for synchronized access to shared data in a multiprocessing environment, RCU provides substantial performance

enhancements to an operating system and is thus a very important part of any multiprocessing operating system.

9. SCO's version of RCU first appeared in UNIX SVR4.2 MP and is referred to herein as UNIX RCU.
10. The Linux operating system includes a routine, Linux RCU, that is substantially similar to UNIX RCU.
11. Specifically, Linux RCU and UNIX RCU perform the same five acts:
 - a. Allocating a new data structure of a certain size (this creates the space for updating to be done while other processes continue to read the old data structure);
 - b. Copying the contents of the old data structure to the new data structure (this allows the process to update data while other processes continue to read the old data structure);
 - c. Updating the new data structure;
 - d. Updating or redirecting a pointer to point to the new data structure (this allows new processes to be steered to the updated data structure rather than to the old data structure); and
 - e. Arranging for deferred deletion of the old data structure (deletion cannot occur until all processes accessing the old data during updating have finished accessing the data).
12. There are several ways to synchronize data updating and sharing without using the five acts just described. In other words, synchronizing access to shared data

in a multiprocessing environment can be expressed in ways other than the expressions in UNIX RCU.

13. For example, synchronizing access to shared data can be expressed with mutual exclusion locks (mutex). Mutual exclusion locks may be used to lock a data location so that only one operating system process can have read/write access to that data location at one time. If another operating system process needs to access the same data location, the second operating process has to wait until the data location has been unlocked. Using mutual exclusion locks, access to shared data is seriatim rather than simultaneous. Synchronizing access to shared data can also be expressed with reader/writer locks. Using reader/writer locks, multiple processes can acquire read access to a data location, but only one process at a time can acquire write access to update that data location.
14. UNIX RCU and Linux RCU perform the same five acts in the same sequence, and UNIX RCU and Linux RCU have the same structure and organization.
15. Each of the five acts of the UNIX RCU -- and of the Linux RCU -- routine is expressed in one or a few lines of code.
16. The first act, "allocating a new data structure of a certain size," is expressed in UNIX RCU and Linux RCU by a single line of nearly identical code. From a software programmer's perspective, the UNIX RCU expression of the act of allocating a new data structure has been identically copied into Linux RCU. As can be seen in attached Exhibit A, the Linux RCU code (column 4) for the first act is nearly identical to the UNIX RCU code (column 1) for the first act.

17. The second act, "copying the contents of old data structure to the new data structure," is expressed in UNIX RCU by a single line of code. In Linux, that line of code has been translated from the UNIX system to the Linux system by using a subroutine call instead of the original structure copy -- a stylistic difference, but one which is functionally equivalent and therefore substantially similar to UNIX RCU. From a software programmer's perspective, the second act has been directly translated from UNIX RCU into Linux RCU. *See Exhibit A*, for a comparison of Linux RCU code (column 4) and UNIX RCU code (column 1) associated with the second act.
18. The third act, "updating a new data structure," is substantially similar in Linux RCU and UNIX RCU because both update the data structure by modifying data structure fields. The RCU routine can be applied to different operating system functions. As can be seen in Exhibit A, UNIX applies RCU to assist with the implementation of the resource limits function (`rlimits()`), while the Linux RCU routine is used to assist in the implementation of the grow array function (`grow_ary()`). Because the RCU routine is used as part of the implementation of different functions, the update act is slightly different in UNIX than in Linux. In Linux, in the grow array function, the shared memory location is updated by modifying to null the new fields of the shared memory, while in UNIX, in the resource limits function, the shared memory location is updated by modifying the old data.

19. The fourth act, "updating a pointer," is substantially similar in UNIX RCU and Linux RCU because both UNIX RCU and Linux RCU use pointers to indicate the new updated data structure. In UNIX RCU two pointers are used, while in Linux RCU a single pointer is used. *See* Exhibit A, for a comparison of Linux RCU code (column 4) and UNIX RCU code (column 1) associated with the fourth act.
20. The fifth act, "arranging for deferred deletion of the old data," is achieved somewhat differently in UNIX than in Linux. In UNIX RCU, the fifth act of deferred deletion of the old data structure is achieved by setting flags for the old data which are checked by the operating system at convenient times. If a flag associated with an old data structure shows no current users, the old data structure can be deleted.
21. In Linux RCU, in contrast, the fifth act of deferred deletion is achieved by a callback function that is automatically called when no current users remain so that the old data structure may be deleted.
22. Although the fifth act in Linux RCU and in UNIX RCU are expressed differently, they both achieve deferred deletion of the old data structure.
23. In my opinion, Linux RCU is substantially similar to UNIX RCU, and appears to be derived from UNIX RCU.
24. As represented to me, contributors to Linux had access to UNIX RCU. Showing access to UNIX RCU is a two step process. First, UNIX RCU was copied into Dynix, which is Sequent's version of UNIX, and then UNIX RCU

was released from Dynix into Linux. Sequent Computer System, Inc.'s (Sequent) software engineers could have accomplished the first step -- of developing a Dynix RCU based on the UNIX RCU -- when they worked under the Multiprocessor Software Cooperation Agreement (the "MP Agreement" -- Exhibit B) with Unix System Laboratories, Inc. (USL) engineers to develop the UNIX version of RCU. The MP Agreement was signed in September of 1990. USL is a predecessor of SCO.

25. Jack Slingwine and Paul McKenney are the credited authors of Dynix RCU, and were both Sequent employees. At least Mr. Slingwine was involved in the UNIX development work under the MP Agreement. At least Mr. Slingwine had access to the UNIX RCU work because of his involvement in the UNIX development work. I believe that Mr. Slingwine would have used that access to UNIX development to review UNIX RCU because of his clear interest in RCU. Regarding his clear interest in RCU, Mr. Slingwine and Mr. McKenney authored a paper on RCU, "Read-Copy Update : Using Execution History To Solve Concurrency Problems," which refers to "work performed at Sequent." See Exhibit C. In this paper, Mr. Slingwine and Mr. McKenney thank (among others) Brent Kingsbury, and Mr. Kingsbury was one of the authors of a design document for UNIX which discusses, among other things, UNIX RCU.
26. As represented to me, evidence that the Sequent RCU work was performed during the same time frame that the MP Agreement was operative is clear. Specifically in that regard, the MP Agreement, signed in September 1990,

appears to terminate in November 1992. It took two years to develop UNIX RCU under the MP Agreement, from late 1990 to late 1992. On July 19, 1993, a patent application related to RCU entitled "Apparatus and Method for Achieving Reduced Overhead Mutual Exclusion and Maintaining Coherency in a Multiprocessor..." was filed listing Mr. Slingwine and Mr. McKenney as co-inventors. The patent issued on August 19, 1995 as U.S. Patent No. 5,442,758 and lists Sequent as the assignee. *See* Exhibit D.

27. In sum, at least Mr. Slingwine (and perhaps Mr. McKenney) had access to UNIX developments during the USL/Sequent collaboration under the MP Agreement, which included development of UNIX RCU, and both showed great interest in RCU by filing a patent application (as co-inventors) relating to RCU immediately after the MP Agreement collaboration.
28. Thus, I believe that UNIX RCU was copied into another version of UNIX known as Dynix by engineers who worked for Sequent at the time and who later worked for IBM when IBM acquired Sequent.
29. As far as the second step, IBM thereafter released Dynix, with a copied and modified UNIX RCU in Dynix, into Linux. More specifically, the Dynix version of RCU was used by IBM employee (and former Sequent employee) Dipankar Sarma to create a software patch for placing a substantially similar version of RCU into Linux. I believe that the first patch appears to be for Linux version 2.4.1 and was contributed by Mr. Sarma. *See* Exhibit E. A paper entitled "Read-Copy Update" also lists Mr. Sarma along with Mr. McKenney

and others as authors. *See* Exhibit F. Another patch was also provided to Linux version 2.5.44 by IBM employee Mingming Cao. *See* Exhibit G. This patch appears to be incorporated into the Linux version 2.6.

30. I will now describe the user level synchronization (ULS) routines that are used for blocking and unblocking of processes. I believe that the ULS routines for blocking and unblocking of computer processes in Linux version 2.6 are substantially similar to SCO's ULS in UNIX SVR4.2 MP.
31. The ULS routines in Linux are commonly referred to as FUTEX (Fast User Mutex), but will be called Linux ULS here.
32. The ULS routines in UNIX are sometimes referred to as usync, but will be called UNIX ULS here.
33. The main purpose of the ULS routines is to facilitate inter-process synchronization by blocking and unblocking processes attempting to access shared data.
34. Synchronization of user level processes accessing shared data is important to prevent two processes from modifying the same data at the same time. The blocking and unblocking of access to shared data by ULS allows only one process at a time to use the shared data. ULS is a very significant piece of code because it is a very important part of any operating system to synchronize access by processes to shared data.
35. SCO's version of ULS first appeared in UNIX SVR4.2 MP.
36. Linux ULS and UNIX ULS include the same nine (9) acts:

- a. Given a pointer to the process address space and a user virtual address, acquire the pointer to the descriptor for the region within that process address space where the user virtual address is located;
 - b. Check for shared versus private mappings;
 - c. From the region descriptor, acquire the pointer to the file system node;
 - d. Calculate the offset relative to the beginning of the memory segment;
 - e. For blocking: Using the N-tuple consisting of the file system node and offset acquired in acts c and d, find or allocate a waitqueue entry in the hash table (for unblocking, skip from act d to act h);
 - f. Add the process to the list associated with the waitqueue entry;
 - g. Block the process;
 - h. Using the N-tuple consisting of the file system node pointer and offset acquired in acts c and d, find the head of the queue of blocked processes;
and
 - i. Traverse the queue, unblock the processes, and count the number of processes that are unblocked.
37. Six of the nine acts in Linux ULS and UNIX ULS are implemented substantially similarly.
38. Each blocking sequence has seven (7) acts and each unblocking sequence has six (6) acts. The first four (4) acts of the Linux ULS and UNIX ULS sequences are the same as each other, and are the same whether the routine is for blocking or unblocking. The blocking sequence is listed in Exhibit H and the unblocking

sequence is listed in Exhibit I. As can be seen in Exhibits H and I, there are four common acts to begin, acts one (1) through four (4), and then acts five through seven (5, 6, 7) are used to perform blocking, or acts eight (8) and nine (9) are used to perform unblocking. Exhibit J shows the ULS symbols that are used in both UNIX ULS and Linux ULS. A description of each symbol is also provided in Exhibit J.

39. Each act in the Linux and UNIX ULS routines correspond to one or a just a few lines of code and the acts are in effect the implementation of the routine.
40. For example, the first act "...acquire the pointer to the descriptor..." is represented in UNIX ULS and Linux ULS by a single line of code that has been translated from UNIX to Linux using different names for functions and variables. *See* Exhibits H through J. From a software programmer's perspective, the expression of the act of finding the region descriptor has been identically copied into Linux ULS.
41. The second act, "check for shared versus private mappings" is an individual act represented by a single line of code which is very similarly implemented in Linux and UNIX.
42. The third act of "[f]rom the region descriptor, acquire the pointer to the file system node" is implemented in a substantially similar way because both Linux ULS and UNIX ULS have a region descriptor and acquire the file system pointer to the file system node from the region descriptor (in two lines of substantially similar code).

43. The fourth act is implemented substantially similar because both UNIX ULS and Linux ULS in a few lines of code determine the offset in memory relative to the beginning of the memory segment.
44. Act five "...find or allocate a waitque entry . . ." is implemented by both Linux and UNIX in one line of substantially similar code.
45. Act eight "... find the head of the queue of blocked processes" is implemented in one UNIX line of code and two Linux lines of code which are substantially similar.
46. Linux ULS is substantially similar to UNIX ULS since the nine acts are the same and the specific implementation or expression of six of the nine acts is substantially similar.
47. There are several ways to implement the blocking and unblocking for ULS. Therefore, the result of ULS can be achieved in several other ways. One example of a different implementation of ULS is IBM's implementation in AIX.
48. Another alternative implementation of blocking and unblocking for ULS is the Linux implementation that existed prior to Linux copying the UNIX ULS implementation. This alternative ULS implementation is described in the article attached as Exhibit K. In this article, Rusty Russell, of IBM, describes how the earlier Linux ULS implementation was improved by Jamie Lokier and Hugh Dickins in the new Linux ULS implementation.
49. While working on changing the Linux ULS, Mr. Lokier learned of the UNIX ULS implementation, may have had access to it, and may have incorporated the

infringing code into Linux. Some evidence of this is the changelog to the current Linux ULS code which designates Mr. Lokier as author of the Linux ULS code. Mr. Lokier's author's notes in the changelog give special thanks to Mr. Russell of IBM and to Hugh Dickins (a former SCO employee) for help with the patch to the Linux ULS code which Mr. Lokier authored. I believe that the collaboration between Jamie Lokier and Hugh Dickins to improve the Linux ULS code benefited from access to and knowledge of the UNIX ULS implementation.

50. I now turn to instances of copying in Linux of UNIX System V IPC routines. Linux's implementation of System V IPC in version 2.4.20 (hereinafter Linux SysVIPC) contains code that is either an identical or a substantially similar copy of the UNIX System V IPC routines (hereinafter UNIX System V IPC).
51. IPC stands for Inter-Process Communication. UNIX System V IPC is used to communicate and synchronize between operating system processes on the same machine in a multiprocessing environment. UNIX System V IPC consists of three mechanisms: message queues, semaphores, and shared memory. In addition, four header files are associated with UNIX System V IPC: <ipc.h>, <sem.h>, <msg.h>, and <shm.h>. SCO's version of IPC first appeared in UNIX III Release 4.1 in 1980, which was incorporated in all releases of UNIX System V, including UNIX System V Release 3.2 (hereinafter UNIX SVR3.2) (Registration No. Tx 5-750-271 and Tx 5-750-268).

52. Linux SysVIPC is identical to, or substantially similar to, UNIX System V IPC because (1) the organization of Linux SysVIPC is identical to UNIX System V IPC, both consisting of three mechanisms: message queues, semaphore, and shared memory; (2) the structure of each of the three IPC mechanisms is identical – each IPC mechanism includes identical routines; and (3) three of the four header files associated with Linux SysVIPC are substantially similar to UNIX System V IPC, in that: a) the names of the four header files are identical in Linux SysVIPC and UNIX System V IPC; and b) the code in three header files of Linux SysVIPC is essentially identical to the code in the corresponding header files of UNIX System V IPC.
53. With regard to the organization of Linux SysVIPC and UNIX System V IPC, both consist of three mechanisms: message queues, semaphore, and shared memory. There is no reason for the organization to be identical other than the fact that Linux SysVIPC has been copied from UNIX System V IPC. In addition, the structure of each of the three IPC mechanisms is identical. Message queues in both Linux SysVIPC and UNIX System V IPC consist of four routines: `msgctl()`, `msgget()`, `msgrcv()`, and `msgsnd()`. Semaphores in both Linux SysVIPC and UNIX System V IPC consist of three routines: `semctl()`, `semget()`, and `semop()`. Shared memory in both Linux SysVIPC and UNIX System V IPC consists of four routines: `shmctl()`, `shmget()`, `shmdt()`, and `shmat()`. Again, the identical structure of each of the three IPC mechanisms shows Linux's copying of UNIX System V IPC.

54. In addition to the organization and the structure of the mechanisms, the header files associated with Linux SysVIPC are substantially similar to the header files associated with UNIX System V IPC. Both Linux SysVIPC and UNIX System V IPC include the same four header files: <ipc.h>, <sem.h>, <msg.h>, and <shm.h>.
55. With regard to these header files, evidence of identical copying can be seen by comparing the code from three of the four UNIX System V IPC header files with code in Linux, as shown in Tables 1 through 6 immediately below. Tables 1 through 6 show a portion of UNIX System V IPC header file code side-by-side with a portion of Linux SysVIPC header file code, and show identical copies of the UNIX copyrighted code in Linux. The UNIX System V IPC header file code comes from the three UNIX System V IPC header files: sem.h, msg.h, and shm.h and from UNIX System V IPC tuning files master.d/sem, master.d/msg, master.d/shm, which are associated with the header files sem.h, msg.h, shm.h, respectively. Each tuning file specifies parameters to control the operation of the corresponding header file. The Linux header file code comes from the Linux version specified in the heading of each respective Table 1 through 6. In the code presented in Tables 1 through 6, the comments (the text beginning with “/*” and ending with “*/”) and other irrelevant information have been omitted for clarity. Also, in some cases, the code lines have been reordered for convenience. The code with comments and other information can be seen in the attached Exhibits L through T.

56.

TABLE 1

UNIX SVR3.2-sem.h	Linux-2.4.20-sem.h
struct sem *sem_base	struct sem *sem_base
<pre> struct seminfo { int semmap, semmni, semmns, semmnu, semmsl, semopm, semume, semusz, semvmx, semaem; }; </pre>	<pre> struct seminfo { int semmap; int semmni; int semmns; int semmnu; int semmsl; int semopm; int semume; int semusz; int semvmx; int semaem; }; </pre>

57.

TABLE 2

UNIX SVR3.2-master.d/sem	Linux-2.4.20-sem.h
SEMMAP	#define SEMMAP
SEMMNI	#define SEMMNI
SEMMNS	#define SEMMNS
SEMMNU	#define SEMMNU
SEMMSL	#define SEMMSL
SEMOPM	#define SEMOPM
SEMUME	#define SEMUME
SEVMX	#define SEVMX
SEMAEM	#define SEMAEM

58.

TABLE 3

UNIX SVR3.2-msg.h	Linux-2.4.20-msg.h
<pre> struct msginfo { int msgmap, msgmax, msgmnb, msgmni, msgssz, msgtql; ushort msgseg; }; </pre>	<pre> struct msginfo { int msgpool; int msgmap; int msgmax; int msgmnb; int msgmni; int msgssz; int msgtql; unsigned short msgseg; }; </pre>
<pre> struct msg msg_first struct msg msg_last ushort msg_cbytes struct msgbuf { long mtype char mtext[1] </pre>	<pre> struct msg msg_first struct msg msg_last unsigned short msg_cbytes struct msgbuf { long mtype char mtext[1] </pre>

59.

TABLE 4

UNIX SVR3.2-master.d/msg	Linux-2.4.20-msg.h
MSGMAP	#define MSGMAP
MSGMAX	#define MSGMAX
MSGMNB	#define MSGMNB
MSGMNI	#define MSGMNI
MSGSSZ	#define MSGSSZ
MSGTQL	#define MSGTQL
MSGSEG	#define MSGSEG

60.

TABLE 5

UNIX SVR3.2-shm.h	linux-2.4.20-shm.h
<pre>struct shminfo { int shmmx, shmmn, shmmni, shmseg, shmll; };</pre>	<pre>struct shminfo { int shmmx; int shmmn; int shmmni; int shmseg; int shmll; };</pre>
<pre>#define SHM_R 0400 #define SHM_W 0200</pre>	<pre>#define SHM_R 0400 #define SHM_W 0200</pre>

61.

TABLE 6

UNIX SVR3.2-master.d/shm	linux-2.4.20-shm.h
<pre>SHMMAX SHMMIN SHMMNI SHMSEG SHMALL</pre>	<pre>#define SHMMAX #define SHMMIN #define SHMMNI #define SHMSEG #define SHMALL</pre>

62. SCO owns valid copyrights for UNIX System V IPC header file code. As can be seen from Tables 1-6, SCO's UNIX System V IPC header file code has been identically copied into Linux. Slight variations in some of the terms are insubstantial differences in programming. As to access, UNIX System V IPC header file code was included in any system running a UNIX System V derived operating system. Therefore, anyone having access to a system installed with UNIX System V or one or more of its derivatives could have had access to UNIX System V IPC header file code.

63. I will now describe certain UNIX System V headers and interfaces that are copied either identically or substantially similarly in Linux.
64. Regarding access to this header and interface code, UNIX header and interface source code is available in SCO-copyrighted documentation, such as manual pages. These manual pages are published on the Internet with copyright notices. Also, UNIX header and interface code is available to any entity having a license to UNIX, or who can otherwise access the UNIX code.
65. A header file is a programming source file containing declarations of interfaces that facilitate communication between different regular source files that comprise the program. The interfaces can come from the program itself, or from libraries. The libraries provide (define) the interfaces that are intended to be used elsewhere, typically in an application.
66. At least three header files from UNIX System V: eti.h, form.h, and menu.h, are completely and substantially copied into Linux.
67. Header eti.h in UNIX System V contains code that is identically copied into Linux. Exhibit U shows that eti.h macro definitions are identically copied into Linux. As can be seen from Exhibit U, the sequence of and values given the macros are identical even though not essential parts of the code. The only "difference" between the two code sets is that in Linux, the values of the macros are enclosed in parenthesis, which is an inconsequential change.
68. UNIX System V header file form.h includes code that is identically copied into Linux. Exhibit V compares UNIX and Linux code for the form.h "PAGE" data

structure. This comparison shows that Linux has the exact same four members of the PAGE data structure as are present in UNIX, and no other members. The only "difference" is that Linux uses a "short int" instead of a "plain int" for the integer size of these four members. The term "short" refers to a 16-bit field and "plain" refers to a 32-bit field. Furthermore, the comments are strikingly similar.

69. The PAGE data structure is not named in any documentation, and there is no reason why Linux would need to express the PAGE data structure in the same way as the PAGE data structure is expressed in UNIX.
70. As shown in Exhibit W, the form.h "FIELD" data structure is identically copied from UNIX System V into Linux. The UNIX System V and Linux data structures consist of the same member names (and no others) in the same order and mostly with the same types. The only difference is the use of "short int" instead of "plain int", use of "void *" as a generic pointer-to-object instead of the historic "char *" found in UNIX System V, and the "Field_Options" typedef name instead of "OPTIONS". These differences are insignificant.
71. Further, the UNIX System V form.h "FORM" data structure is identically copied into Linux. The UNIX System V and Linux data structures contain the same member names (and no others) in the same order and the same (or strongly related) types are declared. Any differences are trivial. Exhibit X shows UNIX System V form.h file, lines 87 – 116 compared to Linux, lines 111 – 140.

72. Finally, as to the header and interface files, Exhibit Y shows that Linux includes an identical copy of UNIX System V header file menu.h. As can be seen from Exhibit Y, the Linux version of the menu.h header file uses exactly the same hexadecimal values for the listed macros, with the only difference being that the Linux version encloses the values in parentheses, an inconsequential difference.
73. I now turn to instances of copying in Linux of SCO's copyrighted SYS V init code. Linux version 2.6 contains code that is an identical copy of SYS V init code.
74. SYS V init was accessible for copying because the manual pages defining SYS V init features, for example init and inittab, are published as electronic documents and are available to anyone with an Internet browser. These manual pages, however, carry appropriate copyright notices. Using the manual pages, a skilled programmer could copy the structure, sequence, and organization of SYS V init routines. SYS V init and inittab were included in documentation with each release of UNIX System V as manual pages init(1M) and inittab(4), respectively. See Exhibits Z and AA. Also, SYS V init code is available to any entity having a license to UNIX, or who can otherwise access the UNIX code.
75. SYS V init code is a general process spawner that, when executed, creates processes from information stored in an inittab file. Init is the first user-level process activated after the kernel completes initialization during computer system startup. Init then starts all subsequent processes on the computer system. These processes include checking integrity of file systems, mounting

file systems, starting background processes to handle printing and networking, and initiating user logins.

76. An example of identical copying from UNIX SYS V init into Linux can be seen by comparing action keywords. Such a comparison is shown in Exhibit BB. As can be seen from the UNIX SYS V init column, every action keyword in UNIX SYS V init is identically copied into Linux.
77. I will now describe certain UNIX Executable and Linking Format (ELF) code that is copied either identically or substantially similarly in Linux.
78. The ELF code is used for processing executables and object code. The ELF code is found in elf.h files in both UNIX and Linux. Tables 7 – 11 show ELF code that is identical in UNIX and Linux. These elf.h files come from UNIX System V, Release 4.2 MP (UNIX SVR4.2 MP) and Linux version 2.4.21, respectively. In the code presented in Tables 7 - 11, comments (i.e., text beginning with “/*” and ending with “*/”) and other irrelevant information have been omitted for clarity. Also, in some cases, the code lines and/or code sections have been reordered for convenience. The code with comments and other information can be seen in the attached Exhibits CC (UNIX code) and DD (Linux code).
79. Table 7 below shows the side-by-side comparison between SCO’s ELF code (UNIX elf.h) and the Linux elf.h code (Linux – elf.h code).

TABLE 7

UNIX - elf.h	Linux - elf.h
<pre> unsigned char e_ident[EI_NIDENT]; Elf32_Half e_type; Elf32_Half e_machine; Elf32_Word e_version; Elf32_Addr e_entry; Elf32_Off e_phoff; Elf32_Off e_shoff; Elf32_Word e_flags; Elf32_Half e_ehsize; Elf32_Half e_phentsize; Elf32_Half e_phnum; Elf32_Half e_shentsize; Elf32_Half e_shnum; Elf32_Half e_shstrndx; } Elf32_Ehdr; Elf32_Word p_type; Elf32_Off p_offset; Elf32_Addr p_vaddr; Elf32_Addr p_paddr; Elf32_Word p_filesz; Elf32_Word p_memsz; Elf32_Word p_flags; Elf32_Word p_align; } Elf32_Phdr; #define SHT_NULL 0 #define SHT_PROGBITS 1 #define SHT_SYMTAB 2 #define SHT_STRTAB 3 #define SHT_RELA 4 #define SHT_HASH 5 #define SHT_DYNAMIC 6 #define SHT_NOTE 7 #define SHT_NOBITS 8 #define SHT_REL 9 #define SHT_SHLIB 10 #define SHT_DYNSYM 11 #define SHT_NUM 12 #define SHT_LOUSER 0x80000000 #define SHT_HIUSER 0xffffffff </pre>	<pre> unsigned char e_ident[EI_NIDENT]; Elf32_Half e_type; Elf32_Half e_machine; Elf32_Word e_version; Elf32_Addr e_entry; Elf32_Off e_phoff; Elf32_Off e_shoff; Elf32_Word e_flags; Elf32_Half e_ehsize; Elf32_Half e_phentsize; Elf32_Half e_phnum; Elf32_Half e_shentsize; Elf32_Half e_shnum; Elf32_Half e_shstrndx; } Elf32_Ehdr; Elf32_Word p_type; Elf32_Off p_offset; Elf32_Addr p_vaddr; Elf32_Addr p_paddr; Elf32_Word p_filesz; Elf32_Word p_memsz; Elf32_Word p_flags; Elf32_Word p_align; } Elf32_Phdr; #define SHT_NULL 0 #define SHT_PROGBITS 1 #define SHT_SYMTAB 2 #define SHT_STRTAB 3 #define SHT_RELA 4 #define SHT_HASH 5 #define SHT_DYNAMIC 6 #define SHT_NOTE 7 #define SHT_NOBITS 8 #define SHT_REL 9 #define SHT_SHLIB 10 #define SHT_DYNSYM 11 #define SHT_NUM 12 #define SHT_LOUSER 0x80000000 #define SHT_HIUSER 0xffffffff </pre>

UNIX - elf.h	Linux - elf.h
<pre>#define SHF_WRITE 0x1 #define SHF_ALLOC 0x2 #define SHF_EXECINSTR 0x4 #define SHF_MASKPROC 0xf0000000</pre>	<pre>#define SHF_WRITE 0x1 #define SHF_ALLOC 0x2 #define SHF_EXECINSTR 0x4 #define SHF_MASKPROC 0xf0000000</pre>
<pre>#define SHN_UNDEF 0 #define SHN_LORESERVE 0xff00</pre>	<pre>#define SHN_UNDEF 0 #define SHN_LORESERVE 0xff00</pre>
<pre>#define SHN_ABS 0xffff1 #define SHN_COMMON 0xffff2 #define SHN_HIRESERVE 0xffff</pre>	<pre>#define SHN_ABS 0xffff1 #define SHN_COMMON 0xffff2 #define SHN_HIRESERVE 0xffff</pre>

TABLE 7 (cont.)

81. The data shown in Tables 8 -11 are additional examples of copying of UNIX elf.h code in Linux.

82.

TABLE 8

UNIX - elf.h		Linux - elf.h	
#define EI_MAG0	0	#define EI_MAG0	0
#define EI_MAG1	1	#define EI_MAG1	1
#define EI_MAG2	2	#define EI_MAG2	2
#define EI_MAG3	3	#define EI_MAG3	3
#define EI_CLASS	4	#define EI_CLASS	4
#define EI_DATA	5	#define EI_DATA	5
#define EI_VERSION	6	#define EI_VERSION	6
#define EI_PAD	7	#define EI_PAD	7
#define ELF_MAG0	0x7f	#define ELF_MAG0	0x7f
#define ELF_MAG1	'E'	#define ELF_MAG1	'E'
#define ELF_MAG2	'L'	#define ELF_MAG2	'L'
#define ELF_MAG3	'F'	#define ELF_MAG3	'F'
#define ELF_MAG	"\177ELF"	#define ELF_MAG	"\177ELF"
#define SELFMAG	4	#define SELFMAG	4
#define ELFCLASSNONE	0	#define ELFCLASSNONE	0
#define ELFCLASS32	1	#define ELFCLASS32	1
#define ELFCLASS64	2	#define ELFCLASS64	2
#define ELFCLASSNUM	3	#define ELFCLASSNUM	3
#define ELFDATANONE	0	#define ELFDATANONE	0
#define ELFDATA2LSB	1	#define ELFDATA2LSB	1
#define ELFDATA2MSB	2	#define ELFDATA2MSB	2
#define EV_NONE	0	#define EV_NONE	0
#define EV_CURRENT	1	#define EV_CURRENT	1
#define EV_NUM	2	#define EV_NUM	2

83.

TABLE 9

UNIX - elf.h	Linux - elf.h
Elf32_Word st_name;	Elf32_Word st_name;
Elf32_Addr st_value;	Elf32_Addr st_value;
Elf32_Word st_size;	Elf32_Word st_size;
unsigned char st_info;	unsigned char st_info;
unsigned char st_other;	unsigned char st_other;
Elf32_Half st_shndx;	Elf32_Half st_shndx;
} Elf32_Sym;	} Elf32_Sym;

84.

TABLE 10

UNIX - elf.h	Linux - elf.h
#define STB_LOCAL 0	#define STB_LOCAL 0
#define STB_GLOBAL 1	#define STB_GLOBAL 1
#define STB_WEAK 2	#define STB_WEAK 2
#define STT_NOTYPE 0	#define STT_NOTYPE 0
#define STT_OBJECT 1	#define STT_OBJECT 1
#define STT_FUNC 2	#define STT_FUNC 2
#define STT_SECTION 3	#define STT_SECTION 3
#define STT_FILE 4	#define STT_FILE 4

85.

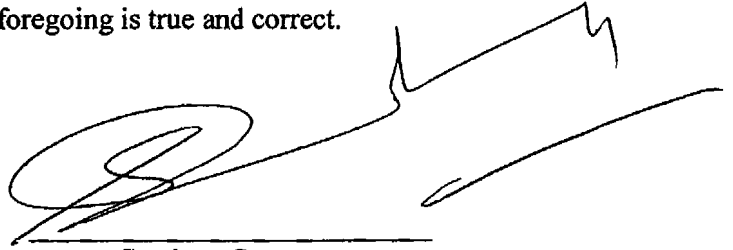
TABLE 11

UNIX - elf.h	Linux - elf.h
Elf32_Addr r_offset;	Elf32_Addr r_offset;
Elf32_Word r_info;	Elf32_Word r_info;
} Elf32_Rel;	} Elf32_Rel;
Elf32_Addr r_offset;	Elf32_Addr r_offset;
Elf32_Word r_info;	Elf32_Word r_info;
Elf32_Sword r_addend;	Elf32_Sword r_addend;
} Elf32_Rela;	} Elf32_Rela;

86. As can be seen by looking at Tables 7-11, portions of the Linux elf.h code are identical or at least substantially similar to portions of UNIX elf.h code. Moreover, the portions of Linux that are direct copies of UNIX are the meaningful, functional portions of the code (*e.g.*, the data type (*e.g.*, ELF32_Half) and member (*e.g.*, e_type) names).

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

July 7, 2004

A handwritten signature in black ink, consisting of a large, stylized initial 'S' followed by a series of connected loops and a long horizontal stroke extending to the right.

Sandeep Gupta

CERTIFICATE OF SERVICE

Plaintiff, The SCO Group, hereby certifies that a true and correct copy of
**DECLARATION IN SUPPORT OF SCO'S OPPOSITION TO IBM'S CROSS-MOTION
FOR PARTIAL SUMMARY JUDGMENT** was served on Defendant International Business
Machines Corporation on the 9th day of July, 2004, as follows:

BY HAND DELIVERY:

Alan L. Sullivan, Esq.
Todd M. Shaughnessy, Esq.
Snell & Wilmer L.L.P.
15 West South Temple, Ste. 1200
Salt Lake City, Utah 84101-1004

Evan R. Chesler, Esq.
Cravath, Swaine & Moore LLP
825 Eighth Avenue
New York, NY 10019

Donald J. Rosenberg, Esq.
1133 Westchester Avenue
White Plains, New York 10604

A handwritten signature in cursive script, appearing to read "James Campbell", is written over a horizontal line.