

588

SNELL & WILMER LLP
Alan L. Sullivan (3152)
Todd M. Shaughnessy (6651)
15 West South Temple
Gateway Tower West
Salt Lake City, Utah 84101-1004
Telephone: (801) 257-1900
Facsimile: (801) 257-1800

CRAVATH, SWAINE & MOORE LLP
Evan R. Chesler (admitted pro hac vice)
David R. Marriott (7572)
Worldwide Plaza
825 Eighth Avenue
New York, New York 10019
Telephone: (212) 474-1000
Facsimile: (212) 474-3700

*Attorneys for Defendant/Counterclaim-Plaintiff
International Business Machines Corporation*

IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF UTAH

THE SCO GROUP, INC.

Plaintiff/Counterclaim-Defendant,

-against-

INTERNATIONAL BUSINESS MACHINES
CORPORATION,

Defendant/Counterclaim-Plaintiff.

**REBUTTAL DECLARATION
OF RANDALL DAVIS**

Civil No. 2:03CV-0294 DAK

Honorable Dale A. Kimball

Magistrate Judge Brooke C. Wells

FILED UNDER SEAL

I. INTRODUCTION

1. I have been asked by counsel for IBM to respond to the Declaration of Marc Rochkind submitted by The SCO Group, Inc. ("SCO") in opposition to IBM's motion to limit SCO's claims. This declaration is limited to responding to the issues raised by Mr. Rochkind.

2. Like my declaration dated March 29, 2006, this declaration is based on my experience working in the field of computer science and evaluating allegations of intellectual property violations. I further base the facts and opinions set out in this declaration upon careful review of SCO's Final Disclosure of Allegedly Misused Material dated December 22, 2005 (the "Final Disclosures").

3. In summary, the Rochkind declaration talks past the conclusions set out in my previous declaration. While Mr. Rochkind states that he "strongly disagree[s] with [my] assertion (at paragraph 11) that SCO has failed to identify with specificity 198 challenged Items in the December submission" (Rochkind Decl. ¶ 7), he fails to directly confront the facts set out in my declaration and his conclusion is clearly based on a very different inquiry. Put differently, Mr. Rochkind reached a different conclusion than I did because he answered a different question.

4. If the question is whether SCO provided version, file and line information for each of the 198 Items at issue, then the answer is unquestionably "No". Although Mr. Rochkind uses words that might be understood to suggest that these coordinates have been provided where possible (Rochkind Decl. ¶¶ 11-12), they have not been. Notably, he makes no effort to show that they have. Without version, file and line information concerning SCO's allegations, it is simply not possible fully to understand its claims, which puts IBM at an extraordinary disadvantage.

5. If the question is whether version, file, and line information is still needed even where the allegations concern misuse of methods and concepts (rather than copying of code), the answer is unquestionably “Yes.” It is entirely possible for the party making the allegations of misuse to assemble such information: Where a method or concept is in fact used in a program, there must be lines of source code in the program that implement the method or concept. The alleging party need simply cite the program version, file, and lines in which that source code appears.

6. In the next section of this declaration, I explain some basic background that helps in understanding why Mr. Rochkind’s position is untenable. Section III details the fundamental disagreement between my original declaration and the Rochkind declaration. In Section IV, I respond to Mr. Rochkind’s assertions that the 198 Items are essentially all methods and concepts claims and that such claims do not require version, file and line information. In Section V, I address Mr. Rochkind’s assertion that IBM has more than enough information to defend itself. Finally, in Section VI, I consider Mr. Rochkind’s claim regarding “willfulness”.

II. IMPORTANT BACKGROUND

7. Several basic facts concerning this case are worth reviewing briefly to help make clear the difficulties presented by SCO’s position on disclosure.

8. First, SCO has alleged misuse of its intellectual property, claiming, among other things, that its System V Unix code was copied by IBM into AIX and/or Dynix and then contributed, with and without AIX and Dynix code, by IBM to Linux. For these allegations to be understood, SCO must (a) specify the System V code that was allegedly copied, (b) specify where in AIX and/or Dynix the code allegedly derived from System V appears, and (c) specify where in Linux the allegedly infringing code appears.

9. Without such specification, how can IBM respond, much less prepare a defense? How, for instance, can it determine whether the code allegedly copied from System V is in fact protectable, or instead is unoriginal; an idea, process or procedure; dictated by externalities; or in the public domain? Such an analysis must proceed from the specific code, and absent an indication of what code is in question, the analysis cannot even begin.

10. Second, the volume of code at issue in this case is so enormous as to make it pragmatically impossible to determine what code might be in question unless the version, file and lines are specified. To put this in more familiar terms, consider that a single recent version of Linux contains about 6 million lines which if printed would be about 110,000 pages. In other words, a *single* recent version of Linux is the equivalent of a *218 volume encyclopedia*.¹ Now consider that there are *597 distinct versions*, and think of each version as an edition of the encyclopedia. Hence in the absence of a specification of version, file and line information for the allegedly misused code, SCO is essentially saying “somewhere in the *597 distinct editions* of this *multi-(in many cases 100+) volume encyclopedia* you have misused some of our property.”

11. Without a specification of the System V code that has allegedly been copied, IBM cannot know with specificity even what IBM stands accused of misusing. Without a specification of where the accused code allegedly appears in Linux and AIX or Dynix, IBM faces a pragmatically impossible task of finding it. For the record, there are:

¹ Assuming 500 pages to a volume. Even the earliest and smallest version of Linux contains over 175,000 lines of text, the equivalent of over 3000 pages.

- At least 11 versions of System V code totaling almost 24 million lines of text²;
- At least 9 versions of AIX totaling almost 1.2 billion lines;
- At least 25 versions of Dynix totaling almost 157 millions lines; and
- At least 597 versions of Linux totaling almost 1.4 billion lines.

12. In the case of the 198 challenged items, SCO thus has offered an impossibly non-specific accusation, attempting to leave both the interpretation of the allegations and the finding of the evidence (should there be any) as an exercise for IBM. Requesting version, file, and line numbers for all the code in question is no more unreasonable on the face of it than an encyclopedia publisher asking that an allegation of plagiarism be specified in terms of the edition, volume and page where the accused text appears, as well as a listing of the text from which it was allegedly copied.

13. In the absence of such information, allegations are impossible even to analyze: imagine the publisher of the *Encyclopedia Americana* telling the publisher of the *Encyclopedia Britannica* “your encyclopedia contains material that was copied from us” and then refusing to specify what was copied (what text from which edition, volume and page of the *Americana*) or where it appears (which edition, volume and page of the *Britannica*).

III. THE REASON FOR MR. ROCHKIND’S DISAGREEMENT

14. Mr. Rochkind disagrees with my conclusion that the 198 Items are not disclosed with the requisite specificity. The primary reason for this disagreement is that

² Each complete version of an operating system is typically given a distinct “release number,” as, for example, version 2.6.9 of Linux. The version counts given above list the number of distinct versions shown in the Declaration of Todd Shaughnessy, dated April 4, 2006; the total lines of text cited report all text contained in both the complete versions and any additional “patches” (i.e., incremental changes), as listed in the Shaughnessy Declaration.

he used a very different test to evaluate specificity than I did. When the test that I understand to be the correct test is applied, the 198 Items come nowhere close to passing muster.

15. I was instructed by counsel for IBM to evaluate the 198 Items based on the language of the Court's orders of December 12, 2003, March 3, 2004, and July 1, 2005. As described in Exhibit A, I understand the orders to require the disclosure of the allegedly misused material by version, file and line of code. That is the standard (and most precise) means of identifying the code, methods and concepts, and concepts of an operating system.

16. The Court's Order of December 12, 2003, states that SCO is required:

(1) "To identify and state with specificity the source code(s) that SCO is claiming form the basis of their action against IBM." (¶ 4.)

(2) "To respond fully and in detail to Interrogatory Nos. 1-9 as stated in IBM's First Set of Interrogatories" (¶ 1), which provide, for example, as follows:

Interrogatory 1: "Please identify, with specificity (by product, file and line of code, where appropriate) all of the alleged trade secrets and any confidential or proprietary information that plaintiff alleges or contends IBM misappropriated or misused,"

Interrogatory 3: "Please . . . describe, in detail, . . . all places or locations where the alleged trade secret or confidential information may be found or accessed."

Interrogatory 4: "Please describe, in detail, . . . with respect to any code or method plaintiff alleges or contends that IBM misappropriated or misused, the location of each portion of such code or method in any product, such as AIX, in Linux, in open source, or in the public domain."

(3) "To respond fully and in detail to Interrogatory Nos. 12 and 13 as stated in IBM's Second Set of Interrogatories" (¶ 2), which provide, for example, as follows:

Interrogatory 12: "Please identify, with specificity (by file and line of code), (a) all source code and other material in Linux . . . to

which plaintiff has rights; and . . . how the code or other material derives from UNIX.”

Interrogatory 13: “[P]lease . . . describe in detail how IBM is alleged to have infringed plaintiff’s rights. . . .”

17. The Court’s Order of March 3, 2004, required SCO to: (1) “provide and identify *all specific lines of code* that IBM is alleged to have contributed to Linux from either AIX or Dynix” (§ 2), (2) “provide and identify *all specific lines of code from Unix System V* from which IBM’s contributions from AIX or Dynix are alleged to be derived” (§ 3), and (3) “provide and identify *with specificity all lines of code in Linux* that it claims rights to” (§ 4, emphasis added). It is difficult to imagine instructions that are any clearer, more specific, or more unambiguous.

18. The Court’s Order of July 1, 2005 (at 4) reiterated SCO’s obligations to specify its claims and ordered it to update its interrogatories accordingly.

19. Note that the Court’s orders required no more of SCO than SCO required of IBM. In its First Request for Production of Documents, SCO defined the term “identify” as follows:

“DEFINITIONS AND INSTRUCTIONS . . .

The term “identify” shall mean: . . .

e. in the case of alleged trade secrets or confidential or proprietary information, whether computer code, methods, or otherwise, to give a complete and detailed description of such trade secrets or confidential or proprietary information, including but not limited to an identification of the specific lines and portions of code claimed as trade secrets or confidential or proprietary information, and the location (by module name, file name, sequence number or otherwise) of those lines of code within any larger software product or property.” (Exhibit B (emphasis added).)

I understand that SCO subsequently incorporated this identical definition in eight additional document requests, five additional sets of interrogatories, seven 30(b)(6)

deposition notices, and three requests for admission, the latest of which was served on March 10, 2006. Thus, SCO itself has continuously demanded the same degree of specificity ordered by the Court and requested by IBM.

20. Despite the language of the Court's orders, and of SCO's own discovery requests, the Final Disclosures do not provide version, file and line information for each of the 198 Challenged Items. As is illustrated in my original declaration (Addendum B), and summarized in the following table, SCO provides version, file and line information for very few of the Challenged Items:

	Version(s)	File(s)	Line(s)
System V	1	1	0
AIX	1	1	0
Dynix	2	3	0
Linux	27	149	3

Note that there is not even one Item for which SCO provides a *complete* set of coordinates.

21. Mr. Rochkind does not seem to disagree that SCO has not provided a complete set of coordinates for each of the 198 Items. Instead, he asserts that, with respect to many of the Items, SCO has provided sufficient detail relating to claims because it has summarized its allegations of misuse, provided documents relating to the alleged misuse, identified persons involved in the alleged misuse and/or pointed IBM to source code. (Rochkind Decl. ¶ 9.)

22. It is true that SCO has, for most of the Items, summarized its allegations, listed persons involved in the alleged misuse and referred IBM to certain documents. That is simply not the appropriate measure of compliance, as I understand the Court's orders. Nor would it be the appropriate measure of compliance under SCO's own discovery requests. Mr. Rochkind's declaration defines his own standard of specificity and asserts that SCO's Final Disclosures, of which Mr. Rochkind claims to be the primary author, meet the standard.

23. Putting aside the language of the Court's orders, it is difficult to consider the information SCO has provided as sufficiently specific when (1) many of the summaries are extremely general (*e.g.*, Item 180 claims only that IBM misused the "internals" of System V Release 4, without any mention of which part of the several-million-line operating system was misused); (2) the documents provided are mostly documents that IBM provided to SCO, and they tell IBM little more than it would have known before SCO filed its complaint; (3) for many of the Items, SCO does not identify any individuals, it says only "IBM"; and (4) according to Exhibit B to Mr. Rochkind's declaration, SCO identified code with respect to no more than 16 of the 198 Items.

24. If the Court's orders required only that SCO provide some minimal, additional information about its allegations, then I agree with Mr. Rochkind that it has done that. If they required that SCO provide the standard coordinates for identifying allegedly misused aspects of an operating system (code, methods and concepts), then SCO's disclosures regarding the 198 Items fall far short. For some of the Items (*e.g.*, Item 93), the Final Disclosures reveal little more than the minimal description found in SCO's Complaint.

IV. MR. ROCHKIND'S ASSERTIONS ABOUT METHODS AND CONCEPTS

25. Rather than disagree with the fact that SCO has not provided version, file and line information regarding any of the 198 Items, Mr. Rochkind devotes the better part of his declaration to rationalizing SCO's decision *not* to provide the information. Contrary to Mr. Rochkind's suggestion, however, there is no reason SCO could not have provided the missing information with respect to its methods and concepts-misuse allegations, as well as its code-misuse allegations.

26. To begin, Mr. Rochkind seems to suggest that virtually all of the 198 Items concern methods and concepts rather than source code. (Rochkind Decl. ¶¶ 8-9.) According to Mr. Rochkind, less information is required to evaluate a method than is required to evaluate code. (Rochkind Decl. ¶ 10.) Thus, Mr. Rochkind states, there was no need for SCO to identify version, file and line information relating to methods and concepts. (Rochkind Decl. ¶ 10.) Putting aside the fact that the Court's orders -- on their face -- require version, file and line information for methods and concepts as well as code, Mr. Rochkind is mistaken both as to the number of Items that concern methods and concepts and the information needed fully to evaluate operating-system methods and concepts.

27. Contrary to Mr. Rochkind's suggestion, a significant portion of the 198 Items concern the alleged misuse of code. As described in Exhibit C, the language of many of the challenged Items themselves relate to the alleged misuse of code. For example:

Item 17: "Port of discontinuous memory code from ptx to Linux 2.5".

Item 22: "Port of ptx NUMA code to Linux".

Item 27: "Transferring ptx source code to AIX developers".

Thus, it is simply wrong for SCO's Mr. Rochkind to imply that the only Items in dispute concern methods and concepts.

28. Mr. Rochkind suggests that all Items of allegedly misused code are disclosed by SCO with appropriate line specificity. That is unfortunately patently false and Mr. Rochkind is ignoring dozens of the 198 challenged Items that do concern alleged code misuse. In fact, many of the Items that clearly concern the alleged misuse of code comprise SCO's most *imprecise* allegations. In 39 of the Items (Items 232 to 270), for example, SCO accuses IBM of making improper reference to Dynix source code as a basis for writing additional code, while providing essentially no further information. Each of these 39 items has an "Improper Disclosure" claim of the form: "Use of ptx [i.e., Dynix] programming experience (and a fortiori exposure to related aspects of Unix System V) in programming [or 'implementing'] _____," where the blank contains things such as "MP preemption and synchronization code", "i686 large-memory SMP systems", "code for SCSI Mid-layer Multi-Path IO", and so forth. That is, SCO is specifically accusing IBM of referring to Dynix *code* and System V *code*, and then using that as the basis for creating additional *code* (e.g., "MP preemption and synchronization *code*"). Yet there is absolutely no specification of any kind (no version, file, or line numbers) of which Unix code was allegedly referenced, or of which Dynix code was allegedly referenced. IBM is left to guess as to which of the 470,000-plus files and 156 million-plus lines of Dynix code included within SCO's vague claims are in fact challenged by these Items.

29. As if to further justify SCO's failure to provide version, file and line information, Mr. Rochkind suggests that it is not possible to identify version, file and line coordinates with respect to methods and concepts. (Rochkind Decl. ¶ 10.) That is simply

incorrect. The methods and concepts employed in an operating system (or any computer program) *are in the source code*. It could not be otherwise: The source code of a program specifies all of its possible behavior. If that behavior truly embodies a method, that method must be expressed in specific lines of the source code; there is just no other way to do it.

30. Consider, for example, Item 146, which alleges (among other things) that IBM improperly disclosed a method called “differential profiling”. Simply put, the method suggests ways of finding performance bottlenecks by counting the events that happen inside a program and then analyzing those counts. But the counting and analyzing can be done only by *code*, i.e., source code written to keep track of the number of times an event happens and written to analyze the counts as explained in the method. Any time a method is used, it can only be because there is source code that implements it. It really is that simple. Hence, if System V, AIX, Dynix or Linux used that method, they must contain source code that implements it, and SCO ought to cite the specific lines of code.

31. Although, as Mr. Rochkind states (Rochkind Decl. ¶ 10), methods and concepts are sometimes discussed in text books without reference to source code, such discussions are, most often, at a high level of generality. The mere fact that a method can be discussed generally without referring to source code does not mean that its corresponding source code cannot be identified. It can, and SCO--having alleged that System V, AIX, Dynix or Linux code somewhere embody a method--bears the responsibility of identifying the specific code it claims embodies that method.

32. The disclosure of the corresponding source code also greatly aids in understanding the method, as Mr. Rochkind’s own text illustrates. Despite his attempt to make the identification of source code seem irrelevant to the identification of methods and

concepts (Rochkind Decl. ¶ 10), his own book on the subject of operating systems, Advanced Unix Programming (2d ed. 2004), (which he asserts, “explain[s] in detail how to use UNIX system calls” (¶ 5)) devotes considerable space to describing methods and concepts with reference to source code. For example, it states that “this new book includes thousands of lines of example code”. (xii). Indeed, his chapter devoted to “Fundamental Concepts” describes UNIX concepts using, in many cases, nearly full-page excerpts of source code and even refers back to his own website to offer complete code listings where the excerpts are not enough. (*See, e.g., id.* at 24-38. (Exhibit D).)

33. SCO’s Chief Technology Officer, Sandeep Gupta, testified concerning the importance of having version, file and line information with respect to methods and concepts. Mr. Gupta was asked the following questions and provided the following answers:

Q. “Okay. How would you determine whether a particular description was specific enough to describe an aspect of System V as a method?”

A. “I have to look at the source code.”

Q. “Okay. What would you do if you looked at the source code?”

A. “I look at various steps that are taken, specific for that particular method.”

Q. “Okay. So in order to determine what a particular method or concept is, you would actually have to look at the source code?”

A. “In some cases, yes.”

Q. “Okay. I mean, I -- I understand you just articulated a few from memory and --

A. “Yeah.”

Q. “-- I’m impressed with that, but in general, would you have to look at the source code to be able to accurately describe a method or concept in UNIX?”

A. “That’s my opinion, yes.” (Exhibit E (03/17/2006 Gupta Dep. Tr. at 266-67) (objections omitted).)

34. Furthermore, as stated, SCO itself specifically demanded that IBM identify methods and concepts with reference to files and lines of code. It did that, no doubt, because the standard means of identifying an operating system method with specificity is by file and line of code. I assume SCO would not have demanded that IBM provide information that could not be provided.

35. In truth, it is even more important to have version, file and line information regarding methods and concepts claims than it is to have the information for code claims. When specific lines of source code are identified by a plaintiff who alleges they have been improperly copied, a defendant can at least automate the process of looking for literal infringement: he can set a computer to work searching through his own code to see if it contains the lines identified by the plaintiff.

36. But the same cannot be said for methods and concepts. Consider once again the alleged “Improper Disclosure” in Item 146: “The idea is to compare corresponding buckets of the profile data to determine which portion of the code is most responsible for the slowdown.” There are no automated techniques for finding the lines of code that embody that method. Because they are abstractions, methods and concepts must instead be located by manual review of the code, and given that there are between tens of millions (System V) and billions (AIX, Linux) of lines to be searched, locating such methods and concepts are simply untenable here. Given the size of the code base here, manual review is, as a practical matter, an impossible task.³ Hence, without a specification by SCO of

³ Returning to our analogy of the two encyclopedias, imagine that the *Americana* accused the *Britannica* of copying a specific passage of *Americana*'s text. *Britannica*

the location of the code implementing the method, the claim cannot be adequately analyzed.

V. DEFENDING AGAINST SCO'S CLAIMS

37. I stated in my original declaration that SCO's failure to provide version, file and line information makes it impossible, practically speaking, for IBM to defend itself. Mr. Rochkind disagrees. (Rochkind Decl. ¶ 7.) However, his view is supported only by naked assertions and does not survive even the weakest scrutiny.

38. The kinds of questions that must be asked to defend against SCO's allegations are not a secret. They have been involved (more or less) in each of the 30-plus cases in which I have been retained as an expert to deal with alleged misappropriation of intellectual property, including in *Computer Associates v. Altai*, in which I served as an expert appointed by and for the Court.

39. Among the many questions IBM must answer are the following:

- Did IBM offer the Item to Linux?
- Did the Item originate in or derive from System V and AIX or Dynix?
- Was the Item accepted into Linux and, if so, when and to what effect?
- Is the Item copyrightable (or is it unoriginal; a mere idea, process or procedure; dictated by externalities; or in the public domain)?
- Has the disclosure of the Item or its inclusion in Linux had a negative effect on SCO or a positive impact on IBM?

could do an automated search for that text. But imagine instead if *Americana* accused *Britannica* of using what *Americana* claimed to be its proprietary "non-Eurocentric method of describing history" (*i.e.*, ensuring a more global, inclusive world view), and then refused to give any information about which edition(s), volume(s) or page(s) in *Britannica* had done that. Consider the nature and difficulty of the task *Britannica* would face in trying to find places that had used that method.

I do not understand Mr. Rochkind to dispute the relevance of these questions, which must be answered on a line-by-line basis.

40. Many thousands of persons have contributed to the development of Linux, and IBM has made many contributions to Linux, some of which represent only a few lines of code in a file comprised of hundreds of lines of code. The only way to know whether IBM made a given contribution is to know precisely what the alleged contribution is. Similarly, whether a given contribution originated in, or is derived from, System V, AIX or Dynix is a line-specific inquiry. One line may have; another may not have. Version, file and line information is no less critical to determining whether a line of code--and especially a method--is in Linux, since it is composed of millions of lines of codes and many thousands of methods and concepts and concepts.

41. To determine whether an Item is copyrightable requires line information because that is the only way to assess originality, determine whether the line is merely an idea, process or procedure, evaluate whether the Item is dictated by programming practices, governed by standards, or in the public domain. These questions simply defy generalized examination. In a given file, one line might be original, whereas another might not; one might be in the public domain, whereas another might not; and so on. For these same reasons, it is also not possible to evaluate whether a method has a positive impact on Linux and IBM (or a detrimental impact on SCO) without understanding precisely what it is.

42. Absent the production of the version, file and line information referenced in the Court's orders, it is very difficult, if not impossible, to answer these questions. As described in the Declaration of Todd Shaughnessy, dated April 4, 2006, the size of the code bases implicated by SCO's claims is enormous.

Operating System	Version(s)	File(s)	Line(s)
System V	11	112,622	23,802,817
AIX	9	1,079,986	1,216,698,259
Dynix	37	472,176	156,757,842
Linux	597	3,485,859	1,394,381,543
Total:	654	5,150,643	2,791,640,461

43. Mr. Rochkind does not disagree that the implicated code base is enormous. Nor does he disagree that SCO's Chris Sontag provided sworn testimony early in the case that it would take 25,000 person-years to review a code data base .2% the size of the stack of code at issue. Mr. Rochkind states only that he has helped SCO to provide enough information for IBM to find the 198 needles in the haystack. Having helped to decide what the needles are, Mr. Rochkind may well feel as if he knows what they are. But I do not. Nor do I believe that other independent experts would.

44. Examining the only Item specifically mentioned in Mr. Rochkind's declaration, Item 146, makes the point. In Item 146, SCO complains about IBM's "Use of Dynix/ptx for Linux development" by reference to: (1) an email asking for help with a performance problem, (2) an email response with a suggested analysis technique (differential profiling), (3) a technical paper written by Paul McKenney, (4) a URL reference to scripts that might be of help, and (5) a list of 11 Linux files (names only, no versions or lines). Contrary to Mr. Rochkind's claim that I ignored these materials, in fact it was by examining them closely that I concluded, as stated in my original declaration, that SCO has provided no meaningful information about what IBM is alleged to have done wrong.

45. The claim in Item 146 is sufficiently vague as to lead to several different interpretations. As it takes several pages to analyze all of the possible interpretations, and to point out all of Mr. Rochkind's errors, I have put the analysis in Exhibit F. The details can be found there; the summary points are simple enough:

- Mr. Rochkind points out that the email cited in Item 146 contains “an actual Linux patch” and “a specification for the files and lines being patched”. He conveniently overlooks the fact that the patch, and the files and lines being patched, are all the “before” version of the code. That is, the patch and associated files contain the code that *didn't work* well enough, the code that the application of differential profiling was supposed to help repair. There is in fact no code cited that is alleged actually to contain the use of the method.
- Mr. Rochkind points out that there is “an exact URL reference to a 9-page technical paper by McKenney explaining the method and concept at issue.” Indeed there is, and the paper was published in the open literature in 1995 (the 1995 IEEE MASCOTS Symposium), six years prior to the email in question.
- As Mr. Rochkind points out, there are “11 Linux file paths” specified in Item 146, but, as he omits to mention, no specific version of Linux is cited. Even so, unfortunately for his position, none of these files appears to deal with differential profiling.⁴

⁴ Item 146 indicates yet another level of difficulty in deciphering SCO's claims: even where SCO *does* specify file names (but still not versions or line numbers), IBM *still* has to guess what SCO is talking about: 4 of the 11 Linux filenames in SCO's Item 146 are simply incorrect: there is no Linux file named arch/i386/oprofile/rmi_int.c, arch/i386/oprofile/rmi_int.c, arch/i386/oprofile/op_counter.c, or arch/i386/oprofile/op_x86_model.c. There are files whose names are close to these, and are likely what was intended, but this presents yet another step IBM must take to determine what SCO actually means.

46. As described in Exhibit F, rather than clearly state its claims to Item 146, SCO identifies a series of dots and leaves IBM to try to connect them. The problem is they do not connect. At most they leave IBM to guess as to which of any number of claims SCO might actually be making. To defend itself, IBM is left to respond not just to what is at issue -- which is not clear -- but to all of the possibilities. For Items like Item 146, there are at least a handful of possibilities. As to other items, the possibilities are almost innumerable. When SCO accuses IBM of misusing the internals of System V (*e.g.*, Item 180) or of misusing its experience with Dynix/ptx, for example, SCO accuses IBM of misusing any one of the millions of lines of code and the thousands of methods and concepts contained in these operating systems.

47. Even if IBM could feasibly chase all of the possibilities held open by the Final Disclosures (which clearly it could not do without years of additional effort), the generality, uncertainty and ambiguity inherent in the final disclosures are sure to lead to games of "where's the pea" during the expert and summary judgment phases of the case. Based on the information SCO has provided (or, more accurately, not provided), it is difficult to imagine any meaningful exchange of views among experts. Likewise, a court can hardly evaluate at summary judgment what cannot be defined. Had SCO provided full code coordinates for the allegedly misused material, games of "where's the pea" would not be possible. SCO's claims could have been understood and analyzed. Unintentional allegations could have been eliminated.

48. While I do not believe that IBM can fairly defend itself absent version, file and line information for each of the Items, it would -- at the risk of stating the obvious -- require a very significant period of time for IBM to conduct an investigation into the general allegations set out in the 198 Items. Without engaging a very large corps of

experts, it would take years. Even then it is very unlikely that IBM could succeed in learning what is ultimately known only to SCO: its allegations.

VI. MR. ROCHKIND'S WILLFULNESS ASSERTIONS

49. Finally, Mr. Rochkind addresses IBM's contention that SCO acted willfully in failing to provide version, file and line information. (Rochkind Decl. ¶¶ 16-18.)

Mr. Rochkind claims that IBM is wrong to state that SCO acted "willfully in not specifying its claims" and wrong that "SCO has declined, as a practical matter, to tell IBM what is in dispute." (Rochkind Decl. ¶ 16.) Here again, Mr. Rochkind's view appears to turn on his own, self-defined view of the appropriate standard.

50. I am not a legal expert, and do not pretend to be an authority on the meaning of the term "willfully" for purposes of a court's deciding whether a party should be limited in submitting evidence in support of its claims. In responding to Mr. Rochkind's assertion, however, I rely on the definition of the term used in the cases provided by counsel for IBM, *e.g.*, *Schroeder v. Southwest Airlines*, 129 Fed. Appx. 481, 484-85, 2005 WL 984495 (10th Cir. 2005) (holding that "[w]illful failure means 'any intentional failure as distinguished from involuntary noncompliance. No wrongful intent need be shown'"); *F.D.I.C. v. Daily*, 956 F.2d 277, 1992 WL 43488, at *3-6 (10th Cir. 1992) (same); and *In re Standard Metals Corp.*, 817 F.2d 625, 628-29 (10th Cir. 1987) (same).

51. Using the definition of "willfully" set out in these cases, I have no difficulty concluding that SCO acted "willfully" in submitting its Final Disclosures and omitting the information called for in the Court's orders. The Court's orders clearly call for version, file and line information, with respect both to code and methods and concepts. Identifying code and methods and concepts by version, file and line of code is the standard method of identifying operating system source code and methods and concepts

with specificity. SCO asked nothing less than this of IBM. There is no reason it could not be provided here. Indeed, without it, the 198 Items are too vague and indefinite to permit complete analysis.

52. As Mr. Rochkind's declaration makes clear, SCO does not claim to have assembled the Final Disclosures unwittingly. It plainly did not, as evidenced by the fact that SCO provides version, file and line information for a number of Items that are not challenged in this motion. There is no dispute that SCO made a deliberate decision to provide the information it provided and the information it did not. (Rochkind Decl. ¶ 10.) And SCO deliberately created a different standard to apply to itself than it demanded of IBM, and the court required. SCO's failure to provide version, file and line information was not unknowing or inadvertent.

53. Moreover, the information omitted from SCO's disclosures is unquestionably within SCO's control. (Rochkind Decl. ¶ 14 n.3.) The Court's orders, as I understand them, direct SCO (in substantial part) to make its allegations specific. For example, to the extent SCO claims that IBM improperly used Dynix code and methods and concepts in contributing to Linux (and the vast majority of SCO's allegations are of this type), the orders (on their face) require SCO to "describe, in detail, . . . with respect to any code or method plaintiff alleges or contends that IBM misappropriated or misused, the location of each portion of code or method in any product." Only SCO knows what it alleges. No amount of investigation by IBM can connect the dots. Yet SCO systematically omitted this information from the 198 Items as described in Addendum B to my initial declaration.

54. In sum, Mr. Rochkind's claim that SCO did not willfully withhold information in its possession with respect to version, file and line of code misses the point. As has been demonstrated, it is possible to obtain version, file and line information

with respect to methods and concepts if an effort to do so is undertaken. SCO, simply put, has willfully failed to undertake any such effort.

VII. CONCLUSION

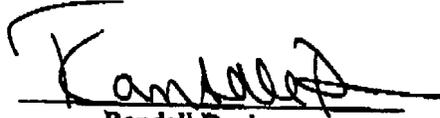
55. Upon careful review of Mr. Rochkind's declaration, I find that he fails even to address the central assertions in my opening declaration. He does not -- and could not -- dispute that SCO has not provided version, file and line information regarding each of the 198 Items at issue in IBM's motion

56. There is no reason SCO could not have provided this information, including with respect to methods and concepts, for which it is even more necessary, not less necessary. SCO's own discovery demands make the point.

57. Without the missing information, IBM lacks precisely the kind of information needed to conduct a basic inquiry relating to the facts of SCO's claims. Given enough time, IBM might be able to discover some of the information SCO has failed to provide. It will never be able to find all of the information, however, because only SCO knows its allegations.

58. It is for this reason, in significant part, that I have no difficulty disagreeing with Mr. Rochkind's statements regarding willfulness. To my knowledge, SCO has never argued (and could not credibly argue) that SCO's failure to provide version, file and line information was an oversight.

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


Randall Davis

Date: 28 April 06

Place: Cambridge, MA

EXHIBIT A

System V

AIX

Dynix

Linux

12/12/03
Order

- “To identify and state with specificity the source code(s) that SCO is claiming form the basis of their action against IBM.” (¶ 4)
- “To respond fully and in detail to Interrogatory Nos. 1-9 as stated in IBM’s First Set of Interrogatories.” (¶ 1)

Interrogatory 1: “Please identify, with specificity (by product, file and line of code, where appropriate) all of the alleged trade secrets and any confidential or proprietary information that plaintiff alleges or contends IBM misappropriated or misused,”

Interrogatory 3: “[P]lease . . . describe, in detail, . . . all places or locations where the alleged trade secret or confidential or proprietary information may be found or accessed.”

Interrogatory 4: [P]lease describe, in detail, . . .with respect to any code or method plaintiff alleges or contends that IBM misappropriated or misused, the location of each portion of such code or method in any product, such as AIX, in Linux, in open source, or in the public domain.”

- “To respond fully and in detail to Interrogatory Nos. 12 and 13 as stated in IBM’s Second Set of Interrogatories.” (¶ 2)

Interrogatory 12: “Please identify, with specificity (by file and line of code), (a) all source code and other material in Linux . . . to which plaintiff has rights; and . . . how the code or other material derives from UNIX.”

Interrogatory 13: “[P]lease . . . describe in detail how IBM is alleged to have infringed plaintiff’s rights”

3/3/04
Order

- **“SCO is to provide and identify all specific lines of code from Unix System V from which IBM’s contributions from AIX or Dynix are alleged to be derived.”** (¶ 3)
- **“As previously ordered, SCO is to provide and identify all specific lines of code that IBM is alleged to have contributed to Linux from either AIX or Dynix.”** (¶ 2)
- **“SCO is to provide and identify with specificity all lines of code in Linux that it claims rights to.”** (¶ 4)

7/1/05
Order

- The court sets an “Interim Deadline for Parties to Disclose with Specificity All Allegedly Misused Material Identified to Date and to Update Interrogatory Responses Accordingly” and a “Final Deadline for Parties to Identify with Specificity All Allegedly Misused Material.”

EXHIBIT B

Defendant is directed to give answers to the written interrogatories separately, fully, in writing, under oath, and in accordance with the following definitions and instructions. Defendant is requested to produce the documents and things in its possession, custody or control pursuant to the document requests.

Answers to the interrogatories and all documents and things responsive to the document requests must be served on the undersigned attorneys for The SCO Group at the offices of Boies, Schiller & Flexner LLP, 100 Southeast Second Street, Suite 2800, Miami, Florida 33131 within 30 days of service of these interrogatories and document requests.

DEFINITIONS AND INSTRUCTIONS

For purposes of these interrogatories and requests for production of documents, the following definitions and instructions apply.

A. Definitions.

1. The term "AIX" shall mean the UNIX-based operating system distributed and/or developed by IBM, including all prior versions, releases and maintenance modifications.
2. The term "concerning" shall mean relating to, referring to, reflecting, describing, evidencing, referencing, discussing or constituting.
3. The term "describe" shall mean in the case of an event or circumstance, to set forth in detail the date, time, place, individuals or entities involved and context and content of the event or circumstance.
4. The term "document" shall be deemed to include every record of every type including, without limitation, information stored on any electromagnetic storage device, or computer, any written, printed, typed, recorded, stored, or graphic matter, however

produced, reproduced, or existing in the possession, custody, or control of Defendant, or any agent, employee, or attorney of the Defendant, and all drafts, notes, or preparatory material concerned with said document, and every additional copy of such record or document where such copy contains any commentary, notation, or other change whatsoever that does not appear on the original or other copy of the document produced. "Document" shall be deemed also to include any summary of a document or documents called for hereafter.

5. The term "Dyrix" shall mean the UNIX-based operating system distributed and/or developed by Sequent Computer Systems, Inc. and/or IBM, including all prior versions, releases, derivative works, methods, and modifications.
6. The terms "IBM," "Defendant" "you," "your," and any synonym thereof and derivatives therefrom are intended to and shall embrace IBM and include its parents, subsidiaries, divisions, or affiliates and any corporate predecessor or successor of any of them, including Sequent Computer Systems, Inc., and, in addition all of the Defendant's attorneys and accountants, and all of its respective agents, servants, associates, employees, representatives, investigators, officers, directors and others who are or have been in possession of or may have obtained information for or on behalf of such Defendant in any manner with respect to any matter referred to in the pleadings in the above-styled case.
7. The term "identify" shall mean:
 - a. in the case of a natural person, to state the full name, current or last known job title and position, current or last known address, current or last known home and work telephone numbers, and current or last

known electronic mail address, and to indicate the basis of that person's knowledge, including but not limited to the identification of documents and communications and a description of his/her personal involvement in any transaction, meeting, software development, marketing, or other activity relating in any way to the allegations of the Complaint and any defenses;

- b. in the case of any entity other than a natural person, to state its name, address, principal place of business and, if applicable, place of incorporation and a contact person at the entity;
- c. in the case of a document, to state the author(s), title, subject matter, date, place, source of publication of the document and substance of the document;
- d. in the case of an oral communication, to give a complete description of such oral communication, including but not limited to: (i) the speaker(s) and actual or intended recipient(s) or witnesses of the communication; (ii) the date of the communication; and (iii) the substance of the communication;
- e. in the case of alleged trade secrets or confidential or proprietary information, whether computer code, methods or otherwise, to give a complete and detailed description of such trade secrets or confidential or proprietary information, including but not limited to an identification of the specific lines and portions of code claimed as trade secrets or confidential or proprietary information, and the location (by module

name, file name, sequence number or otherwise) of those lines of code within any larger software product or property.

- f. in the case of an alleged right, to give a completed and detailed description of such right, including but not limited to: (i) the issuer of the right; (ii) the date the right became effective; (iii) the date the right expired; and (iv) any limitations placed upon such right.
8. The term "open source" shall mean any software code that is made available in source code form without any confidentiality restrictions, including but not limited to any code made available under the General Public License, the BSD license, or the MIT license.
9. The term "person" shall be deemed to include natural persons, partnerships, firms, and corporations, and all of their subsidiaries or divisions, and, in the case of partnerships, firms, and corporations, the individual member(s) or agent(s) thereof.
10. The term "source code" shall mean the human-readable form of a computer program written in the original and preferred form for human inspection and modification, and includes but is not limited to source code listings; compiler and/or assembler output listings for such source code; source code listings for macros or "includes" (both executable and mapping) listings used in such source code; job control language files; and/or other files required to create an executable version of a program, including but not limited to user interface components; panels; screen definitions and help text; and e-lists.

B. Instructions.

1. Unless otherwise indicated, all requests and interrogatories are from January 1, 1999 to present.
2. Information requested in these interrogatories shall include information within the knowledge or possession of any of Defendant's agents, employees, attorneys, investigators or any other persons, firms or entities directly or indirectly subject to Defendant's control in any way whatsoever.
3. Each interrogatory shall be answered in its entirety. If any interrogatory or subsection thereof cannot be answered in full, it shall be answered to the fullest extent possible with an explanation as to why a complete answer is not provided.
4. If there is a claim of privilege as to any communication concerning information requested by these interrogatories, specify the privilege claimed, the communication and/or answer to which that claim is made, the topic discussed in the communication and/or answer to which that claim is made, the topic discussed in the communication and the basis upon which the claim is asserted.
5. These interrogatories are continuing in nature and require supplemental or additional responses in accordance with Rule 33 of the Federal Rules of Civil Procedure.
6. All documents produced in response to these requests shall be produced in the same order as they are kept or maintained in the ordinary course of business and, where multiple pages or documents are assembled, collated, grouped, or otherwise attached, shall not be separated or disassembled.

7. With respect to any document responsive to this request that is withheld from production based upon a claim of privilege, please provide the information required pursuant to Rule 26(b)(5) of the Federal Rules of Civil Procedure.
8. If, for reasons other than a claim of privilege, you refuse to produce any document requested herein, state the grounds upon which the refusal is based with sufficient specificity to permit a determination of the propriety of such refusal.
9. If there are no documents responsive to any paragraph or subparagraph set forth in the requests, please provide a written response so stating.

These requests are continuing and, pursuant to Rule 26(e) of the Federal Rules of Civil Procedure, require further and supplemental production by Defendant whenever Defendant acquires, makes, or locates additional documents between the time of the initial production hereunder and the time of the trial in this action.

REQUESTED DOCUMENTS

1. All documents concerning or relating to any agreements entered into with AT&T relating to UNIX, including but not limited to the agreements attached to the First Amended Complaint.
2. All versions or iterations of AIX source code, modifications, methods and/or derivative works from May 1999 to the present, including but not limited to version 4.3 and above.
3. All versions or iterations of Sequent Dynix source code, derivative works, modifications and/or methods from January 1, 1999 to the present.

4. All documents concerning IBM's efforts, if any, to maintain the confidentiality of UNIX source code, derivative works, modifications, and/or methods.
5. All documents concerning IBM's efforts, if any, to maintain the confidentiality of AIX source code, derivative works, modifications, and/or methods.
6. All documents concerning IBM's efforts, if any, to maintain the confidentiality of Sequent Dynix source code, derivative works, modifications, and/or methods.
7. All documents concerning IBM's efforts, if any, to restrict distribution of Unix source code, derivative works, modifications, and/or methods.
8. All documents concerning IBM's efforts, if any, to restrict distribution of AIX source code, derivative works, modifications, and/or methods.
9. All documents concerning IBM's efforts, if any, to restrict distribution of Sequent Dynix source code, derivative works, modifications, and/or methods.
10. All documents concerning Prerequisite Source Licenses, including but not limited to all instances in which IBM required persons or entities to obtain a Prerequisite Source License under paragraph 2.2(a) of its contract with its customers.
11. All contributions made without confidentiality restrictions by IBM or anyone under its control including, but not limited to, source code, binary code, derivative works, methods, and modifications to Open Source Development Lab, Linus Torvalds, Red Hat or any other entity.
12. All documents that identify any person or entity to whom IBM has provided UNIX source code, derivative works, modifications and/or methods.
13. All documents that identify any person or entity to whom IBM has provided AIX source code, derivative works, modifications and/or methods.

14. All documents that identify any person or entity to whom IBM has provided Sequent Dynix source code, derivative works, modifications and/or methods.
15. All documents that identify any person at IBM and Sequent who had access to UNIX source code, derivative works, modifications and/or methods.
16. All documents that identify any person at IBM and Sequent who had access to AIX source code, derivative works, modifications and/or methods.
17. All documents that identify any person at IBM and Sequent who had access to Sequent Dynix source code, derivative works, modifications and/or methods.
18. All documents, agreements and correspondence between IBM or any person or entity under IBM's control and Linus Torvalds including, but not limited to, those with or copied to Sam Palmisano.
19. All documents, agreements and correspondence with Open Source Development Lab.
20. All documents, agreements and correspondence with Red Hat.
21. All documents, agreements and correspondence with SuSe.
22. All documents, agreements and correspondence between IBM and Novell regarding UNIX, including but not limited to all correspondence with Jack Messman, Chris Stone and/or Novell's counsel.
23. All documents, agreements and correspondence between IBM and Santa Cruz Operation regarding UNIX.
24. All documents, agreements and correspondence between IBM and Caldera.
25. All documents, agreements and correspondence between IBM and The SCO Group.
26. All documents identifying any IBM personnel who are or were employed or working at the Linux Technology Center.

27. All documents identifying any IBM personnel who are or were employed or working at the Linux Center of Competency.
28. All documents concerning Project Monterey.
29. All documents concerning any UNIX source code, derivative works, modifications or methods disclosed by IBM to any third party or to the public.
30. All documents concerning any AIX source code, derivative works, modifications or methods disclosed by IBM to any third party or to the public.
31. All documents concerning any Sequent Dynix source code, derivative works, modifications or methods disclosed by IBM to any third party or to the public.
32. All documents concerning any UNIX source code, derivative works, modifications or methods found in Linux, open source, or the public domain:
33. All documents concerning any AIX source code, derivative works, modifications or methods found in Linux, open source, or the public domain.
34. All documents concerning any Sequent Dynix source code, derivative works, modifications or methods found in Linux, open source, or the public domain.
35. All documents concerning any contributions to Linux or to open source made by IBM and/or Sequent.
36. All documents sufficient to show IBM's organizational and personnel structure, including but not limited to organizational charts, flow charts and personnel directories.
37. All documents concerning any statement, affidavit, declaration, or opinion in IBM's possession relating to contributions by IBM to open source, including but not limited

to those statements identified in the Complaint made by Messrs. Mills, LeBlanc and Strassmeyer.

38. All documents concerning the Open Source Developer's Class, including any guidelines relating thereto.
39. All documents concerning export controls for any UNIX source code, derivative works, modifications or methods contributed to open source, including all portions of AIX, and Dynix and their derivative works, modifications, or methods.
40. All documents concerning IBM's use of Intel processors prior to January 1, 1998.
41. All documents concerning IBM's use of Intel processors after January 1, 1998.
42. All documents concerning IBM's contributions to development of the 2.4 and 2.5 Linux Kernel.
43. All documents concerning IBM's First Affirmative Defense that the Complaint fails to state a claim upon which relief can be granted.
44. All documents concerning IBM's Second Defense that Plaintiff's claims are barred because IBM has not engaged in any unlawful or unfair business practices, and IBM's conduct was privileged, performing the exercise of an absolute right, proper and/or justified.
45. All documents concerning IBM's Third Affirmative Defense that Plaintiff lacks standing to pursue its claims against IBM.
46. All documents concerning IBM's Fourth Affirmative Defense that Plaintiff's claims are barred, in whole or in part, by the applicable statutes of limitations.

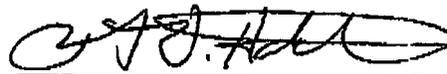
47. All documents concerning IBM's Fifth Affirmative Defense that Plaintiff's claims are barred, in whole or in part, by the economic loss doctrine or the independent duty doctrine.
48. All documents concerning IBM's Sixth Affirmative Defense that Plaintiff's claims are barred by the doctrines of laches and delay.
49. All documents concerning IBM's Seventh Affirmative Defense that Plaintiff's claims are barred by the doctrines of waiver, estoppel and unclean hands.
50. All documents concerning IBM's Eighth Affirmative Defense that Plaintiff's claims are, in whole or in part, preempted by federal law.
51. All documents concerning IBM's Ninth Affirmative Defense that Plaintiff's claims are improperly venued in this district.
52. All documents used, referred to, identified, or relied upon in responding to Plaintiff's First Set of Interrogatories.

INTERROGATORIES

1. Identify the name and address of the person(s) answering these interrogatories, and, if applicable, the persons' official position or relationship with Defendant?
2. List the names and addresses of all persons who are believed or known by you, your agents, or your attorneys to have any knowledge concerning any of the issues of this lawsuit; and specify the subject matter about which the witness has knowledge.
3. If you intend to call any expert witness at the trial of this case, state, as to each such expert witness, the name and business address of the witness, the witness' qualifications as an expert, the subject matter upon which the witness is expected to testify, the substance of the facts and opinions to which the witness is expected to testify, and a summary of the grounds for each opinion.
4. Identify all persons who have or had access to UNIX source code, AIX source code and Dynix source code, including derivative works, modifications, and methods. For each such person, set forth precisely the materials to which he or she had access.
5. Identify all IBM or Sequent personnel that work or worked on developing source code, derivative works, modifications or methods for AIX, Dynix and Linux, specifying for each person their precise contributions to each.

DATED this 24th day of June, 2003.

By:



Brent O. Hatch
HATCH, JAMES & DODGE

David Boies
Stephen N. Zack
Mark J. Heise
BOIES, SCHILLER & FLEXNER LLP
Attorneys for Plaintiff

Plaintiff, The SCO Group, hereby certifies that a true and correct copy of Plaintiff's First Request for Production of Documents and First Set of Interrogatories was served on Defendant International Business Machines Corporation on this 24th day of June, 2003, by hand delivery on their counsel of record as follows:

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

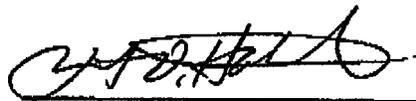
Copies by U.S. Mail to

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

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

HATCH, JAMES & DODGE

By:



Brent O. Hatch
Attorneys for Plaintiff

Snell & Wilmer

L.L.P.
LAW OFFICES

15 West South Temple, Suite 1200
Gateway Tower West
Salt Lake City, Utah 84101
(801) 257-1900
Fax: (801) 257-1800
www.swlaw.com

SALT LAKE CITY, UTAH
PHOENIX, ARIZONA
TUCSON, ARIZONA
IRVINE, CALIFORNIA
DENVER, COLORADO
LAS VEGAS, NEVADA

FACSIMILE TRANSMISSION

DATE: March 25, 2003

TIME IN:
TIME OUT:

TO:

Name	Fax Number	Phone Number
David R. Marriott Peter Ligh CRAVATH SWAINE & MOORE	(212) 474-3700	(212) 474-1000

FROM: Todd M. Shaughnessy PHONE: 801-257-1937

RE: Caldera v. IBM

MESSAGE:

David and Peter,
Attached is a copy of Plaintiff's First Request for Production of Documents and First Set of Interrogatories.
Please contact me if you have any questions.
Todd.

ORIGINAL DOCUMENT: Will not be sent NUMBER OF PAGES (Including Cover):

CONFIRMATION NO.: CLIENT MATTER NO.: 43649.0001

PLEASE RETURN TO: Debbie PERSONAL FAX: No

REQUESTOR: Todd M. Shaughnessy DIRECT LINE: 801-257-1937

IF YOU HAVE NOT PROPERLY RECEIVED THIS TELECOPY, PLEASE CALL US AT (801) 257-1922. OUR FACSIMILE NUMBER IS (801) 257-1800.

THE INFORMATION CONTAINED IN THIS FACSIMILE MESSAGE IS ATTORNEY PRIVILEGED AND CONFIDENTIAL INFORMATION INTENDED ONLY FOR THE USE OF THE INDIVIDUAL OR ENTITY NAMED ABOVE. IF THE READER OF THIS MESSAGE IS NOT THE INTENDED RECIPIENT, OR THE EMPLOYEE OR AGENT RESPONSIBLE TO DELIVER IT TO THE INTENDED RECIPIENT, YOU ARE HEREBY NOTIFIED THAT ANY DISSEMINATION, DISTRIBUTION OR COPYING OF THIS COMMUNICATION IS STRICTLY PROHIBITED. IF YOU HAVE RECEIVED THIS COMMUNICATION IN ERROR, PLEASE IMMEDIATELY NOTIFY US BY TELEPHONE, AND RETURN THE ORIGINAL MESSAGE TO US AT THE ABOVE ADDRESS VIA THE U.S. POSTAL SERVICE. THANK YOU.

EXHIBIT C

EXHIBIT C

SCO Item Number	SCO's Description of Disclosure
3	NUMA Aware locks from ptx to Linux
4	Disclosure of Dynix/PTX NUMA-aware spinlocks and statement that they have been ported to Linux.
5	Disclosure of Dynix/PTX "jlock" by porting it to Linux.
6	Disclosure of "decoder ring" with ptx primitives in column 1 and the closest Linux equivalent in column 2
8	Confirmation that all ptx optimizations are in patch submitted to Linux by D. Sarma
10	Port of highly parallel distributed lock manager from ptx to Linux
11	Port of Sequent SPIE test suites to Linux
12	Evidence of disclosure of ptx RCU into Linux
15	Detailed disclosure of ptx NUMA-aware locks for adaptation and use in Linux
16	O_Direct ptx SPIE tests ported to Linux and also disclosed in documentation to "Andrea"
17	Port of discontinuous memory code from ptx to Linux 2.5
18	150,000 lines of testing code ported from ptx spie test suites to Linux Test Project. Additional test suites and test cases ported from ptx and AIX to Linux.
19	Disclosure of Dynix/PTX implementations of NUMA-aware locking.
20	Confirmation that ptx was used as as source reference for Linux development
22	Port of ptx NUMA code to Linux
24	Confirmation that ptx was used as source reference for Linux development
25	Use of ptx and AIX technology as roadmap for Linux in various significant areas (locks, counters, search trees, allocators and RDBMS)
27	Transferring ptx source code to AIX developers
28	Disclosure of implementation of reference counters from ptx to Linux
31	Implementation of ptx locking algorithms in Linux
32	Submission of NUMA APIs from Dynix/ptx and AIX to open source, release of klock from ptx to Linux and AIX, and suspected release of NUMA-aware locks to Linux
33	Authorization for open-source disclosure of AIX and ptx NUMA-aware locking primitives
39	Use of ptx as source reference for programming Linux
41	Use of ptx as source reference for memory programming in Linux
55	Disclosure of Dynix/PTX code and method for avoiding a lock via "cut-and-paste from ptx code".
64	Use of Dynix/ptx as source reference for programming memory virtual address space in Linux
79	Confirmation that an earlier RCU patch was based on the Dynix/PTX algorithm.
82	Disclosure that patch is based on the Dynix/PTX implementation, and that it uses a per-CPU context-switch counter.

EXHIBIT C

SCO Item Number	SCO's Description of Disclosure
84	Discloses RCU patches, and acknowledges that they were based on original Dynix/PTX code.
87	Disclosure of Dynix/PTX RCU code, documentation, and API.
91	Dynix/ptx RCU facilities in AIX
92	Information that IBM contributed Dynix/PTX code to Linux from "michael," who appears to be a former Sequent employee. Possibly M. Anderson.
98	Use of ptx NUMA internals in Linux
99	NUMA aware spinlocks developed originally for ptx ported to Linux
100	Sequent Lock Manager (SLM) ported to from ptx to Linux 2.4 using RCU locks and patches of other ptx primitives
102	Suspected disclosure of AIX APIs to SuSE for use in Linux and disclosure of training slideset on ptx scheduler for use in Linux scheduler maintenance
103	Discosure of AIX and ptx APIs for NUMA in Linux
104	Confirmation of intended disclosure of design documents and API descriptions from ptx to SuSE
105	Suspected disclosure of AIX and ptx design documents SuSE
107	Reference to ptx source when creating Linux RCU
108	Suspected use of SMP scaling data points from AIX and ptx for use in Linux
109	Use of ptx design information (Macsyma scripts) to do rclock performance analysis in Linux for NUMA and SMP
112	Suspected disclosure of System V package tools for use by Verizon in Linux
143	Disclosure of nineteen test suites from ptx to Linux Test Project
144	Disclosure of Dynix/ptx and AIX algorithms and techniques via the K42 development project
145	Evidence of IBM's building on ptx performance and correctness experience in coding Linux
146	Use of Dynix/ptx for Linux development
147	Delivery of 400 ptx test cases to Linux Test Project
148	Coding linux equivalent for kmem goodptr ptx primitive; ported ktest rc from ptx to linux
149	Emulation of the SVr4 system implementation in memory mapping
165	Disclosure of STREAMS implementation from SVR4
166	Disclosure of STREAMS implementation from SVR4
167	Disclosure of SVR4 mapping of virtual memory page 0 as "read only"
169	Copying from SVR4 ELF specifications for IBM S390 Linux implementation
170	Memory mapping page 0 as "read only," copying from SVR4
171	Use of SVR4 ABI as source reference in Linux programming, revealing details of SVR4 ELF/ABI specification and use of SVR4 ELF/ABI specification to develop Linux
172	Attempts to reference of SV43/i386 specs from SCO website as source reference for Linux programming and revealing information from SVR4/ABI specification

EXHIBIT C

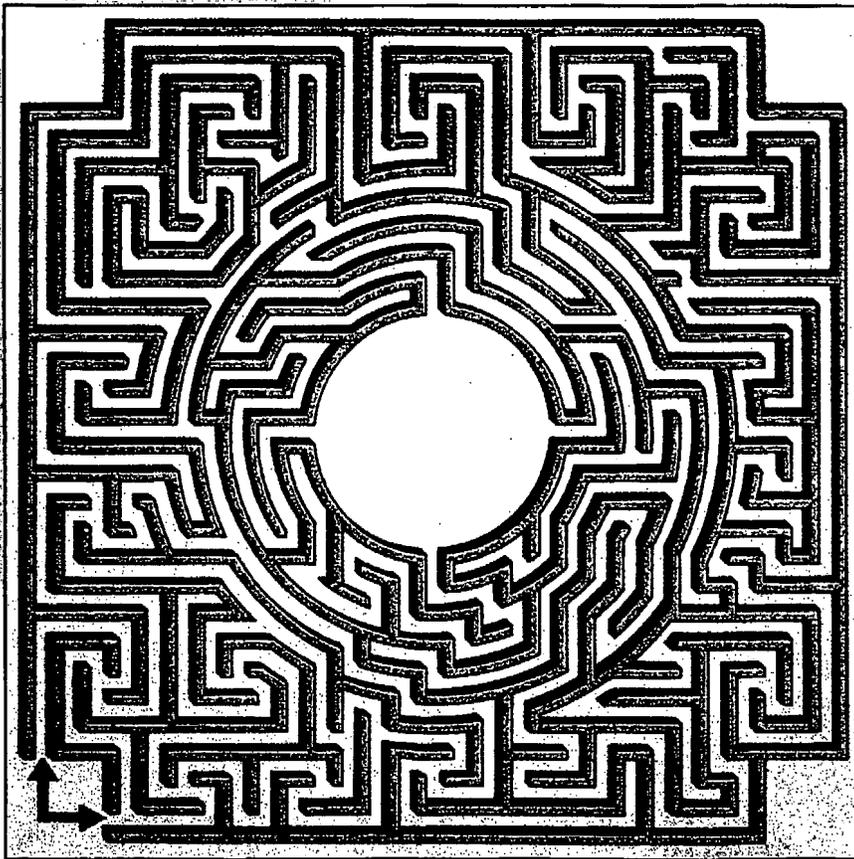
SCO Item Number	SCO's Description of Disclosure
173	Use of SVR4 ABI as source reference for programming Linux
174	Proposal for use of SVR4 internals for Linux development
175	Use of SVR4 as source reference for programming memory mapping system call in Linux
176	Use of ptx as source reference for programming ESR in Linux
177	Use of SVR4 as source reference in programming VFS for Linux
178	Disclosure of Enterprise Volume Management System code from AIX to Linux
179	Disclosure of Enterprise Class Event Logging code from AIX and Dynix to Linux
180	Use of SVR4 internals as a reference for programming Linux
182	Disclosure of Dynix/PTX Direct I/O Test Suite.
189	Disclosure of Dynix/ptx MPIO technology
192	Disclosure of Dynix/ptx virtual memory implementation techniques
193	Disclosure of Dynix/ptx fast walk and dcache implementation techniques
281	Port of ptx ktest-rc from ptx to Linux
282	Use of ptx performance counters in programming Linux
284	Dynix/ptx implementation of RCU
285	Dynix/ptx implementation of RCU
286	Dynix/ptx implementation of RCU
287	Dynix/ptx implementation of RCU
288	Dynix/ptx implementation of RCU
289	AIX network test tool
291	Port of ptx lightweight reader-writer lock
292	Dynix/ptx kmem
293	Dynix/ptx totimeout

EXHIBIT D

► Updated Classic!

Advanced UNIX[®] Programming

SECOND EDITION



MARC J. ROCHKIND

◆ ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

To sort things out, it's not enough to have complete documentation, just as the Yellow Pages isn't enough to find a good restaurant or hotel. You need a guide that tells you what's good and bad, not just what exists. That's the purpose of this book, and why it's different from most other UNIX programming books. I tell you not only how to use the system calls, but also which ones to stay away from because they're unnecessary, obsolete, improperly implemented, or just plain poorly designed.

Here's how I decided what to include in this book: I started with the 1108 functions defined in Version 3 of the Single UNIX Specification and eliminated about 590 Standard C and other library functions that aren't at the kernel-interface level, about 90 POSIX Threads functions (keeping a dozen of the most important), about 25 accounting and logging functions, about 50 tracing functions, about 15 obscure and obsolete functions, and about 40 functions for scheduling and other things that didn't seem to be generally useful. That left exactly 307 for this book. (See Appendix D for a list.) Not that the 307 are all good—some of them are useless, or even dangerous. But those 307 are the ones you need to know.

This book doesn't cover kernel implementation (other than some basics), writing device drivers, C programming (except indirectly), UNIX commands (shell, vi, emacs, etc.), or system administration.

There are nine chapters: Fundamental Concepts, Basic File I/O, Advanced File I/O, Terminal I/O, Processes and Threads, Basic Interprocess Communication, Advanced Interprocess Communication, Networking and Sockets, and Signals and Timers. Read all of Chapter 1, but then feel free to skip around. There are lots of cross-references to keep you from getting lost.

Like the first edition, this new book includes thousands of lines of example code, most of which are from realistic, if simplified, applications such as a shell, a full-screen menu system, a Web server, and a real-time output recorder. The examples are all in C, but I've provided interfaces in Appendices B and C so you can program in C++, Java, or Jython (a variant of Python) if you like.

The text and example code are just resources; you really learn UNIX programming by doing it. To give you something to do, I've included exercises at the end of each chapter. They range in difficulty from questions that can be answered in a few sentences to simple programming problems to semester-long projects.

I used four UNIX systems for nuts-and-bolts research and to test the examples: Solaris 8, SuSE Linux 8 (2.4 kernel), FreeBSD 4.6, and Darwin (the Mac OS X

way to do it. Also, it makes it easier later if you're looking for a bug and are trying to compare the code to the documentation. You don't have to solve a little puzzle in your head to compare the two.

1.4 Error Handling

Testing error returns from system calls is tricky, and handling an error once you discover it is even trickier. This section explains the problem and gives some practical solutions.

1.4.1 Checking for Errors

Most system calls return a value. In the `read` example (Section 1.3.1), the number of bytes read is returned. To indicate an error, a system call usually returns a value that can't be mistaken for valid data, typically `-1`. Therefore, my example should have been coded something like this:

```
if ((amt = read(fd, buf, numbyte)) == -1) {
    fprintf(stderr, "Read failed!\n");
    exit(EXIT_FAILURE);
}
```

Note that `exit` is a system call too, but it can't return an error because it doesn't return. The symbol `EXIT_FAILURE` is in Standard C.

Of the system calls covered in this book, about 60% return `-1` on an error, about 20% return something else, such as `NULL`, zero, or a special symbol like `SIG_ERR`, and about 20% don't report errors at all. So, you just can't assume that they all behave the same way—you have to read the documentation for each call. I'll provide the information when I introduce each system call.

There are lots of reasons why a system call that returns an error indication might have failed. For 80% of them, the integer symbol `errno` contains a code that indicates the reason. To get at `errno` you include the header `errno.h`. You can use `errno` like an integer, although it's not necessarily an integer variable. If you're using threads, `errno` is likely to involve a function call, because the different threads can't reliably all use the same global variable. So don't declare `errno` yourself (which used to be exactly what you were supposed to do), but use the definition in the header, like this (other headers not shown):

```
#include <errno.h>

if ((amt = read(fd, buf, numbyte)) == -1) {
    fprintf(stderr, "Read failed! errno = %d\n", errno);
    exit(EXIT_FAILURE);
}
```

If, say, the file descriptor is bad, the output would be:

```
Read failed! errno = 9
```

Almost always, you can use the value of `errno` only if you've first checked for an error; you can't just check `errno` to see if an error occurred, because its value is set only when a function that is specified to set it returns an error. So, this code would be wrong:

```
amt = read(fd, buf, numbyte);
if (errno != 0) { /* wrong! */
    fprintf(stderr, "Read failed! errno = %d\n", errno);
    exit(EXIT_FAILURE);
}
```

Setting `errno` to zero first works:

```
errno = 0;
amt = read(fd, buf, numbyte);
if (errno != 0) { /* bad! */
    fprintf(stderr, "Read failed! errno = %d\n", errno);
    exit(EXIT_FAILURE);
}
```

But it's still a bad idea because:

- If you modify the code later to insert another system call, or any function that eventually executes a system call, before the call to `read`, the value of `errno` may be set by *that* call.
- Not all system calls set the value of `errno`, and you should get into the habit of checking for an error that conforms exactly to the function's specification.

Thus, for almost all system calls, check `errno` only after you've established that an error occurred.

Now, having warned you about using `errno` alone to check for an error, this being UNIX, I have to say that there are a few exceptions (e.g., `sysconf` and `readdir`) that do rely on a changed `errno` value to indicate an error, but even they return a specific value that tells you to check `errno`. Therefore, the rule

about not checking `errno` before first checking the return value is a good one, and it applies to most of the exceptions, too.

The `errno` value 9 that was displayed a few paragraphs up doesn't mean much, and isn't standardized, so it's better to use the Standard C function `perror`, like this:

```
if ((amt = read(fd, buf, numbyte)) == -1) {
    perror("Read failed!");
    exit(EXIT_FAILURE);
}
```

Now the output is:

```
Read failed!: Bad file number
```

Another useful Standard C function, `strerror`, just provides the message as a string, without also displaying it like `perror` does.

But the message "Bad file number," while clear enough, isn't standardized either, so there's still a problem: The official documentation for system calls and other functions that use `errno` refer to the various errors with symbols like `EBADF`, not by the text messages. For example, here's an excerpt from the SUS entry for `read`:

[EAGAIN]

The `O_NONBLOCK` flag is set for the file descriptor and the process would be delayed.

[EBADF]

The `files` argument is not a valid file descriptor open for reading.

[EBADMSG]

The file is a `STREAM` file that is set to control-normal mode and the message waiting to be read includes a control part.

It's straightforward to match "Bad file number" to `EBADF`, even though those exact words don't appear in the text, but not for the more obscure errors. What you really want along with the text message is the actual symbol, and there's no Standard C or SUS function that gives it to you. So, we can write our own function that translates the number to the symbol. We built the list of symbols in the code that follows from the `errno.h` files on Linux, Solaris, and BSD, since many symbols are system specific. You'll probably have to adjust this code for your

own system. For brevity, not all the symbols are shown, but the complete code is on the AUP Web site (Section 1.8 and [AUP2003]).

```
static struct {
    int code;
    char *str;
} errcodes[] =
{
    { EPERM, "EPERM" },
    { ENOENT, "ENOENT" },
    --
    { EINPROGRESS, "EINPROGRESS" },
    { ESTALE, "ESTALE" },
#ifdef BSD
    { ECHRNG, "ECHRNG" },
    { EL2NSYNC, "EL2NSYNC" },
    --
    { ESTRPIPE, "ESTRPIPE" },
    { EDQUOT, "EDQUOT" },
#ifdef SOLARIS
    { EDOTDOT, "EDOTDOT" },
    { EUCLEAN, "EUCLEAN" },
    --
    { ENOMEDIUM, "ENOMEDIUM" },
    { EMEDIUMTYPE, "EMEDIUMTYPE" },
#endif
#endif
    { 0, NULL}
};

const char *errsymbol(int errno_arg)
{
    int i;

    for (i = 0; errcodes[i].str != NULL; i++)
        if (errcodes[i].code == errno_arg)
            return errcodes[i].str;
    return "[UnknownSymbol]";
}
```

Here's the error checking for read with the new function:

```
if ((amt = read(fd, buf, numbyte)) == -1) {
    fprintf(stderr, "Read failed!: %s (errno = %d; %s)\n",
        strerror(errno), errno, errsymbol(errno));
    exit(EXIT_FAILURE);
}
```

Now the output is complete:

```
Read failed!: Bad file descriptor (errno = 9; EBADF)
```

It's convenient to write one more utility function to format the error information, so we can use it in some code we're going to write in the next section:

```
char *syserrmsg(char *buf, size_t buf_max, const char *msg, int errno_arg)
{
    char *errmsg;

    if (msg == NULL)
        msg = "???";
    if (errno_arg == 0)
        snprintf(buf, buf_max, "%s", msg);
    else {
        errmsg = strerror(errno_arg);
        snprintf(buf, buf_max, "%s\n\t\t*** %s (%d: \"%s\") ***", msg,
            errsymbol(errno_arg), errno_arg,
            errmsg != NULL ? errmsg : "no message string");
    }
    return buf;
}
```

We would use `syserrmsg` like this:

```
if ((amt = read(fd, buf, numbyte)) == -1) {
    fprintf(stderr, "%s\n", syserrmsg(buf, sizeof(buf),
        "Call to read function failed", errno));
    exit(EXIT_FAILURE);
}
```

with output like this:

```
Call to read function failed
*** EBADF (9: "Bad file descriptor") ***
```

What about the other 20% of the calls that report an error but don't set `errno`? Well, around 20 of them report the error another way, typically by simply returning the error code (that is, a non-zero return indicates that an error occurred and also what the code is), and the rest don't provide a specific reason at all. I'll provide the details for every function (all 300 or so) in this book. Here's one that returns the error code directly (what this code is supposed to do isn't important right now):

```
struct addrinfo *infop;

if ((r = getaddrinfo("localhost", "80", NULL, &infop)) != 0) {
```

```
    fprintf(stderr, "Got error code %d from getaddrinfo\n", r);
    exit(EXIT_FAILURE);
}
```

The function `getaddrinfo` is one of those that doesn't set `errno`, and you can't pass the error code it returns into `strerror`, because that function works only with `errno` values. The various error codes returned by the non-`errno` functions are defined in [SUS2002] or in your system's manual, and you certainly could write a version of `errsymbol` (shown earlier) for those functions. But what makes this difficult is that the symbols for one function (e.g., `EAI_BADFLAGS` for `getaddrinfo`) aren't specified to have values distinct from the symbols for another function. This means that you can't write a function that takes an error code alone and looks it up, like `errsymbol` did. You have to pass the name of the function in as well. (If you do, you could take advantage of `gai_strerror`, which is a specially tailored version of `strerror` just for `getaddrinfo`.)

There are about two dozen functions in this book for which the standard [SUS2002] doesn't define any `errno` values or even say that `errno` is set, but for which your implementation may set `errno`. The phrase "errno not defined" appears in the function synopses for these.

Starting to get a headache? UNIX error handling is a real mess. This is unfortunate because it's hard to construct test cases to make system calls misbehave so the error handling you've coded can be checked, and the inconsistencies make it hard to get it right every single time. But the chances of it getting any better soon are zero (it's frozen by the standards), so you'll have to live with it. Just be careful!

1.4.2 Error-Checking Convenience Macros for C

It's tedious to put every system call in an `if` statement and follow it with code to display the error message and exit or return. When there's cleanup to do, things get even worse, as in this example:

```
if ((p = malloc(sizeof(buf))) == NULL) {
    fprintf(stderr, "%s\n", syserrmsg(buf, sizeof(buf),
        "malloc failed", errno));
    return false;
}
if ((fdin = open(filein, O_RDONLY)) == -1) {
    fprintf(stderr, "%s\n", syserrmsg(buf, sizeof(buf),
        "open (input) failed", errno));
```

```

    free(p);
    return false;
}
if ((fdout = open(fileout, O_WRONLY)) == -1) {
    fprintf(stderr, "%s\n", syserrmsg(buf, sizeof(buf),
        "open (output) failed", errno));
    (void)close(fdin);
    free(p);
    return false;
}

```

The cleanup code gets longer and longer as we go. It's hard to write, hard to read, and hard to maintain. Many programmers will just use a `goto` so the cleanup code can be written just once. Ordinarily, `gotos` are to be avoided, but here they seem worthwhile. Note that we have to carefully initialize the variables that are involved in cleanup and then test the file-descriptor values so that the cleanup code can execute correctly no matter what the incomplete state.

```

char *p = NULL;
int fdin = -1, fdout = -1;

if ((p = malloc(sizeof(buf))) == NULL) {
    fprintf(stderr, "%s\n", syserrmsg(buf, sizeof(buf),
        "malloc failed", errno));
    goto cleanup;
}
if ((fdin = open(filein, O_RDONLY)) == -1) {
    fprintf(stderr, "%s\n", syserrmsg(buf, sizeof(buf),
        "open (input) failed", errno));
    goto cleanup;
}
if ((fdout = open(fileout, O_WRONLY)) == -1) {
    fprintf(stderr, "%s\n", syserrmsg(buf, sizeof(buf),
        "open (output) failed", errno));
    goto cleanup;
}
return true;

cleanup:
free(p);
if (fdin != -1)
    (void)close(fdin);
if (fdout != -1)
    (void)close(fdout);
return false;

```

Still, coding all those `ifs`, `fprints`, and `gotos` is a pain. The system calls themselves are almost buried!

We can streamline the jobs of checking for the error, displaying the error information, and getting out of the function with some macros. I'll first show how they're used and then how they're implemented. (This stuff is just Standard C coding, not especially connected to system calls or even to UNIX, but I've included it because it'll be used in all the subsequent examples in this book.)

Here's the previous example rewritten to use these error-checking ("ec") macros. The context isn't shown, but this code is inside of a function named `fcn`:

```
char *p = NULL;
int fdin = -1, fdout = -1;

ec_null( p = malloc(sizeof(buf)) )
ec_neg1( fdin = open(filein, O_RDONLY) )
ec_neg1( fdout = open(fileout, O_WRONLY) )

return true;

EC_CLEANUP_BGN
free(p);
if (fdin != -1)
    (void)close(fdin);
if (fdout != -1)
    (void)close(fdout);
return false;
EC_CLEANUP_END
```

Here's the call to the function. Because it's in the main function, it makes sense to exit on an error.

```
ec_false( fcn() )

/* other stuff here */

exit(EXIT_SUCCESS);

EC_CLEANUP_BGN
exit(EXIT_FAILURE);
EC_CLEANUP_END
```

Here's what's going on: The macros `ec_null`, `ec_neg1`, and `ec_false` check their argument expression against `NULL`, `-1`, and `false`, respectively, store away the error information, and go to a label that was placed by the `EC_CLEANUP_BGN` macro. Then the same cleanup code as before is executed. In main, the test of the return value of `fcn` also causes a jump to the same label in main and the program exits. A function installed with `atexit` (introduced in Section 1.3.4) displays all the accumulated error information:

```
ERROR: 0: main [/aup/c1/errorhandling.c:41] fcn()
      1: fcn [/aup/c1/errorhandling.c:15] fdin = open(filein, 0x0000)
      *** ENOENT (2: "No such file or directory") ***
```

What we see is the `errno` symbol, value, and descriptive text on the last line. It's preceded by a reverse trace of the error returns. Each trace line shows the level, the name of the function, the file name, the line number, and the code that returned the error indication. This isn't the sort of information you want your end users to see, but during development it's terrific. Later, you can change the macros (we'll see how shortly) to put these details in a log file, and your users can see something more meaningful to them.

We accumulated the error information rather than printing it as we went along because that gives an application developer the most freedom to handle errors as he or she sees fit. It really doesn't do for a function in a library to just write error messages to `stderr`. That may not be the right place, and the wording may not be appropriate for the application's end users. In the end we did print it, true, but that decision can be easily changed if you use these macros in a real application.

So, what these macros give us is:

- Easy and readable error checking
- An automatic jump to cleanup code
- Complete error information along with a back trace

The downside is that the macros have a somewhat strange syntax (no semicolon at the end) and a buried jump in control flow, which some programmers think is a very bad idea. If you think the benefits outweigh the costs, use the macros (as I will in this book). If not, work out your own set (maybe with an explicit `goto` instead of a buried one), or skip them entirely.

Here's most of the header file (`ec.h`) that implements the error-checking macros (function declarations and a few other minor details are omitted):

```
extern const bool ec_in_cleanup;

typedef enum {EC_ERRNO, EC_EAI} EC_ERRTYPE;

#define EC_CLEANUP_BGN\  
    ec_warn();\  
    ec_cleanup_bgn:\  
    {\  
        bool ec_in_cleanup;\  
        ec_in_cleanup = true;
```

```
#define EC_CLEANUP_END\
)

#define ec_cmp(var, errrtn)\
{\
    assert(!ec_in_cleanup);\
    if ((intptr_t)(var) == (intptr_t)(errrtn)) {\
        ec_push(__func__, __FILE__, __LINE__, #var, errno, EC_ERRNO);\
        goto ec_cleanup_bgn;\
    }\
}

#define ec_rv(var)\
{\
    int errrtn;\
    assert(!ec_in_cleanup);\
    if ((errrtn = (var)) != 0) {\
        ec_push(__func__, __FILE__, __LINE__, #var, errrtn, EC_ERRNO);\
        goto ec_cleanup_bgn;\
    }\
}

#define ec_ai(var)\
{\
    int errrtn;\
    assert(!ec_in_cleanup);\
    if ((errrtn = (var)) != 0) {\
        ec_push(__func__, __FILE__, __LINE__, #var, errrtn, EC_EAI);\
        goto ec_cleanup_bgn;\
    }\
}

#define ec_negl(x) ec_cmp(x, -1)
#define ec_null(x) ec_cmp(x, NULL)
#define ec_false(x) ec_cmp(x, false)
#define ec_eof(x) ec_cmp(x, EOF)
#define ec_nzero(x)\
{\
    if ((x) != 0)\
        EC_FAIL\
}

#define EC_FAIL ec_cmp(0, 0)

#define EC_CLEANUP goto ec_cleanup_bgn;
```

```
#define EC_FLUSH(str)\
{\
    ec_print();\
    ec_reinit();\
}
```

Before I explain the macros, I have to bring up a problem and talk about its solution. The problem is that if you call one of the error-checking macros (e.g., `ec_neg1`) inside the cleanup code and an error occurs, there will most likely be an infinite loop, since the macro will jump to the cleanup code! Here's what I'm worried about:

```
EC_CLEANUP_BGN
    free(p);
    if (fdin != -1)
        ec_neg1( close(fdin) )
    if (fdout != -1)
        ec_neg1( close(fdout) )
    return false;
EC_CLEANUP_END
```

It looks like the programmer is being very careful to check the error return from `close`, but it's a disaster in the making. What's really bad about this is that the loop would occur only when there was an error cleaning up after an error, a rare situation that's unlikely to be caught during testing. We want to guard against this—the error-checking macros should increase reliability, not reduce it!

Our solution is to set a local variable `ec_in_cleanup` to true in the cleanup code, which you can see in the definition for the macro `EC_CLEANUP_BGN`. The test against it is in the macro `ec_cmp`—if it's set, the assert will fire and we'll know right away that we goofed.

(The type `bool` and its values, `true` and `false`, are new in C99. If you don't have them, you can just stick the code

```
typedef int bool;
#define true 1
#define false 0
```

in one of your header files.)

To prevent the assertion from firing when `ec_cmp` is called outside of the cleanup code (i.e., a normal call), we have a global variable, also named `ec_in_cleanup`, that's permanently set to `false`. This is a rare case when it's OK (essential, really) to hide a global variable with a local one.

Why have the local variable at all? Why not just set the global to `true` at the start of the cleanup code, and back to `false` at the end? That won't work if you call a function from within the cleanup code that happens to use the `ec_cmp` macro legitimately. It will find the global set to `true` and think it's in its own cleanup code, which it isn't. So, each function (that is, each unique cleanup-code section) needs a private guard variable.

Now I'll explain the macros one-by-one:

- `EC_CLEANUP_BGN` includes the label for the cleanup code (`ec_cleanup_bgn`), preceded by a function call that just outputs a warning that control flowed into the label. This guards against the common mistake of forgetting to put a `return` statement before the label and flowing into the cleanup code even when there was no error. (I put this in after I wasted an hour looking for an error that wasn't there.) Then there's the local `ec_in_cleanup`, which I already explained.
- `EC_CLEANUP_END` just supplies the closing brace. We needed the braces to create the local context.
- `ec_cmp` does most of the work: Ensuring we're not in cleanup code, checking the error, calling `ec_push` (which I'll get to shortly) to push the location information (`__FILE__`, etc.) onto a stack, and jumping to the cleanup code. The type `intptr_t` is new in C99: It's an integer type guaranteed to be large enough to hold a pointer. If you don't have it yet, `typedef` it to be a `long` and you'll probably be OK. Just to be extra safe, stick some code in your program somewhere to test that `sizeof(void *)` is equal to `sizeof(long)`. (If you're not familiar with the notation `#var`, read up on your C—it makes whatever `var` expands to into a string.)
- `ec_rv` is similar to `ec_cmp`, but it's for functions that return a non-zero error code to indicate an error and which don't use `errno` itself. However, the codes it returns are `errno` values, so they can be passed directly to `ec_push`.
- `ec_ai` is similar to `ec_rv`, but the error codes it deals with aren't `errno` values. The last argument to `ec_push` becomes `EC_EAI` to indicate this. (Only a couple of functions, both in Chapter 8, use this approach.)
- The macros `ec_neg1`, `ec_null`, `ec_false`, and `ec_eof` call `ec_cmp` with the appropriate arguments, and `ec_nzero` does its own checking. They cover the most common cases, and we can just use `ec_cmp` directly for the others.

- `EC_FAIL` is used when an error condition arises from a test that doesn't use the macros in the previous paragraph.
- `EC_CLEANUP` is used when we just want to jump to the cleanup code.
- `EC_FLUSH` is used when we just want to display the error information, without waiting to exit. It's handy in interactive programs that need to keep going. (The argument isn't used.)

The various service functions called from the macros won't be shown here, since they don't illustrate much about UNIX system calls (they just use Standard C), but you can go to the AUP Web site [AUP2003] to see them along with an explanation of how they work. Here's a summary:

- `ec_push` pushes the error and context information passed to it (by the `ec_cmp` macro, say) onto a stack.
- There's a function registered with `atexit` that prints the information on the stack when the program exits:

```
static void ec_atexit_fcn(void)
{
    ec_print();
}
```

- `ec_print` walks down the stack to print the trace and error information.
- `ec_reinit` erases what's on the stack, so error-checking can start with a fresh trace.
- `ec_warn` is called from the `EC_CLEANUP_BGN` code if we accidentally fall into it.

All the functions are thread-safe so they can be used from within multithreaded programs. More on what this means in Section 5.17.

1.4.3 Using C++ Exceptions

Before you spend too much time and energy deciding whether you like the "ec" macros in the previous section and coming up with some improvements, you might ask yourself whether you'll even be programming in C. These days it's much more likely that you'll use C++. Just about everything in this book works fine in a C++ program, after all. C is still often preferred for embedded systems, operating systems (e.g., Linux), compilers, and other relatively low-level software, but those kinds of systems are likely to have their own, highly specialized, error-handling mechanisms.

C++ provides an opportunity to handle errors with exceptions, built into the C++ language, rather than with the combination of `gotos` and `return` statements that

we used in C. Exceptions have their own pitfalls, but if used carefully they're easier to use and more reliable than the "ec" macros, which don't protect you from, say, using `ec_null` when you meant to use `ec_neg1`.

As the library that contains the system-call wrapper code is usually set up just for C, it won't throw exceptions unless someone's made a special version for C++. So, to use exceptions you need another layer of wrapping, something like this for the `close` system call:

```
class syscall_ex {
public:
    int se_errno;

    syscall_ex(int n)
        : se_errno(n)
    { }
    void print(void)
    {
        fprintf(stderr, "ERROR: %s\n", strerror(se_errno));
    }
};

class syscall {
public:
    static int close(int fd)
    {
        int r;
        if ((r = ::close(fd)) == -1)
            throw(syscall_ex(errno));
        return r;
    }
};
```

Then you just call `syscall::close` instead of plain `close`, and it throws an exception on an error. You probably don't want to type in the code for the other 1100 or so UNIX functions, but perhaps just the ones your application uses.

If you want the exception information to include location information such as file and line, you need to define *another* wrapper, this time a macro, to capture the preprocessor data (e.g., via `__LINE__`),¹³ so here's an even fancier couple of classes:

13. We need the macro because if you just put `__LINE__` and the others as direct arguments to the `syscall_ex` constructor, you get the location in the definition of `class syscall`, which is the wrong place.

```

class syscall_ex {
public:
    int se_errno;
    const char *se_file;
    int se_line;
    const char *se_func;

    syscall_ex(int n, const char *file, int line, const char *func)
        : se_errno(n), se_file(file), se_line(line), se_func(func)
    { }
    void print(void)
    {
        fprintf(stderr, "ERROR: %s [%s:%d %s()]\n",
                strerror(se_errno), se_file, se_line, se_func);
    }
};

class syscall {
public:
    static int close(int fd, const char *file, int line, const char *func)
    {
        int r;
        if ((r = ::close(fd)) == -1)
            throw(syscall_ex(errno, file, line, func));
        return r;
    }
};

#define Close(fd) (syscall::close(fd, __FILE__, __LINE__, __func__))

```

This time you call `Close` instead of `close`.

You can goose this up with a call-by-call trace, as we did for the “ec” macros if you like, and probably go even further than that.

There’s an example C++ wrapper, `Ux`, for all the system calls in this book that’s described in Appendix B.

1.5 UNIX Standards

Practically speaking, what you’ll mostly find in the real world is that the commercial UNIX vendors follow the Open Group branding (Section 1.2), and the open-source distributors claim only conformance to POSIX1990, although, with few exceptions, they don’t bother to actually run the certification tests. (The tests have now been made freely available, so future certification is now more likely.)

EXHIBIT E

In The Matter Of:

**THE SCO GROUP, INC., v.
INTERNATIONAL BUSINESS MACHINES CORPORATION**

SANDEEP GUPTA

March 17, 2006

CONFIDENTIAL

LEGALINK MANHATTAN

420 Lexington Avenue - Suite 2108

New York, NY 10170

PH: 212-557-7400 / FAX: 212-692-9171

GUPTA, SANDEEP - Vol. I



LEGALINK

A WORDWARE COMPANY

Page 1

1 CONFIDENTIAL
2
3 IN THE UNITED STATES DISTRICT COURT
4 FOR THE DISTRICT OF UTAH
5 THE SCO GROUP, INC., : CIVIL ACTION
6 Plaintiff/Counterclaim- :
7 Defendant,
8
9 -v-
10 :
11 INTERNATIONAL BUSINESS
12 MACHINES CORPORATION, :
13 Defendant/Counterclaim- : NO. 2:03CV-294 DAK
14 Plaintiff.
15
16
17
18
19
20
21
22
23
24
25

MARCH 17, 2006
9:08 A.M.

Videotaped deposition of SANDEEP GUPTA,
taken by Defendants, at the offices of
BOIES, SCHILLER & FLEXNER LLP, 150 John F.
Kennedy Parkway, Short Hills, New Jersey,
before Debra Sapio Lyons, a Registered
Diplomat Reporter, a Certified Realtime
Reporter, a Certified Shorthand Reporter and
Notary Public of the States of New Jersey
and New York.

Page 2

1
2 APPEARANCES:
3 BOIES, SCHILLER & FLEXNER LLP
4 Attorneys for the Plaintiff
5 10 North Pearl Street
6 Albany, New York 12207
7
8 BY: DANIEL P. FILOR, ESQUIRE
9
10
11
12 CRAVATH, SWAINE & MOORE LLP
13 Attorneys for the Defendant
14 825 Eighth Avenue
15 New York, New York 10019-7475
16
17 BY: MICHAEL P. BURKE, ESQUIRE
18 AND
19 ANA TERESA MIRANDA-KREYSZIG, ESQUIRE
20
21
22
23
24
25

ALSO PRESENT:
DOUGLAS HUEBNER, VIDEO TECHNICIAN
LEGALINK ACTION VIDEO

Page 3

1 SANDEEP GUPTA - CONFIDENTIAL
2 THE VIDEO TECHNICIAN: This is
3 the video operator speaking, Douglas
4 Huebner of Legalink Action Video, 420
5 Lexington Avenue, New York, New York.
6 Today is March 17th, 2006 and the time is
7 9:08.
8 We're at the offices of Boies,
9 Schiller, 150 Kennedy Parkway, Short
10 Hills, New Jersey to take the video
11 deposition of Sandeep Gupta in the matter
12 of the SCO Group, Inc., versus
13 International Business Machines Corp., in
14 the United States District Court for the
15 District of Utah, Civil Action Number
16 2:03CV-294 DAK.
17 Will counsel please introduce
18 themselves for the record.
19 MR. BURKE: Mike Burke from
20 Cravath, Swaine & Moore on behalf of IBM.
21 MS. MIRANDA-KREYSZIG: Ana
22 Miranda-Kreyszig from Cravath, Swaine &
23 Moore on behalf of IBM.
24 MR. FILOR: Daniel Filor from
25 Boies, Schiller & Flexner on behalf of

Page 4

1 SANDEEP GUPTA - CONFIDENTIAL
2 plaintiff, the SCO Group.
3 THE VIDEO TECHNICIAN: Will the
4 court reporter please swear the witness.
5 ---
6 SANDEEP GUPTA, having been
7 first duly sworn, was examined and
8 testified as follows:
9 ---
10 EXAMINATION
11 ---
12 BY MR. BURKE:
13 Q. Good morning, Mr. Gupta.
14 A. Good morning.
15 Q. My name's Mike Burke. I'm a
16 lawyer for IBM and I'm here to ask you some
17 questions today.
18 A. Uh-huh.
19 Q. Could you just state your
20 current home address for the record?
21 A. Absolutely. I live in 4 Villa
22 Farms Circle, Monroe, New Jersey. That's
23 Zip code 08831.
24 Q. And have you ever been deposed
25 before?

Page 69

1 SANDEEP GUPTA - CONFIDENTIAL
2 the, quote, Date of first issue: 16 May
3 2003, close quote, and then, quote, Next
4 renewal date 16 May 2006, close quote.
5 Do you see that?
6 A. Yes.
7 Q. Do you know if there's any
8 plans to renew this certification?
9 A. I don't know. This definitely
10 not one of my job function.
11 Q. Okay. Do you know whose job
12 function it would be?
13 A. It would be the product
14 management.
15 Q. Okay. And let me -- as of
16 today, could you just describe your job
17 functions?
18 I understand you're the Chief
19 Technology Officer of SCO; is that correct?
20 A. That's correct.
21 Q. Okay.
22 A. That's my title.
23 Q. Could you -- okay.
24 Could you describe for me what
25 your job functions are today?

Page 70

1 SANDEEP GUPTA - CONFIDENTIAL
2 A. My job function today is to
3 bring new products and new technologies to
4 SCO and that's I build the new products and
5 I bring in new technologies into those
6 products.
7 Q. So as Chief Technology Officer
8 of SCO, you don't have responsibility for
9 the -- strike that.
10 As Chief Technology Officer of
11 SCO, you don't have responsibility for SCO's
12 work with the X/Open brand program; correct?
13 A. As a Chief Technology Officer,
14 I don't personally have a direct
15 responsibility. If there are certification
16 tests that we have, then my test team does
17 run it.
18 Q. Have you ever supervised a
19 certification test?
20 A. No, those are supervised by my
21 directors who manage those groups.
22 Q. Okay. And the directors are --
23 you said -- strike that.
24 You referred to your directors.
25 Are those people who are -- who report to

Page 71

1 SANDEEP GUPTA - CONFIDENTIAL
2 you?
3 A. That's correct.
4 Q. Okay. Do you know who would
5 have this responsibility?
6 A. I don't know if I want to give
7 the names out of people right now. I don't
8 know if that's something --
9 MR. FILOR: Well, there's a
10 confidentiality order in the case, so you
11 can disclose those.
12 THE WITNESS: Okay. So under
13 that, I -- I want to protect names of the
14 people --
15 BY MR. BURKE:
16 Q. Sure.
17 A. -- on my team.
18 Stan Krieger is the Senior
19 System Test Manager.
20 Q. Okay.
21 A. He would definitely know about
22 this.
23 Q. Krieger?
24 A. K-R-I-E-G-E-R.
25 Q. Okay. Anyone else?

Page 72

1 SANDEEP GUPTA - CONFIDENTIAL
2 A. I think he's the -- he would
3 know the most and there may be other people.
4 Q. Do you know the names of any of
5 the other folks?
6 A. His team comprises of
7 engineers, and manager level people. One
8 other person would be George Grinlinger.
9 George and Grinlinger, he may know.
10 Q. Do these folks you referred to
11 have other job duties beyond certification
12 tests?
13 A. That's correct.
14 Q. Okay.
15 MR. BURKE: Bless you.
16 THE WITNESS: Coffee is already
17 empty. I'll wait till you guys are ready
18 MR. BURKE: Okay. Mark that
19 for me.
20 MR. FILOR: Since I brought it
21 up, why don't I put on the record now
22 that we're going to be marking this
23 transcript as Confidential under the
24 Protective Order.
25 (Exhibit 1478, string of emails

Page 265

1 SANDEEP GUPTA - CONFIDENTIAL
2 Q. And it -- it sounded like you
3 did. I didn't understand what you said,
4 but...
5 At what point does -- does
6 something go from just a general description
7 to a method?
8 MR. FILOR: Objection to form.
9 THE WITNESS: I am not an
10 expert in this area when it becomes from
11 concept to methods.
12 BY MR. BURKE:
13 Q. Okay. Oh, my question wasn't
14 from concepts to methods.
15 It was just from a general
16 description to a concept or method.
17 MR. FILOR: Objection to form.
18 THE WITNESS: I still -- I'm
19 not an expert to a general -- you know,
20 what do you mean by general description
21 to a concept or method?
22 BY MR. BURKE:
23 Q. Okay. What -- what kind of
24 expert would you think you'd have to be?
25 A. Oh, I --

Page 266

1 SANDEEP GUPTA - CONFIDENTIAL
2 MR. FILOR: Objection to form.
3 THE WITNESS: -- I think you're
4 asking some legal question. So if you
5 can clarify the question, maybe I can
6 answer.
7 BY MR. BURKE:
8 Q. Okay. What I'm asking is --
9 let's see.
10 Well, if I described RCU
11 locking as reading a number of threads,
12 would that describe the method of RCU
13 locking?
14 MR. FILOR: Objection to form.
15 THE WITNESS: Reading a number
16 of threads, probably not.
17 BY MR. BURKE:
18 Q. Okay. Is that part of the
19 method?
20 A. It could be.
21 Q. Okay. How would you determine
22 whether a particular description was
23 specific enough to describe an aspect of
24 System V as a method?
25 MR. FILOR: Objection to form.

Page 267

1 SANDEEP GUPTA - CONFIDENTIAL
2 Calls for legal conclusion.
3 THE WITNESS: I have to look at
4 the source code.
5 BY MR. BURKE:
6 Q. Okay. What would you do if you
7 looked at the source code?
8 A. I look at various steps that
9 are taken, specific for that particular
10 method.
11 Q. Okay. So in order to determine
12 what a particular method or concept is, you
13 would actually have to look at the source
14 code?
15 MR. FILOR: Objection to form.
16 THE WITNESS: In some cases,
17 yes.
18 BY MR. BURKE:
19 Q. Okay. I mean, I -- I
20 understand you just articulated a few from
21 memory and --
22 A. Yeah.
23 Q. -- I'm impressed with that, but
24 in general, would you have to look at the
25 source code to be able to accurately

Page 268

1 SANDEEP GUPTA - CONFIDENTIAL
2 describe a method or concept in UNIX?
3 MR. FILOR: Objection to form.
4 THE WITNESS: That's my
5 opinion, yes.
6 BY MR. BURKE:
7 Q. I think you testified that
8 there were thousands of methods and concepts
9 in System V --
10 A. Uh-huh.
11 Q. -- is that right?
12 If we had a week to do it,
13 would you be able to describe each one of
14 them without referring to the source code?
15 A. No, it would be very hard.
16 Q. Okay. And why it -- why would
17 it be very hard?
18 A. Because it took three decades
19 for UNIX development to get where it is.
20 MR. BURKE: Okay. All right.
21 That's all I've got.
22 THE WITNESS: Thank you.
23 MR. FILOR: Okay.
24 THE VIDEO TECHNICIAN: That's
25 it?

EXHIBIT F

EXHIBIT F

Item 146 of SCO's Final Disclosure, as well as Mr. Rochkind's declaration, leave as an exercise for the technically informed reader the task of determining how five pieces of information fit together to tell a coherent story of alleged wrongdoing. A reader without a technical background must simply either assume that the pieces fit or believe SCO's assertion that they do. In fact, they do not and neither Item 146 nor Mr. Rochkind ever explains what the alleged wrongdoing was. Mr. Rochkind simply asserts that SCO has assembled five things: some emails that prove Linux development occurring, a paper published in the open literature that describes a method, a patch that contains some changes to code, a pointer to scripts (sequences of commands to the computer), and the names of some Linux files that happen to contain the term "profile." This, Mr. Rochkind asserts, constitutes enough detail.

One has only to read the emails carefully and pay attention to dates to see that the pieces do not fit together as Mr. Rochkind claims. The events appear to have occurred as follows: IBM employee Manesh Soni made changes to the code in the Linux file "file.c", as part of an experiment to try to improve the performance of the Linux virtual file system. The changes are shown in the form of a patch,¹ but there is no indication this patch was ever sent outside IBM. From the context, it is obvious that this was an internal experiment, and if it did not yield positive results, the change would never be proposed to the Linux community. According to the email, the patch did not produce positive results (Soni writes "Now the problem is that instead of [the change yielding an] increase in

¹ A "patch" lists just the changes made (e.g., delete the first four lines of the file, insert these seven new lines, modify the 15th line as follows..., etc.). It is used because listing the changes is typically much shorter than listing the entire modified file.

performance we are seeing a drop (near about 30%).” In response Soni asked IBM employee Paul McKenney for advice. McKenney suggested using a technique called “differential profiling,” pointing Soni to a paper published in 1995 that explained the technique, and to some scripts that might be helpful.

The technique McKenney suggested was an analysis technique. To the extent that the scripts may have implemented the technique, they were an analysis tool that existed outside the scope of the operating system itself, either Dynix/ptx or Linux. The paper describing the technique does not mention Dynix/ptx at all. It discusses “programs” as the object of investigation using this technique.

Paying attention to the dates shows that the patch SCO cites (inside email #1) predates the email where McKenney suggests using the alleged “method”, yet SCO states their evidence “contains an actual Linux patch”, implying that the patch is the evidence that the method was carried into Linux. Yet the patch was written before the differential profiling method was even mentioned to Soni.

IBM is left, as is the reader, to figure out how SCO might say the pieces fit together. We are missing any clear allegation. Instead we have at least three possible allegations that can serve as SCO’s cups for a “where’s the pea?” game. But is there a pea? Only SCO knows.

What might SCO’s allegation be?

(1) *SCO might be alleging that “differential profiling,” as a method, was part of the Dynix/ptx operating system, and was re-implemented for and contributed to Linux by IBM. This is the most obvious interpretation of SCO’s allegation, because Mr. Rochkind states very clearly “The method and concept is completely explained by the paper ...”.*

(Rochkind Decl. ¶ 14.) However, this allegation is inconsistent with SCO's statements that the cited patch was an offending contribution, because the patch is an experimental change to the Linux file system, and predates McKenney's proposed use of the differential profiling technique. To interpret SCO's allegation as the initial sentence of this paragraph proposes, would be to suggest that SCO showed willful disregard for basic technical integrity of their allegation.

(2) *SCO might be alleging that a technique (and/or tools) developed to analyze Dynix/ptx performance was subsequently used to analyze (and improve) Linux file system performance.* This allegation would be consistent with Disclosure Items 109 and 289, which seem to say that an internal tool was used first with Dynix/ptx, then later was used with Linux. However, this allegation would again not be consistent with SCO's statements that the cited patch was an offending contribution, since the patch is an experimental change to the Linux file system, and predates McKenney's proposed use of the differential profiling technique. This allegation would be especially inconsistent with SCO's cited location(s) in Linux of the offending method, which is in files named "profile", not in the Linux file system or in "file.c". If this were SCO's allegation, there is no evidence that McKenney's suggestion was ever acted upon, and if it was, there is no evidence Soni was able to figure out his problem. And if he was, there is no evidence Soni came up with a better patch, which had positive results, and if he did, there is no evidence any patch resulting from all this analysis was ever submitted to the Linux community, much less accepted into Linux.

(3) *SCO might be alleging that that "differential profiling" is a method tainted by contact with Dynix/ptx, and that something McKenney, Soni, or Sarma contributed to*

Linux subsequent to the cited emails was derived from the idea of using differential profiling to analyze Linux file system performance. IBM could ask its employees and former employees for the full story of what happened after this email was sent, and then try to prove the negative with respect to this interpretation of the claim. This interpretation would be consistent with SCO's suggestions that IBM knows better than SCO what it did "wrong", and therefore no additional particularity is required. This interpretation of item 146 would demonstrate SCO's disregard for the court's rejection of the idea that it's IBM who knows what the claim really means, not SCO.

Each of the previous interpretations is plausible in light of the fact that SCO's five components of evidence for Item 146 simply do not connect to form a clear allegation. Each of the previous interpretations could be defended, but the expert analysis required for each defense is substantially different. Item 146 is so vague that nothing prevents SCO from moving from one interpretation to the next in the face of IBM's response. IBM is left to deal with all possible interpretations, which would require at least three times the effort and gives SCO at least three times the opportunity to avoid scrutiny of its claim.