EXHIBIT 14

Page 1

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
TYLER DIVISION

BEDROCK COMPUTER)
TECHNOLOGIES LLC
DOCKET NO. 6:09cv269
-vs-)
Tyler, Texas
8:50 a.m.
YAHOO!, INC.) April 29, 2011

TRANSCRIPT OF TRIAL
MORNING SESSION
BEFORE THE HONORABLE LEONARD DAVIS,
UNITED STATES DISTRICT JUDGE

APPEARANCES

FOR THE PLAINTIFF:

MR. DOUGLAS A. CAWLEY
MR. THEODORE STEVENSON, III
MR. SCOTT W. HEJNY
MR. JASON D. CASSADY
McKOOL SMITH
300 Crescent Court, Ste. 500
Dallas, TX 75201

MR. ROBERT M. PARKER
MR. ROBERT CHRISTOPHER BUNT
PARKER, BUNT & AINSWORTH
100 E. Ferguson, Ste. 1114
Tyler, TX 75702

COURT REPORTERS:

MS. JUDY WERLINGER
MS. SHEA SLOAN

Proceedings taken by Machine Stenotype; transcript was produced by a Computer.

Page 14

ANSWER: Yes.

QUESTION: And, again, you said that when you sent e-mails to Mr. Absher, you had no reason to be dishonest; isn't that true?

ANSWER: No. Absolutely.

QUESTION: Now, isn't it true, in that paragraph, sir, you wrote to Mr. Absher stating: My analysis showed that the code written by me does not actually collide with the aforementioned patent. My code uses quite different techniques?

ANSWER: Yes.

QUESTION: And isn't it also true, sir, that the current Linux kernel actually contains logic, which could be considered infringing the patent?

ANSWER: Yes.

QUESTION: Okay. And isn't it also true that you could not find any references describing the idea in the patent before 1999? Isn't that true, sir?

ANSWER: No. It was a mistake.

Actually, I found a lot of references dated back to 1985 about this technique. I just didn't have any references at that point.

QUESTION: Well, sir, in the e-mail that's Exhibit 8, you've already said that you were truthful when you wrote this e-mail, right?

Page 15

ANSWER: Yes, I was truthful. I didn't lie. I just didn't have the information. I got it --

QUESTION: I understand. But in this e-mail, didn't you write: I could not find any references describing the idea before 1999?

ANSWER: I found them quickly after that. To December 15, I have already investigated the case and find all the papers, found who -- I didn't find actually who invented this technique, but I found investigations of analysis of technique dated ten years before that, at least 1985.

QUESTION: Okay. Well, sir, my question is, in this e-mail, isn't it true that you wrote: I could not find any references describing the idea before 1999?

ANSWER: No. It is not true. I was truthful when I wrote it, but I just didn't have that information. So this sentence is not true.

I am truthful, but -- I am truthful, but statement is not true. I didn't lie, but the statement is not true. I just didn't know that it wasn't true at that time when I wrote this.

QUESTION: Now, Mr. Kuznetsov, isn't it true, sir, that the Defendant's position can be difficult to defend, and you believe that they should

Page 16

seek an expert in loopholes of patent rules?

ANSWER: Yes. Yes, I wrote that as well.

QUESTION: Is it a true statement, sir, that the Defendant's position can be difficult to defend, and you believe they should seek an expert in loopholes of patent rules?

ANSWER: No. It was a wrong statement. I thought that it's true when I wrote this. But after I remembered that this code is actually inherited from my code of 1995, I returned my opinion to the opinion which I had a year ago. It is not true.

QUESTION: Yes or no, Mr. Kuznetsov, did you say you should seek an expert in loopholes of patent rules?

ANSWER: Yes.

QUESTION: Okay.

ANSWER: This is not true.

(End of video clip.)

THE COURT: All right. Thank you.

Who will be your next witness?

MR. CAWLEY: Your Honor, at this time, we would call to the stand Mr. David Filo.

THE COURT: All right, Mr. Filo.

MR. CAWLEY: May I proceed, Your Honor?

THE COURT: Yes, you may.

Page 128

behind. It's not -- is not this enough to invalidate the patent?

Q. Exactly. And we will hear the rest of his deposition this afternoon, correct?

A. That's my understanding.

Q. And have you had an opportunity to look at the -- I think Mr. Cawley asked you and you said you had had an opportunity now to look at the -- what we call the old prior art Linux code or the Kuznetsov code or the '95 code?

A. Yes, ma'am.

Q. And do you have a copy in front of you, sir?

A. I do.

Q. And what language is this code written in?

A. This is written in C.

Q. And do you read and program in C?

A. Yes, ma'am.

MS. DOAN: We're on Exhibit 48, and it starts around Page 132, I think, is where the lines are.

Q. (By Ms. Doan) And have you reviewed this old prior art '95 Linux code?

A. Yes, ma'am, I have.

Q. And what version are you looking at, please, sir? Is it 2.0.1?

A. I'm looking at -- yes, 2. -- well, sorry.

Page 129

This is Linux 2.0.1, that's correct.

Q. Okay. And you understand that there's three different versions that we're all talking about, all basically have the same type of code in it, correct?

A. Yes, ma'am. Well, three versions? Sorry.

Q. Right. Of the old Linux code?

A. The old Linux code, yes.

Q. Okay. And does Exhibit No. 48, the old Linux code, have on-the-fly garbage collection with a hashing table and external chaining?

A. Yes, ma'am.

Q. And can you tell us where that is in the prior art?

A. Sure. So -- I'm not sure I have exactly the same thing you have, but let me try.

If you could go to Line -- let's start at Line 1446.

That's not it. I'm going to need -- I don't -- this is a different -- try 1365. I have two different printouts here. Sorry.

Q. Okay. That's fine. 1365?

A. Yes, that's correct.

Q. Okay. Does that match the version we're talking about?

A. Yes, that matches what I have here.

Q. Exhibit 48? Okay.

A. Yes.

Q. And tell us what where -- where in the code, in the 1995 Alexey Kuznetsov code, it has on-the-fly garbage collection with external chaining and a linked list.

A. Okay. Well, this is -- if we could go back previously to the code -- but this is a hash -- this is within a hash table. And Line 1365 represents the beginning of walking the linked list within the hash table.

I think you've seen this structure before, where you have a while loop, and that represents -- you know, this while loop that starts on Line 1365 and ends on Line 1383, that is the code that represents walking the linked list. And, again, I think we've looked at this before in some other examples.

And the idea is you start with the first record and you iterate through. While we're walking through the list here, if you look at Line 1369, we will see that there is a check to see if -- here we see this Cache_TIMEOUT, and basically what this is doing is checking to see if this particular record in the linked list has expired.

We identify that it has expired, and

Page 131

immediately in Lines -- well, in the Lines 1372 through 1378, that is where it's removing the expired record.

And so this is, again, all within the same access of the linked list while we're walking it. It has identified and removed the expired record.

And if you go down further to Line 1382, that's just updating the pointer to continue walking the list.

And then, as I said, as you drop down to 1384, you now have exited the list; and you have completed walking the list.

Q. All right. So that on Line 13 -- 1378 there, it says rt_free(rth). Can you tell us what that means?

A. Rt_free, that is what is removing -- well, it's in combination with 13 -- it's actually a combination of 1372 through 1378. You have to do those multiple operations to do the actual removal. And that's kind of the final step in removing that record.

Q. That removes the record?

A. That's correct.

Q. Okay. So does Exhibit No. 48, the lines we just went over, does that --

A. But it doesn't -- the record was actually removed above that --

Q. Okay.

A. -- in 1372. What that does is actually free the memory. As we've talked about before, once the record is kind of deleted from the list, it is now rt_free on Line 1378, is what is returning the record to the operating system to be used for something else.

Q. I see. So what is the line that actually removes the record from the external chain?

A. Actually, it's 1372, which is what changes the pointer and skips over and is effectively taking that record out of the list.

Q. Okay. So does Exhibit No. 48 -- Defendant's Exhibit No. 48 describe on-the-fly garbage collection with external chaining in a linked list?

A. Yes, ma'am, it does.

Q. Does it also have the automatic removal of expired records?

A. Yes, ma'am. I talked -- just talked about that. It identifies the records and removes them while it's walking the list.

Q. And this code in Exhibit No. 48 was available in 1995 and 1996, approximately one to two years before the '120 patent was even applied for, correct?

A. Yes, ma'am.

Q. I think you talked about this a little bit earlier. Yahoo! had Linux in late '95 or early '96?

Page 133

A. That's correct.

Q. Does the 1995 -- does DX Exhibit No. 48 invalidate the '120 patent?

A. I believe it does.

Q. And, of course, you read the patent?

A. Right.

Q. And you studied it since your deposition --

A. Yes, ma'am.

Q. -- to be able to come and talk to us here today about it?

A. Yes, ma'am.

Q. And you've reviewed other patents in the past?

A. I have.

Q. And you've reviewed Judge Davis' claim construction?

A. Yes, I have.

Q. And you are applying the terms as Judge Davis has construed them in this patent?

A. Yes, ma'am.

Q. Thank you, sir.

Have you also reviewed the NRL code?

A. I have.

MS. DOAN: Casey, I believe that is Exhibit 215 -- oh, 37. I'm sorry. 37.

Q. (By Ms. Doan) Do you have that in front of

Page 134

you, sir?

A. Yes, ma'am. I hope these lines match up.

Q. Well, I have NRL Code No. 37, and it's dated 9/28/1995 in the upper right-hand corner.

Is that what you have?

A. I have the file before that.

Q. The first page?

A. But this actually looks a little bit different, but, again, I think the line numbers will match up.

Q. Okay. You've got key.c; is that right?

A. Yes, ma'am.

Q. Okay. So the lines should match up?

A. They should.

So if you go to Line, I guess, 1332, to see if it matches.

It does not match. Sorry.

Q. That's okay. Let me give you my copy of Exhibit No. 37.

A. Okay. It would be Line 1397.

Okay. Sorry. We're close.

All right. So that says key acquire. That's just the -- that's the function I guess that I'll talk about first. Let's go to --

Q. Are --

Page 135

A. Sorry.

Q. That's all right. Go ahead.

A. This code is a little harder to read. It's got a lot of kind of debugging, slash -- debugging information that kind of confuses things, but...

Q. What is debugging information?

A. It's information that the computer prints out to explain what's happening, for humans to read. And so instead of just doing kind of its work to run the computer, it's also printing this stuff out.

Q. Is this also written in the language or the computer language C?

A. Yes, it is.

Q. And, of course, you read and write in C?

A. Yes, ma'am.

Q. Now, I think we covered this earlier, but Exhibit No. 37 is the key.c file to the NRL code; is that right?

A. Yes, ma'am.

Q. Okay. And where are we -- what's happening on Line 1397?

A. Well, that's just the start of it. I wanted to check to see if it was same.

If you go down to Line 1431, this represents the -- where you see the word "for," unfortunately, this

Page 136

is a little different than what we've seen in the past. When we've looked at walking a linked list, we've seen the word "while."

This "for" is very similar to that, almost equivalent. It's got some other stuff in there, but, effectively -- I don't want to go into too much detail here, I think, but the for loop -- it's called a for loop instead of the while loop, and it's very similar to that.

And if you go down to -- you know, that for loop extends from 1431 down to 1459.

Q. 1459?

A. Yes, ma'am.

So that represents the loop that is walking the list.

Q. Okay.

A. So, again, this represents a linked list and it is -- this code is walking that list, and we can see at the top, it says for. And it says ap = key acquirelist next; ap; ap = ap next). The ap = ap next on 1431, that represents moving the pointer to the next record.

Q. Okay.

A. If you look at Line -- and I don't want to go through all this code, but if you look at Line 1445, we

Page 137

look for a condition that checks to see, in this case, has the record expired. And you can kind of see ap expiretime is less than time.tv_sec.

This is identifying expired records while it's walking the list, and then what it does with that -- in fact, if you read the comments -- and, again, this is not the computer code but it's comments, so it may not necessarily match. But it says since we're already looking at this list, we may as well delete expired entries as we scan through the list.

And if you look down at Line 1454 and 145 -- well, 1454 removes the record from the list. And then we have a similar free, as we had before with the rt_free in 1455, that frees up the memory to give it back to the operating system to do something else.

Q. So Lines 1431 through 1459 are the part that's the on-the-fly garbage collection while walking a linked list; is that right?

A. That's correct. So this is a linked -- yes.

Q. Okay. And is there another part of this same file, key.c in the NRL code, that talks about the hash table with external chaining?

A. Yes, ma'am. It's going to be -- I'm going to have to find it again, but around 615. Let's see if it's close. Oh, 6 -- 6 -- 649.

Page 138

Q. Okay.

A. So there are many examples in this file that look at hash table or that -- this file is full of routines and stuff that work on a hash table with external chaining.

But I'll take you to Line 675, and the code says prevnode = &keytable [indx]. And what that represents, the key table is the hash table. The index is -- in this case, it's being passed into the code. That's the hash value.

And this next part of the code is, again, walking -- is walking the linked list.

Q. Okay.

A. So this part of it -- sorry. So 676 is, again, this for loop construct that I talked about.

And, again, you can see where it walks the list by starting at the front, which is the keynode = keytable [indx].next. And then it advances the pointer to go to the next element by -- at the -- the last part of that which says keynode = keynode.next), it starts on 676. It ends on 685. This code is a little --

Q. We can tell that from sort of the closed bracket or closed paren?

A. Well, actually, so -- yeah, it stops at 682.

Q. 682?

Page 139

A. And the point of this isn't so much to talk about walking the list, but it's just to show that this is a hash table, and the hash table has -- each element of the hash table is an external chain or linked list.

Q. Okay.

A. So this represents a hash table with external chaining.

Q. So Lines 675 to 682 of Exhibit No. 37 describes a hash table with external chaining; is that correct?

A. Yes, ma'am, although you have to kind of go -- the keytable -- I think that's -- that's accurate, yes, ma'am.

Q. All right. So within the key.c of the NRL code, you had one part of the code that talked about a hash table with external chaining, and another part of the card -- code that talked about on-the-fly garbage collection with a linked list; is that right?

A. Yes, ma'am.

Q. And combined these two references, these two lines, sections of lines that you've talked about from the NRL code, do they invalidate the '120 patent?

A. I believe they do.

Q. And would these have been -- the hash table with external chaining and linked list within on-the-fly

Page 140

garbage collection, would that have been something to -- that was well-known that you could interchange within the -- a person of ordinary skill in the art back in 1995?

A. I believe so. And we've talked earlier about the -- the hash tables with external chaining that's been well-known back to the '70s or '60s, even further back.

And the part that we looked at earlier which walked the list, identified the expired entries and removed them, that represents walking the list and automatically expiring some of the -- or automatically removing some of the expired records. So those two concepts are in this file.

I think the concept of walking the list, this represents -- I don't know how far back that goes. I wouldn't suggest it goes back to the '60s, but this is an example in 1995 that represents that concept. So those two concepts together, my belief is, invalidates the patent.

Q. Okay. And combined together would be under the theory of obviousness, right?

A. Yes. I mean, external -- external chaining in a hash table or, again, a linked list, that linked list, if there's some way to operate on a linked list that's

Page 141

found -- discovered elsewhere, that's obvious to apply that to any use of a linked list.

In this case, the linked list happens to be in the hash table. Linked list could be used for lots of different things. They could be used standalone. They could be used in hash tables. They could be used in other data structures.

So taking the capability of automatically expiring, automatically removing expired entries in a linked list and combining that with the hash table is, to me, very obvious.

Q. Okay. And does the NRL code inval -- invalidate the '120 patent, in your opinion?

A. I believe it does.

Q. All right. And, of course, you know that Dan McDonald will be testifying later on in this case about this to explain the e-mail that Mr. Cawley was talking about?

A. Yes, ma'am.

Q. Now, if you would review the actual accused code in this case, 2.6.9 and 2.6.18?

A. I have.

Q. And you're aware that the reason we focus on those two is because the majority of the 196,000 servers with the accused Linux candidate code are in these two versions; is that right?

A. Yes, ma'am.

Q. So, for example, he -- well, I don't know where it went anymore.

So, for example, there are two versions of generation ID code. Do you recall that, that Dr. Jones went through yesterday?

A. Yes, ma'am.

Q. But there's only one offline server at Yahoo! with each of those two versions, right?

A. There used to be.

Q. Okay. So the vast majority of the servers we're talking about is in 2.6.9 and 2.6.18, right?

A. Yes.

Q. All right. Now, let's look at 2.6.9, and I believe that's Exhibit No. DX -- Defendant's Exhibit No. 74.

A. Yes, ma'am. I have that one this time. That's good.

Q. And does 2.6., this Linux candidate code version, identify a record in the same access of a linked list?

A. No, it does not.

Q. How do you know that?

A. Well, looking at the code -- and we have --

you guys, you have looked at this code before. So if we go down to Line --

Q. I think they have a copy of the code, the 2.6.9.

A. Okay.

Q. And is that --

MS. DOAN: Is that the green version or the yellow version? Who's got our copy?

Q. (By Ms. Doan) It says 2.6.9 at the top? 2.6.9?

A. Yes. Sorry. Yes, this is -- again, I have the Exhibit 74.

And so at Line 776 --

Q. 776?

A. -- that's the rt_intern_hash function that we have been talking about.

And if we go down to Line 795, I think you will be familiar with the while loop that begins walking the linked list. And I'll skip over the rest of it, because I think you are somewhat familiar with it.

But 795 begins walking the list and the end of that list is -- sorry -- the end of that while loop is on 836.

Q. Okay.

A. And, again, we've looked at this, but this is

Page 144

the while loop that walks through the list, each entry; and while it's walking through the list, one of the things it does is it scores each -- each record.

Q. How do we know that? Where are those lines?

A. Yes. So that's on Line 826 through 829.

Q. 826 through 829?

A. Sorry. 824 through 829.

Q. Okay.

A. The 824 line is actually computing the score, and then we're keeping track of the lowest score in 826 through 829.

So this is walking the list and is identifying records or -- it's identifying candidates.

And on 835, that's near the end of the while loop. That's updating the pointer, which goes to the next record. And, again, the while loop that begins Line 795 is completed on 836.

So by the time we get to Line 838, we have completed walking the list. We have gone through every single element and analyzed every single element, and we have effectively walked off the end of the list. And we have completed the access of that list.

Q. So that would be between -- the first access of the list is between 795 and --

A. 837.

Page 145

Q. -- 837?

A. Yes.

Q. Okay. And what happens in that first access between 795 and 837?

A. Well, there are a couple of things. It looks for a match, but assuming there's no match, it's doing -- it's doing the scoring for the candidates and keeping track of the lowest score.

Q. Are there any records that are removed in the Linux candidate code 2.6.9 in the first access -- or the first walking all the way down the list from 795 to 837?

A. No, ma'am. As in the earlier example, we saw where it actually removed it while it was walking the list. In this example, there is no removal while it's walking the list.

Q. Where does the removal take place?

A. It takes place after the first access, and you see the lines 838 down to -- call it 847 where it's checking to see if there's a candidate and to see how long the linked list is. I guess -- and you guys, again, have looked at this code and are possibly familiar with it.

And on Lines 846 and 847 -- well, 846 is what takes it out of the list, and 847 is what's freeing the memory to go back to the operating system.

Page 146

Q. So Lines 838 to 847 would be the second access?

A. Yes, ma'am. That's going back and accessing the list a second time. Again, the first list was completed at 837.

Q. And you understand that Judge Davis has instructed us that the '120 patent would require, when the linked list is accessed, both identification and removal of expired records occurs during the same access of a linked list; is that right?

A. Yes, ma'am.

Q. And do both identification and removal of the record happen in the same access in the 2.6.9 code?

A. No, ma'am.

Q. Now, we heard something yesterday about like a spin_lock, spin_lock from Dr. Jones.

Do you recall that?

A. Yes, ma'am.

Q. Okay. So does the existence of a spin_lock or spin_unlock in the Linux candidate code determine the access?

A. No, ma'am. And again, there's no lock -- there's no mention of a lock in the patent, and so, you know, locks are -- locks are fairly arbitrary in where they get placed in computer code. It can vary a lot,

Page 147

depending on what type of lock you use and that type of thing.

And so I think, again, as has been talked about, there is no mention of the word lock in the patent. So I don't see how that would -- would apply to -- to considering where the access is.

Q. Okay. And as a technician, as you are, why would a lock not define the access? Why would a lock matter or not?

A. Why would a lock matter or not?

Well, again, there are many different ways to do locking, and you could -- you could lock huge parts of the operating system and do many, many operations while you have things locked.

And, you know, there are things like giant locks in kernels that lock the system for very long periods of time, do many, many operations. And to consider all that stuff to be a single access just is kind of a silly idea.

Q. All right. And if you look at Exhibit No. -- it's DX77, that's the 2.6.18 Route.c?

A. Yes, ma'am.

Q. 77. That's another version of the candidate code?

A. Yes, ma'am.

Q. Are -- I know the line numbers are different, but essentially is it the same thing, walking the --

A. Yeah. As has been talked about earlier, the differences between 2.6.9 and 2.6.18 are mostly for the purposes of -- these discussions are cosmetic. There are differences in the actual code, but not important to walking the list, identifying records, removing them, and et cetera.

Q. Sure. So we all have our copies of the green code.

A. Oh, sorry.

Q. Can you just take us and tell us which lines --

A. Yes. Right.

Q. -- are the first access and the second?

A. So Line 937 --

Q. Okay.

A. Oh, no. Sorry. I'm sorry. Line 934.

Q. Hold on a second. Mr. Morisseau just pointed out to me, I think the green code is actually a version of 2.6.27. It's not 2.6.18 that we're talking about here.

So do we have an example of 2.6.18 to show the jury?

MR. MORISSEAU: The jurors do not have

Page 149

that.

MS. DOAN: They don't have that? Okay.

THE WITNESS: We can show it on the screen.

MS. DOAN: We can show it on the screen.

Sorry about that. That's fine. Sorry about that.

A. I think I said 934 -- or Line 934. So that's exactly what we just looked at. It's the while loop. Starts at 934, and this while loop ends on 9 -- or 7 -- sorry -- 979.

Q. (By Ms. Doan) Okay.

A. And then --

Q. And that was the first access?

A. That's the first access. That's walking the list, looking at every single record in the list. You see the scoring on 967.

Q. I tell you what. Hold on one second.

MS. DOAN: Judge Davis, we have a copy of the 2.6.18 code. Can we pass them out to the jury real quick so they can follow along?

Judge Davis, we actually have copies of this code. Can we pass it out to the jury real quick?

THE COURT: That's fine.

(Pause.)

Page 150

Q. (By Ms. Doan) All right. This is Defendants' Exhibit 77. And I'm sorry. I think you were on Line 934 to 979?

A. Yes. 934, which, again, is the start of the while loop. And that while loop goes from 934 down to 979.

Q. How do you know that the first access ends there as opposed to what Dr. Jones was telling us yesterday?

A. Well, again, at 979 -- or sorry -- 980, which is the first time you've -- you know, have completed the while loop -- 980 isn't really any code, but it's kind of blank. And so the next statement to execute would be 981.

But at that point, when you've completed the while loop, you have gone through every single record, and you have analyzed every single record, and you have walked the entire list, and you've effectively -- like as I said, you've walked off the end of the list.

Q. So you're at the very end of the list?

A. And you looked at the last pointer, and it was empty, and so now you're -- have nowhere to go, so you're coming off the list.

Q. So once you go all the way -- that was -- when Dr. Jones talked about yesterday with the sticky notes,

Page 151

it was all the way to the eight records at the end of the linked list?

A. And you go to the ninth record and you realize there is none, and that's when you stop the loop.

Q. You have to go back into the --

A. That's the end of that access.

Q. Okay. And so where does the second access take place?

A. Again, this is very similar to the previous code we just talked about, and that's Line 981 where it checks to see if there are any candidates. And if there are and the list is longer than eight or whatever the elasticity is set to, it removes the record from the list in Line 989, and then it frees and returns to the operating system, the memory associated with that, on Line 990.

Q. And again, with respect to 2.6.18, is your same analysis, with respect to -- does it matter about the spin_lock/unlock?

A. It's the same.

Q. And does the 2.6.18 remove expired records as required by Judge Davis in his claim construction?

A. I don't believe they do -- it does.

Q. Okay. And why not?

A. Well, so if you look at -- go back up to Line

Page 152

967 and look at that -- look down through 973, call it --

Q. 973?

A. Well, just do a --

THE WITNESS: Yeah, there you go.

Well, do it in one highlight. That's confusing. Can you just do one highlight instead of two?

Q. (By Ms. Doan) Yeah. So from 9 -- give me those -- the spread of those lines. From 966 to --

THE WITNESS: Just do one highlight that's 966 -- or no -- sorry. Yeah, 967 -- okay.

That's fine. But it doesn't line up, so if you could just kill the highlights and do -- kill the other one, and do 967 through whatever I said, 9 -- or 972.

MS. DOAN: Can you do that if it's on different pages? He has two pages, so he can't really do that.

THE WITNESS: Oh, sorry. Sorry. I didn't understand that.

MS. DOAN: That's okay.

THE WITNESS: Okay. All right. So it's not going to quite line up, but...

A. All right. Okay. So this -- and Lines 967,

Page 153

this is computing the score for that record in the list. And this score is -- there's not a score about whether the record is obsolete; this is a score -- has a bunch of different factors that go into it, but it's simply a score. And then we keep track of the lowest score.

And so in some sense, this is like pulling straws; and because the score that's lowest is going to get kicked out, it's kind of pulling straws, and the one with -- whoever ends -- the record with the smallest straw is going to get kicked out.

Now, it has nothing to do with whether the record is expired or obsolete. These records that are identified are all valid data; they are useful in the future; and they could be used to do the -- you know, the route lookup if it was there.

So there's nothing obsolete about these entries. And again, we're just kind of picking one somewhat at random, which one to pull out, and these are all useful data and not obsolete or expired.

Q. So by applying Judge Davis's claim construction of the word expired, which says obsolete and, therefore, no longer needed or desired in the storage system because of some condition, event, or period of time, the 2.6.18 and 2.6.9 Linux code do not meet that definition of expired record?

Page 154

A. Yes. We've looked at the 2.6.18, but the same -- the same reasoning applies to 2.6.9, which that code is -- I believe it's identical character for character.

Q. In your opinion, does the Linux candidate code -- sorry.

In your opinion, does the Linux candidate code infringe the '120 patent?

A. I believe it does not.

Q. And whether it's Version 2.6.18 or 2.6.9 or any of the other versions that we covered yesterday with Dr. Jones?

A. Anything with the candidate code, I believe, does not.

Q. Okay. You haven't looked at the generation ID code?

A. I haven't looked at that.

Q. Why not?

A. I saw it for the first time -- we have -- sorry. We had two servers running that code. These servers had never performed a single function for the Yahoo! website. They were in a test lab. Once we realized that they were running generation ID code, we immediately wiped those servers clean.

So we no longer have those. We had them. I didn't think it was important to look at that, given that the other 196,338 servers or whatever have the older 2.6.9 and 2.6.18, so that's what I focused on.

Q. Okay. Thank you, sir. I want to switch gears for a little bit --

A. Sure.

Q. -- and talk about now denial of service attacks, because I know that's something that Mr. Cawley covered with you as well.

A. Yes, ma'am.

Q. Has Yahoo!'s system ever been a target for a denial of service attack?

A. Yes.

Q. And was Yahoo! a target for denial of service attacks in 2005 before the candidate code was ever written?

A. Yes, ma'am.

Q. And was Yahoo! a target for denial of service attacks after the candidate code in the Linux operating system?

A. Yes, ma'am.

Q. When you say we're a target for a denial of service attack, what do you mean by that?

A. Target just means, you know, we -- there are a lot of people out there trying to do malicious things,

Page 156

and we have a lot of services we offer, and we serve a lot of people, and we are one of the biggest targets on the web. And because there are a number of people that are out there doing malicious things, we end up being a target for many of them.

Q. I had one other question about this code. I'm so sorry. Do you mind going back to that in 2677 -- Exhibit No. 277 (sic)?

A. Okay.

Q. Because I want to make sure -- we're all talking about so many different lines of code. The 2.6.18 code, that's the entire Route.c; that's the entire route cache, correct?

And if you'll look at the very front page, it says at the top front line --

A. Yes, ma'am.

Q. -- jEdit - Source Code_2.6.18_route.c?

A. Yes.

Q. Okay. That's the entire route cache?

A. I haven't verified that this exhibit represents every single line of that file, but the route cache is in that file.

Q. There's basically -- I think the copy we have has about 3178 lines?

A. I think mine has -- mine has a few more than

Page 157

that, but --

Q. Actually, it does. It's got 3214, right?

A. 3214. Yeah.

Q. 3214 --

A. 3214. I mean, I haven't -- I haven't actually -- I mean, I have downloaded the source code, but I haven't compared it to this to see if they're actually -- this printout matches what you can download.

Q. Okay. And basically, Bedrock is talking about 41 of these lines in 2.6.18, correct?

A. My understanding is it's about 40. I haven't counted it personally.

Q. Okay. And you mentioned that you had downloaded this code recently?

A. Yes, ma'am.

Q. Why did you do that?

A. Well, for purposes of analysis. I downloaded both the 2.6.9 Linux kernel and the 2.6.8 Linux kernel.

Q. And can any of us do that in the courtroom?

A. Yes, ma'am. It's readily available. Many places you can go, but kind of -- source of truth or the -- kind of the main place to go would be kernel.org.

Q. And we're showing a webpage up here from www.kernel.org?

A. Correct. And this has -- you can pretty much