Operating Systems

Description: After catching up on the week's important security news, Steve and Leo continue their tour of the fundamentals of computer technology by looking at the history and present day features of modern operating systems.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-250.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-250-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 250, recorded May 26, 2010: Operating Systems.

It's time for Security Now!, the show that covers everything you need to know about your security and privacy online. And who better to lecture on this subject than Professor Steve Gibson, the king of security. He's from GRC.com, the Gibson Research Corporation, the creator of SpinRite, the world's finest hard drive and maintenance utility. I've given you the title. Steve looks perplexed.

Steve Gibson: Well, it's the word "lecture," Leo.

Leo: Not exactly.

Steve: I'm not, you know, lecture, I've got a kind of a dry, boring connotation. I wouldn't say that the episode known as "The Portable Dog Killer" would...

Leo: Wasn't a lecture. That was a...

Steve: No, that would not be a lecture.

Leo: ...story worthy of Garrison Keillor, that was.

Steve: That was more of an adventure, I think.

Leo: Lecture's not - but, you know, there are some lectures that are exciting and engaging. I mean, a good lecturer is...

Steve: Discussion. I like discussion.

Leo: Discussion, okay.

Steve: More interactive, more participatory, more listener involved. So whatever it is.

Leo: Well, and we've been having a great series of discussions on building a computer from first principles, from the ground up.

Steve: Yup, today we're going to talk about operating systems, the history of them and also what they are doing for us in contemporary machines. So sort of essentially we've, as you said, started at first principles. We've been very carefully layering concepts on top of previous concepts, building an understanding. We've talked about most recently the multiverse, the notion of multitasking, multithreading, multicore and so forth. Well, it's the operating system which is really the final piece of this, which is the overseer, well, it used to be called the "monitor" at one stage in the evolution of computers. You had the monitor that was essentially there governing what goes on. Of course we have essentially that today with user programs running atop this operating system.

So today we're going to plow into the long-term evolution, and then also what are all the features of contemporary operating systems, what do they do for the programs that run in them or on them or around them and through them and so forth. And of course we've got security news, updates, a bunch of errata that I think people will find fun, and an interesting short little SpinRite testimonial sent by a listener. And then all the OS stuff.

Leo: As usual, a jam-packed program full of - chockful of goodness.

Steve: Recharge your iPods.

Leo: Yes. You don't want the battery to run out in the middle. I love these building up from the basic principles. I just think this is just fascinating.

Steve: And we're going to do that again. As we have discussed, the next major series will be starting from the bits, building up the Internet.

Leo: Wow. Wow. So when do we stop? I mean, when are you done? How do you know when you're done? I guess you'll be done when you get to mobile phones? I mean, when are you done?

Steve: Oh, you mean like with networking series stuff.

Leo: No, just in general, like building a computer.

Steve: Well, my feeling is that this episode, operating systems...

Leo: Is going to finish it up?

Steve: It wraps us up. We've gone from - what I mostly wanted to get people to understand is that there isn't anything frightening or scary or magic. I mean essentially that the way our computers work today is 100 percent knowable. And we started by - we started at the beginning by saying, look, all you have to have is a blob of memory and something called a program counter that has a current value pointing into this blob of memory, and that the individual bits in the word determine what happens next. And basically we've built everything up from that. And I know that our listeners have gotten a kick out of it. So we've had a lot of great feedback for it.

Leo: I had mentioned before this book, and I thought I'd mention it as we come to the end of this, again, if people are interested in this notion: "The Elements of Computing Systems: Building a Modern Computer from First Principles." And I got this because I thought this might be a fun thing to do with a class at some point in school. And it's the same, kind of the same idea that you've been doing. A little less - you're very practical. But this is talking about, you know, there's the assembler, there's machine language, there's sequential logic, Boolean arithmetic, Boolean logic. You have to understand all that. And this is a lot of the stuff that you've kind of covered. So, and here's the last chapter, operating system.

Steve: Well, and the whole approach we take on this podcast is one of boldly going where no podcast has gone before. I mean, we've dealt with cryptography and said, look, I'm going to explain to you exactly how the AES cipher works. Here's how it works. And we come away saying, oh, that's all? That's all it is? Now I understand it. And, hopefully, listeners of our computer series now get what it is, what machine language is and so forth. So, yeah, this is when you just say, okay, this isn't frightening. This isn't hard to understand or impossible. It's like, okay, let's understand it.

Leo: All right, Steve. Shall we start with the usual security updates?

Steve: Yeah, we have just a couple things because in recent weeks we've had such catastrophe with Adobe and...

Leo: Oh, it's been terrible, yeah.

Steve: ...Windows and everything. So as a consequence there just isn't that much. I did note, however, that - and I've been meaning to bring this up a couple times, and I finally wrote it down and remembered. There's an extremely popular media viewer, mostly I think used for visual, you know, JPGs, PNGs and so forth, called IrfanView.

Leo: Oh, I use it.

Steve: Yes, I-r-f-a-n-V-i-e-w. The good news is he named it after his - with his first name, Irfan, rather than his last name, which I cannot pronounce. But so the...

Leo: Even "Irfan" confuses people, I have to say.

Steve: Yes, the Irfan is at least pronounceable. He's now at 4.27, and I noted that I was at 4.01, I think. The reason I bring it up is there have been some subtle security problems popping up recently. And there's no automatic update. You can't even go to the app and ask it to update itself. You have got to go do this manually. So I would imagine people who've been using IrfanView for some time probably have older ones. I don't think this is a big emergency. It's not a high-profile attack surface. But it's worth bringing yourself up to date.

Probably the best way to do it, there's two different files anyone needs who wants to do this. One is the IrfanView EXE itself, the installer. And it does upgrade very nicely over its prior version. You don't have to uninstall or go through a bunch of hoops. And the other is a plug-ins because it accepts plug-ins for just a phenomenal number of different media formats that it's able to handle. You can go to the home site of IrfanView, but he doesn't have any downloads from there. He's got a page full of links to other places.

The best other place, I think, is just Software.com. So if you to go www.software.com and then search for IrfanView, that turns up just those two files. And in fact that's the - Software.com is his preferred provider for the plug-in file, which is how I stumbled on this. And I saw that he also has the latest version of the regular main program setup file. So go to Software.com, search on the site for IrfanView, you'll find the two files to download. And you just run the main installer first and then the plug-in installer, and you're set. And that will - this latest version, 4.27, as of this podcast, does fix those small, subtle, known potential security problems in this particular extremely popular viewer. Apparently he's in aggregate seeing more than a million downloads a month. So this thing is widespread.

Leo: You know, it's interesting because you wouldn't think something like that, something so simple as a graphics file viewer could be dangerous. But we're learning that that's one way you can make a data file dangerous is if there's a hole in the viewer.

Steve: Yes, exactly. Exactly.

Leo: And a malicious data file is a particularly nasty beast.

Steve: Well, and these large images are now physically large. So they're able to contain a lot of code.

Leo: Hmm, interesting, yeah.

Steve: Yeah. So it's not - they're not like they're little 5K things. You look at the file from a high-resolution, eight-megapixel camera, even compressed it's a couple meg, and that could be a couple meg of malicious code. And you think, oh, it's a picture, that can't hurt me. Previously we've been trained to believe that executable files were the big problem. What we of course learned is, and we've discussed this many times, is that any sort of problem, even in the rendering of an image, can allow someone who's clever and malicious to get your computer to execute the image rather than display the image, and that executable code can be malicious. So definitely something to worry about.

Again, not a big emergency, but why not do it because this thing isn't - IrfanView is not updating itself automatically. Thus I would imagine lots, I mean, it happens to be my viewer. I've got it on my system. Now I've got 4.27. So our listeners should, sooner or later, if they're users of IrfanView, do that. The only other problem that I ran across that was of any interest really, there's all kinds of obscure little nothing problems. But there's a very popular download manager that's just called Free Download Manager, which is used often for downloading - it's used as an app - oh, oh, one thing I forgot is that...

Leo: Oh, oh, oh.

Steve: [Laughing] I was a little annoyed. IrfanView by default wants to install the Google Toolbar.

Leo: Oh, yeah.

Steve: So, I mean, like that's not a bad thing...

Leo: But I hate it when they...

Steve: I do, too. But I guess he's got to...

Leo: It's how he makes money. Because it's a free, it's a totally free program. And Google doesn't pay him money. But if you do searches through that Google Toolbar, he gets a commission on the ad sales.

Steve: So if you are a Google Toolbar user...

Leo: It's good to support him, you know?

Steve: Yeah, exactly. But I don't - I'm not a person who likes to install superfluous things for all the reasons we've discussed over the last 249 episodes. So I did note it's enabled by default, and so you'll want to disable that and not just click "Next" in the

installing dialogue too quickly. And all I had to say about Free Download Manager is it also doesn't update itself. There is a known publicly exploitable problem with it. So you may know, if you're using the Free Download Manager to download both HTTP and BitTorrent files - apparently it's a popular BitTorrent downloader, too. If you go to their site, they're not very good about managing version number stuff. But the one that's there has fixed the problems. So if you do know that you use the Free Download Manager - I don't, but I ran across this little blurb - you ought to update yourself.

In security news, I noted sadly that the world seems to just have gone nuts over Google's WiFi mistake. And I'm sorry about that. I mean, I really do think it was a mistake, as we discussed in detail last week. I don't think there's anything malicious. I think the lesson here, I mean, and the only reason I'm glad it's getting attention, except that it's getting the wrong kind of attention, is the world needs to wake up to what open WiFi, unencrypted WiFi means. And Google wouldn't have gathered any data, wouldn't have been able to, because they certainly weren't decrypting it, they wouldn't have been - they were just sucking in the packets that were being broadcast by radio in the air, and never had any intention of doing anything with it. And I'm no Google fanboy or apologist for them. I just think this is a problem with unencrypted WiFi. And I wish this was getting the kind of attention I think it should. Instead it's getting, I think, really the wrong kind of attention.

Now - first was Germany, as we discussed last week. Now the U.S. Federal Trade Commission, the FTC, is getting into the act and going to be "opening up an investigation," as are France and Italy. Ireland simply asked Google to delete the data, thank you very much, we don't care about this, just please delete it. The U.K. had asked the same thing, but there are some groups there that are saying, no, no, don't delete it, we want to inspect it. We want to do some sort of inspection. And now a woman in Oregon, Vicki Van Valin, and some guy in Washington have teamed up and filed a class-action lawsuit.

Leo: What a surprise.

Steve: Uh-huh, exactly.

Leo: Any excuse.

Steve: Uh-huh. Now, the good news is, in the U.S. at least it's necessary to prove tangible damages, not just to say, oh, I had open WiFi, and Google drove by. Sorry, being annoyed doesn't qualify in the U.S. So I think this will go nowhere. And I hope whatever attorney is probably drooling at the prospect of going after Google with some class action ends up spending time and getting nothing. I mean, Google's going to fight this, and I think they should. So I'm sorry to see that this has taken the direction that it has. But...

Leo: I think that's to be expected. Nowadays you can't do anything without getting sued; you know?

Steve: Yeah. Yeah.

Leo: And Google I'm sure has a few lawyers somewhere in that big building.

Steve: I think they've got all the money they need to defend themselves. I'm not saying they're by any means defenseless. It's not like the MPAA suing some mom...

Leo: Exactly.

Steve: ...because she had some video on her machine by mistake.

Leo: Right, right.

Steve: So anyway, this, I'm sorry that this has happened. I wish that the lesson was to everyone, I mean, I wish that people hearing this who are not listeners already of Security Now!, to whom I know I'm preaching to the choir, encrypt your WiFi if you don't want people to be able to easily listen in on what's going on.

Leo: Right, right.

Steve: And Microsoft, in one other last piece of news, or actually second to last, Microsoft has confirmed a vulnerability in 64-bit Windows 7 involving what they call the Canonical Display Driver, the CDD, which allows potentially - it will crash Windows, and they're concerned that it could be engineered into a remote code execution vulnerability. There's a problem with the driver's parsing of information as it's passed from the user space into the kernel. We'll be talking about all of what that means in today's operating system episode. It only involves the Aero interface. So if you turn Aero off, if you disable Aero - and actually Microsoft is recommending that Aero be disabled because they have no fix for this currently. It's not in the wild. It's not a zero day. No one has exploited it yet. But I just did want to bring it up for anybody who should know that the 64-bit version of Windows 7 has this problem. And so disabling Aero until Microsoft has a patch for it is what Microsoft is recommending.

Lastly, I ran across a very nice facility which I tweeted about, and actually several times now my tweets have been in the top tweets on Twitter. Which other...

Leo: Did you ever think you'd be saying that, Steve?

Steve: No, no, no. And in fact, Leo, if you go to GRC.com, look at the top of the GRC.com page. You'll see three icons you never expected to see.

Leo: Oh, Blogger, Twitter, and RSS. I'm liking it.

Steve: Yeah.

Leo: I'm liking it.

Steve: We've been sucked up by the dark side.

Leo: So, wow, well, I'll find out about your blog in a second. But that's really good. That's great.

Steve: What I wanted to tell people about is a nice facility that was created just sort of as a benefit to Facebook users called ReclaimPrivacy.org. ReclaimPrivacy.org. And I know from the feedback that I received that even Security Now! listeners who are also Facebook listeners, who thought they understood the amazingly convoluted and complex privacy settings at Facebook, were surprised. What this ReclaimPrivacy.org offers is the ability to check your current Facebook privacy settings to make sure they're doing what you expect. And in several cases, people who thought they had understood what they were asking for discovered, thanks to this script that checks settings, that in fact they had missed a couple things.

Leo: Well, you know, as we've been recording Facebook has been having a very last-minute special press conference. Mark Zuckerberg, speaking to the press about this privacy issue, admitted mistakes, said we weren't clear. They explained, I think fairly coherently, what they had done, why they had done it. And they are going to change the privacy settings. Their new model will be a lot simpler. This is a screenshot from The New York Times of - it's not up yet, the new privacy settings. And you can see there'll be a big button that you can push. So you can say Sharing on Facebook and click Friends only, and really shut everything down. As far as we know. I mean, we have to wait and see what it really does. You can also turn off the sharing with applications and websites, the Yelp, Pandora stuff that they were talking about. So it looks like Facebook has responded with something simpler. But of course this is just a screenshot, and it's not live yet. And meanwhile, I think this ReclaimPrivacy.org is a very good thing to do, a very good solution.

Steve: Right. So they'll be making it simpler, hopefully soon. In the meantime, ReclaimPrivacy.org.

Leo: And my Facebook page as of today is gone. All my Facebook stuff is gone. And, you know, I don't miss it. It's funny. You're becoming more social; I'm becoming more antisocial.

Steve: Well, I did read an interesting page, and I wish I could find it again. And I've looked for it, and I wasn't able to find it again. Someone who was making the point early in this Facebook brouhaha that the Internet was social. I mean, the Internet was lots of pages, all linked together. And essentially I think they were making the point that Facebook was deliberately trying to create an enclosure, which - and one of your complaints, Leo, was that you had to be a member in order to participate at all.

Leo: Right.

Steve: That is, you weren't able to just look at people's Facebook pages. You had to join, create one, and then you were part of the inner sanctum of 400 million people who have done this. And I thought that was sort of an interesting point, that it's like, wait a minute, what do we need Facebook for? I think the concept of easily user-created personal web spaces makes a lot of sense.

Leo: Exactly. You've got a blog now. Far better for you to control your blog. There's no question of them censoring you. Facebook does delete posts that they determine to be spam, or for one reason or another they believe - they've censored posts in the past.

Steve: Well, and then something you wrote or said recently, wasn't there something that was just deleted that was a surprise to you?

Leo: Well, yeah. I on the radio show on Sunday was talking about this.

Steve: That's what it was.

Leo: Yeah. One of our affiliate stations, KNOI in Texas...

Steve: That's what it was, yes.

Leo: The general manager there sent me an email saying, help, they've deleted our pages. And they haven't explained why, which they don't do. They don't say why. And he believed it was because there were links on the page to me deleting my Facebook, the video, and discussion of Facebook privacy and so forth. Now, we won't know, really. You can't know what Facebook's intent was. Facebook did respond, and we're going to have both them - Elliot Schrage has offered to be on the radio show, from Facebook. And we'll get the GM of the radio station on, as well.

But Facebook says, no, he was a spammer. And this was their rationale, which I found very far-fetched. Over the period of a year of this Facebook page he had asked 150 people to be his Friend, and only 24 had said yes. So they felt that meant he was asking too many people to be his friend. He said, well, I was asking people who had fanned us on the fan page if they wanted to be friends with us. It seems to me that's not a lot of spam, to ask 150 people to be your friend and then - over a period of a year.

Steve: Unh-unh.

Leo: So we won't really - we can't ever know why. They've reinstated his page, of

course, because they got a lot of attention. Then during the show, and you probably heard this, as well, I said, well, let's test it. Why don't you post on your Facebook page, I'm thinking about deleting my page, here's how, and a link to the wiki how article. About half of those posts were immediately deleted.

Steve: Oh, no kidding.

Leo: Facebook says, well, our spambot detected - believed that they were spamming that link. But I find this all very difficult to believe. But even if it's true, the underlying point remains, which is you don't control the content of a Facebook page. So don't think of it as your web presence on the 'Net. Even if you persist on Facebook, which is fine, I don't have a problem with that, first of all, just treat it as a public page. And second of all, have a page like your blogger page or somewhere where you control it, and it can't be deleted. Now, even Blogger has deleted pages, if they're pornographic and so forth. So if you run your own server, or you go somewhere where they don't do that, that's fine. But you need more than one. And you need a presence on the web that is yours, I think. Don't you?

Steve: Well, yes.

Leo: You have GRC.com. No one can take that away from you.

Steve: Exactly. Although, again, for these - certainly for 400 million people who are using Facebook, I can see the benefit of creating a web presence, having the facility, like I guess MySpace was before, where you're able to put yourself on the 'Net and be found and connect up and do the social thing and all that. So, I mean, I definitely can see a place for it.

Leo: Yeah. Oh, yeah, it's valuable. In fact, I think the real problem, it's so good and so useful that it's like a roach motel. You can't check out. So I just - I have not missed it. But I love Twitter. I use Twitter. I'm very happy with Twitter. And Buzz, I love Buzz even better. I mean, there's lots of...

Steve: Yeah, well, I mean, you and I are connected and technologists, and certainly our listeners have the ability to put pages up and so forth. Speaking of blogs, GRC and Steve, that is, me, now have blogs.

Leo: As you say on the page, you're entering - dragged kicking and screaming into the 21st Century. Wow.

Steve: Blog.grc.com will take our listeners to GRC's low-traffic blog; and Steve.grc.com is mine. And I actually had these almost put together last week, when you were mentioning - we were talking about WordPress and SquareSpace. And I didn't say anything at the time because I'd already made a pretty substantial investment...

Leo: That's fine, yeah.

Steve: ...[indiscernible] WordPress. I couldn't say anything, or people would jump on it immediately and say, hey, wait a minute, this doesn't - you have typos, and this is only half written. It's like yes, yes, yes, I know. Anyway, so I officially launched it. The email system that I had built 11 years ago currently has 790,006 members. Now, the problem with having 790, or 792,006 members is I can't send email to those people without immediately being shut down as a...

Leo: Spammer.

Steve: ...mass spammer. So my intention is that the blog.grc.com blog will be the replacement for what was GRC's email system, that is, my place to announce corporate-level stuff. There's two postings there now. The first one is called "2008 and 2009: Where Did They Go?" Essentially what was I doing during those two years, to sort of explain where the time went, and the projects which were started and are near completion. And then 2010 is the second posting, which is my plan as of this moment for the year 2010.

So what I would hope people do who are listening and interested is go to blog.grc.com and subscribe to that, which will simply give them a notice when I put something there. It'll be low traffic. It'll be strictly GRC-related news, new services, new things, new features, updates and so forth, new freeware. And there's a bunch of stuff that I'm in the process of wrapping up from '08 and '09 which I'll be announcing before too long, over the course of time. And then my plan is to very delicately and slowly trickle one final piece of email to that mailing list, starting with most recent to least recent, because certainly many of the email addresses I'm sure are long since dead, since it's been many years since I mailed anything to that 792,006 people.

Leo: Geez, that's an amazing number.

Steve: So I'll just trickle out a notice saying, look, this is the last thing you will ever get from this email system. We are formally shutting it down in favor of a blog. Please go to blog.grc.com and subscribe to that. Because, of course, WordPress isn't going to be shut down as a spammer. They're sending email out - I almost said they were sending spam out - they're sending email out all the time for all the subscribers to all of the blogs.

Leo: This is a very nice blog. You did a nice job. I really like it.

Steve: Thanks. Clean, simple, just sort of the basics.

Leo: WordPress.com is a great place. I mean, it's free, and it's powerful, it's robust, it's very simple...

Steve: Well, and I did go look at SquareSpace. And I thought, whoa, this is way more

than I need.

Leo: Yeah, probably the case.

Steve: Because I do have my own website, and I didn't want to rebuild a website. I just wanted a facility that would allow me to do posts. And speaking of that, over at steve.grc.com, and those are all - both blogs are cross-linked, steve.grc.com will be sort of my personal column. I only have one thing there now. I wrote my first blog called "Facebook and the Ford Pinto," which reminds us of basically this notion of what corporations' goals are and that Facebook exists for the purpose of leveraging the information provided by its users. And so, I mean, there will always be this tension between the users and their interest in privacy versus Facebook's interest in using that information somehow.

Leo: And you're getting already a ton of commenting, which is great. That's really great.

Steve: I have some really great followers. I did, I tweeted that the blogs existed. That's how there's any...

Leo: See the power of that? They go hand in hand, don't they.

Steve: Yeah, it really makes a lot of sense.,

Leo: Yeah. Well, well done. Welcome.

Steve: So we're getting there.

Leo: Welcome to our century, Steve.

Steve: Speaking of being in the century, I wanted to mention that, for people who pooh-pooh Twitter - and I don't have anything against people pooh-poohing Twitter. I mean, when I mentioned that I was doing this over in the newsgroups that are traditional, GRC's traditional NNTP old-school newsgroups, there were a lot of people who said, oh, I've lost all respect for you. Well, actually there was one person who said that. But there were certainly a lot of people who were like, oh, no, this means you're going to leave the newsgroups? It's like, no, no, no, it doesn't mean that.

But now that I understand what Twitter is, I want to take a minute just to explain it for people who keep hearing it and just don't quite get it. What I now understand is the degree to which it is public. It is 100 percent public. That is, everything about it is public. When you look at someone on Twitter, like you go to Twitter.com/SGgrc just in your web browser, there is the history of postings, the tweets that I have put up. And you can see how many people are following me. Well, then you can click on followers and see who's following me. And you can click on any of them and see them, and who's following them,

and who they're following.

And my point is that this is all open. It's all wide open and all public. And so I sort of like, I mean, I like that that's what it is. Nothing is hidden. Nothing is secret. There's an API. I actually spent part of the day yesterday writing code, which will be up shortly, to the Twitter API so that that new page on GRC, GRC.com/news.htm, that's the page that those three icons in GRC's menu, the sitewide menu now link to. They all just go to there to say here's the way to do things with GRC. We've got RSS feeds for the blog and for Twitter, the GRC and my personal Twitter account, and something else. Oh, the blogs and twitters and RSS. Yeah. All three of those things.

And so what I like about Twitter, or what I now understand, is that it's just a really public open system. So everyone who's there, you can see who they follow, you can see who follows them, you can follow those links down this massively interconnected tree. And everything that they've ever tweeted is in some database somewhere, which all these clients allow you to access and see, if you're curious. So that's what it is. It's just sort of, it is what it is. It's a whole bunch of people who are interconnected and sharing these short little bits of information with each other. And so for me it's useful to just share things that I run across, share news of things happening with me and GRC. So I think it's neat.

Leo: It's fantastic. I'm really glad you're doing that.

Steve: Yeah. And I learned of something, thanks to someone sending me a note back on Twitter. Someone whose handle was, or is, Buckwalter, found that Skyhook had a WiFi update facility. We talked last week...

Leo: Ah.

Steve: Yes. In the Q&A there was a question from one of our listeners who said, hey, just to remind our listeners of this question last week, his company had moved five miles away, and he found that his iPhone was - the mapping and the geolocation system was being confused until it got a GPS fix. When it was using WiFi it was getting the wrong location because they'd moved the access points. So the Skyhook technology hadn't been updated. Well, you can go to www.skyhookwireless.com/howitworks/submit_ap.php, or he sent me a bit.ly, a short URL, so it's bit.ly/xHMnu. And that expands to the same URL. And what that takes you to is Skyhook's interface for updating their database, specifically for fixing this kind of problem.

Leo: So you go to a website to do it, though. You log in.

Steve: Yes.

Leo: Okay.

Steve: Yeah, you go to their website.

Leo: That's a good way to do it, actually.

Steve: Yeah, it's great. And so they give you instructions for how to get the MAC address of your access point. And so basically you type that in and provide the information. And so there is this sort of way to close the loop and update their database, which I thought was pretty cool.

Leo: Excellent, excellent, excellent.

Steve: And I know of that thanks to Twitter. So there you go.

Leo: You see? You see?

Steve: Meanwhile, a listener of ours, Ed Gillett, sent a nice testimonial. He said, "Steve, yet again, SpinRite has saved the day and brought a Windows 2000 Server with lots of data" - I guess if it's a Windows 2000 Server it's had time to accumulate lots of data - he said, "back to life." And then he says, in all caps, "INACCESSIBLE BOOT DEVICE BSOD," which of course is the Blue Screen of Death. He said, "I keep a copy of the SpinRite CD in my car when I go out to a client's site or to friends, just in case. In this case, Windows 2000 Server C: drive had a corrupt boot sector after a power cut. Irritatingly, the server was attached to an APC UPS. But after the last rebuilt of that box the management software to perform the secure shutdown when battery was depleted hadn't been installed."

So of course what happened was the power outage was longer than the batteries would last, and so the technology wasn't there for the APC UPS to tell Windows to shut itself down. Instead apparently it just died. So he says, "Anyway, quick Level 2 scan blasted through the disk, with one unrecoverable sector right at the start. Level 2 completed," and then he ran a Level 4, which indicated that the problem sector no longer had issues. He rebooted; and, presto, all up and running. He said, "My SpinRite has paid for itself several times over by now. It's a pleasure to have it in my toolkit. It's previously dealt with some truly knackered," as he phrased it, "dying disks and managed to resurrect them enough to pull data off them before giving their final sigh. Servers, laptops, TiVos, you name it, my SpinRite has repaired them all. One of these days I'm going to hit a hard disk with a hammer, burn it, and then immerse it in water, and see if SpinRite recovers it, too. I will let you know. Hats off to you, Mr. Gibson. Thanks again, Ed Gillett." So, thanks, Ed, very much for the nice feedback.

Leo: Excellent feedback. Well, I'm ready if you are. We're going to get to our fundamentals of computing series. Kind of the final, although I think networking really counts as still part of it. But the operating system is about as high a level as you can get in the PC.

Steve: Right.

Leo: So let's hear how operating systems work.

Steve: Okay. We began back in the '50s, so operating systems are - today here we are in 2010 recording this. As of this day, or date, or year, rather, not specifically this particular day, they're 60 years old. Originally computers back at the beginning had no notion of an operating system. They were just these very expensive, rather limited machines. Typically they had, like, word sizes of 32K. Sometimes big ones were 64K. But, I mean, that's as big as they got back then. That wasn't a small machine. That was a big, multi-hundred-thousand-dollar installation in a university would have 32K.

And so the problem was, how do you make use of this machine? And in fact over time, as machines grew larger and also a lot more expensive, keeping them busy was the biggest problem because it would take time to load a program into the operating system. Then the OS would spend some time running, and of course you've got the debugging of the program to deal with, too. I mean, the thing to remember is that computers at this time were rarified. I mean, this was the men in the silver - silver - in the white smocks on elevated floors with air conditioning. And you had one machine that basically held hundreds of people in thrall because their jobs were about keeping this very, very expensive, half-a-million-dollar piece of equipment busy.

So the original model was what was called an "open shop," where the people who had a job to run on this machine would sit down and load their code, load their program into the machine over some length of time and then work with it for however long it took. Meanwhile, everyone else was standing around tapping their feet, wondering when they were going to be through, because it was their turn. Again, the goal was to keep this thing that was so expensive somehow busy all the time.

So this open shop model switched to a so-called "closed shop," where instead of the actual people who wanted the work done doing it themselves on the machine, you created sort of a layer of separation. Now, instead, people submitted their jobs to the staff, who were more adept and more expert at using the machine more quickly. So we got a level of efficiency that way. And also there would be a queue of jobs to be run that had been submitted, so there was a backlog. Well, of course this meant that people often had to wait longer to get their results. But in general the goal of keeping this machine busy, being better at keeping it busy was achieved that way. And so that sort of introduced this notion of batch processing, where there would be a queue of things to do, and people would submit their work into the beginning of the queue, and then experts who were better at processing the data or these jobs in the queue would then do the work.

Well, the problem still was that there was a large setup time in between these jobs. So, for example, programs might take a few minutes to run, but would take half an hour to get themselves set up and going. So people looked at these systems and saw that they were still being very inefficiently used. And the problem was the I/O, that is, typing up punch cards and then having a stack of cards would take a long time to read a deck of cards into the machine because the machine was much faster than the card reader.

And similarly, on the backside, ultimately you were trying to print out reports of some sort, and the printer was much slower than the computer. And we've talked about this when we were talking about interrupts and I/O, the I/O systems, the idea that the computer could be busy doing other things, and every so often send a character to the printer, the printer being so much slower. But here, in a system that was - there was no concept yet of timesharing, of multiprocessing, of doing more than one thing at once.

That just hadn't occurred to anyone. Well, and frankly the systems at the time, they weren't capable architecturally of doing that. They didn't have stacks. That didn't come along until later with Burroughs, the early Burroughs machines, the 5000 series machines, first introduced a stack architecture.

So here we've got this machine that is 100 percent I/O bound. It's sitting around waiting for cards to get read. When the program is finally loaded in, then it takes a short time to run it compared to the time it took just to read it in. And then on the backside, once it's done, now the computer sits around waiting for the printer. So again, looking at this, it's like, okay, how do we solve this problem?

Well, what they used was they used magnetic tape in order to decouple the slow speed of the physical I/O devices from the much faster speed of the computer. So now what happened was, people would punch their cards as sort of like the slowest step of the process. So now you'd have card decks. Then there would be a number of machines which read cards and wrote them to mag tape because mag tape was much faster than cards. So jobs would get punched on cards, and that would happen by some keypunch operator that was skilled at running keypunch. Then those would get written to mag tape. And as many jobs as possible would be put on the tape.

So the tape would be filled up with jobs to do until it was full. Then that would be mounted on a - taken from this machine that its sole purpose was just to read cards onto tape. Then the tape would be stuck on a large mag tape drive connected to the computer. And it had several mag tape drives that were being used in round-robin fashion for input, and another set used in round-robin fashion for output. So this mag tape would get loaded. And as soon as the computer was done with the previous mag tape on a different drive, it would start reading in jobs from mag tape drive number two. Meanwhile, the first one would be rewinding. It would get pulled off and stuck back on the card-reading machine.

And so you could see, I mean, literally it's like, I mean, the people were running around spending all their time trying to keep this machine busy. And you can sort of think of it as like a hierarchical funnel where slow processes fed into a faster process that then fed into a finally very fast process. The upshot of this was that you finally had this very expensive machine. Thanks to having funneled all the I/O into it, you had solved that speed problem. You were still running everything one at a time. That is, again, still no notion of the machine doing more than one thing at a time. But at least now you had solved the input problem so that this thing was - the machine itself was now very quickly loading programs off mag tape, running the programs, and then dumping the output back onto one of a number of output mag tapes.

The printers weren't capable of keeping up with that. And so you had sort of the same sort of, in the way that you have a funnel on the input, you had an expansion on the output. The computer would fill up a mag tape with output and then switch to the next mag tape that was waiting. The operators would pull that full mag tape off, now take it to offline printing, where that mag tape would then be servicing a number of printers that would print the results out. And so that was sort of like the next level of how to keep this extremely expensive machine busy.

So that notion, this whole thing was called "spooling." And it's funny, I didn't realize until recently, when I was doing some buffing up on the history, that "spooling" was actually an acronym. It just seemed to me, I mean, and I'm sure you, Leo, have heard of, like, spooling this off to tape, spooling it off to the printer. Spooling seemed like a thread on a spool, where you were just storing it somewhere. Turns out that it's an acronym. SPOOL stands for Simultaneous Peripheral Operation On Line.

Leo: Oh, you're kidding. SPOOL is an acronym?

Steve: Yeah.

Leo: I never knew that.

Steve: I know, I didn't either. I ran across it.

Leo: It makes sense because it's spooling something out like it's a thread. So I thought that that's what it was doing.

Steve: I always just assumed it was a verb, but it's an acronym.

Leo: Holy cow.

Steve: Simultaneous Peripheral Operation On Line, that's where the word came from.

Leo: How funny.

Steve: So it was in 1959 that, essentially after 10 years of this fire drill of trying to keep a very expensive single machine busy, that John McCarthy at MIT, who of course famously gave us the LISP programming language and lots of other computer science in between, he wrote a memo first suggesting the notion of timesharing, the idea being that - which was radical at the time - the idea being that, rather than the entire machine being committed to a job, to a single thing, we would slice up time and handle it in a round-robin fashion or in some sort of a priority fashion. But the idea being, have many different jobs operating in the machine at the same time.

And in fact McCarthy proposed the notion of terminals being connected to the machine in such a way that the individual users all thought they had exclusive use of the machine. He recognized that the duty cycle of someone typing characters, and how rarified character typing is, would allow essentially the same kind of funneling as was being done with keypunch operators concentrating onto cards, concentrating onto tape, concentrating onto the machine. He said hey, you know, you could achieve the same thing if you had real-time connection among many different consoles, all feeding into a machine which was able to look at them all at the same time. And so which was at the time a huge change, conceptually a big leap in the way systems operated.

So we talked last time, when we talked about the episode called "The Multi-verse" - I guess that was actually three weeks ago because we had a Q&A, and then we had the portable dog killer episode. Oh, and a Q&A before that, so I guess four weeks ago. Anyway, we talked about in the multi-verse this notion of - we've looked at this concept of the stack, which is able to store the state of the machine; and that hardware interrupts that were caused by people pressing keys or a printer saying I'm ready for another character, they could steal tiny bits of time just to fill an input buffer or to empty

an output buffer, and then switch right back to what was being done before. And so this notion of interrupting the flow of instructions, and saving the state of the machine so that you can restore it when you come back, and jumping somewhere else and doing something, it's the growth of that that created what we now have with the current operating systems.

So operating systems over time have had various structures. Before there was a theory, sort of, of operating systems, before we had enough experience, they were just sort of big, monolithic things. The idea was that programs would be clients of the operating system. There would be sort of this code in the machine which is always there. That's the part that's not changing. Programs would be transient. They would come and go and run for some length of time while their user was running them, and then they would be terminated, releasing their resources.

And one of the things that quickly happened with this notion of timesharing was that the designers, the people wanting to use these resources were a little bit aggressive because notice that one thing happens when we chop time up, which is different from when we process jobs sequentially. If we process jobs sequentially, each job has the entire machine to itself while it's running. That is, however much RAM, for example, the machine has, the job can fill all of that RAM. But with McCarthy's notion of timesharing, things get a little complicated because, if you're going to have many programs running at once, then they've all got to be in RAM. That is, now - and I'm saying RAM because I'm used to today. But that was core.

Leo: Core, yeah.

Steve: Yeah. And 64K, remember, was the kind of size of core these machines had. So one of the notions that was created was this notion of swapping. And swapping was a sort of an early form of what we now call "virtual memory." We'll talk about virtual memory and how that works a little bit later in this episode. But with swapping the idea was that you would have fixed-size partitions in core. And you might have, say, 16 users, using the machine at once, at terminals. And the single computer that was being shared might have an operating system that used some piece of core.

And again, back then the operating systems were all written by hand in assembly language, which we know is sort of just an ASCII-ized version of machine language. So a one-to-one correspondence between instructions that had been manually written by the operating system creators and the code that's being executed. And then whatever memory was left over might be, for example, divided into four partitions. So you might have, say in a 64K machine, you might have - normally these operating systems were relatively small because they weren't doing that much. You might have 8K taken up by the operating system, and then the balance of that memory, like, what, 56K would be divided into four equal-size pieces. Those would be partitions.

And you'd have, however, four partitions and 16 people. So obviously we've got a problem because what these four partitions mean is that the system can only actually be running, can be quickly jumping between four programs at once. So swapping was the solution they came up with. They said, okay, we'll have a drum, and drum memory was - this predates disk memory. And we will have partitions on the drum to essentially hold all 16 of these partitions. And we'll swap them one by one, or actually four at a time, essentially, into the machine's memory. So four programs can be running. And after a program had run its allotment of time, it would be sort of frozen and put out, swapped out onto the drum. And another waiting program would be swapped into one of these

partitions. And then it would be, being now in core memory, it had the opportunity of running.

So that was the way, back then, they solved the problem of needing to have essentially more resources because now they were trying to do this timesharing. They needed to have more resources than the machine could technically support because oftentimes you couldn't just add memory, even if you had the budget. Remember that the instructions were so short that the instructions themselves couldn't address more than 64K words of memory at the time.

So originally there was no sort of theory of operating system design because no one had ever designed one before, or written one before. So operating systems started as sort of big, monolithic systems. And they sort of grew awkwardly and not very elegantly. And back at this time a lot of this research was being done, if not all of it, primarily in universities. So the university sort of model is one to say, okay, wait a minute, time to reset. This thing, this operating system has gotten unwieldy. No one really - the people who wrote it graduated from - got their Ph.D.s and left a couple years ago, so we've got these huge chunks of code that no one here knows about anymore. We need to just start over.

One of the approaches that was taken was called a "layered" approach. As people began to apply real sort of academic discipline and thinking to the design of an operating system, they said, okay, wait. Let's try and do this in, like, a layered fashion. The bottom layer will be just the processor's own usage and allocation and the division of time. Then on top of that layer we'll write the sort of the management of memory and the drum swapping code. And then on top of that we need to have an operator, some operator interface that is able to control starting and stopping these jobs running. And then on top of that we need to have input/output so that the operator's console can receive input and then display the output. And then on top of that is user programs that are running. So there was this attention, they began to get the concept of some sort of structure, a hierarchy, a layering of the way things were being built on the OS.

The next problem they ran into, as systems became larger, was again one of complexity. The problem was that they recognized quickly that mistakes that were made in the operating system brought the whole system down. And the operating systems were continuing to sort of grow without end. So at one point this notion of a microkernel was proposed, the idea being to recognize that the kernel is this privileged resource where it's sort of the supervisor, the monitor of all the programs that are running on it. And it provides services, which we'll be talking about in a second, to the programs that are running on the operating system.

The problem with it getting big is that there's a given number of mistakes are going to be made per thousand lines of code, and that just sort of seems to be an immutable fact of the way software is being written. So if that's the case, and if mistakes are bad, and they're especially bad in the kernel because a mistake in the kernel brings the whole system down, as opposed to just a user program being aborted, but everybody else gets to keep running, a mistake in the kernel and it's game over. So the logic was, make it as small as possible. If it's as small as possible, it'll have as few lines of code as possible; and, based on the average number of mistakes per line of code, there'll be fewer mistakes. So the microkernel approach was specifically designed in order to minimize the number of bugs that could bring the system down.

On top of that, then, was created this notion of a client-server model. And that's what we now see in modern-day, for example, UNIX systems and in Windows, where you have a relatively small kernel which provides the fundamental services to both user programs

and services which are running on the system. So the idea is that the kernel is very small, and then - but also not very capable. It does the fundamental sort of lowest common denominator things like handling processes, dealing with threads, slicing up time, managing memory, managing file systems. Sort of the core services of the operating - that everything uses in the operating system. And then you've got two classes of programs essentially running on top of that, that is, using those services. You've got service programs which provide additional features, much richer features. And then you've also got the actual client programs that are clients of those services. And that's sort of the model of operating systems that has evolved.

We've sort of been talking, we were talking originally about mainframe operating systems and evolved into this notion of personal computer operating systems. But it's worth noting that there are operating systems of some sort in a microwave oven. As we were talking about complex systems running in cars, there's operating systems in cars. I mean, there's operating systems literally in anything today that is complex enough to have a computer in it. There will be some sort of operating system. So there's also a hierarchy sort of in OS scale. Mainframe operating systems still exist today, running big Cray hardware. Large insurance companies will have huge mainframes, still with the - I don't know if the guys run around in white coats anymore. But large-scale systems that are still doing oftentimes batch processing, that are producing reams of reports and data, and also lots of communications. They tend to be very I/O heavy.

Coming down smaller we've got server operating systems which are still large, capable hardware that are then servicing the needs of many people at once. Then we come a notch down to the personal operating system, running on our computers. We're having many, typically, many things going on at once, but generally serving one person at a time. And then handheld operating systems that are running smart phones and PDAs still have a computer. There's still a core set of services that those are offering to programs that run on them.

And then the notch down are embedded operating systems. And that's the class of OS where you don't think in terms of there being an operating system. There isn't, often there's no file system. There's no sort of generic user interface. This is an embedded OS, for example, what's probably in our cars and microwave ovens, consumer appliances, DVD players and things. We see a very purpose-specific user interface where there's a display of remote control, buttons that you can push. But back there, in there is an OS.

There are a number of companies that license these for the common chips. And they're embedded in also sometimes real-time. An RTOS, a Real-Time Operating System is typically a small system that prioritizes time over features. That is, it guarantees that code that's running on it will be responsive, and that nothing else going on in the OS will tie up the computer's resources more than a certain amount of time. So there's a guaranteed latency in the operating system's ability to respond to the tasks that are running on top of it.

So what do all these operating systems do? What features do they offer? Well, one way or another the mainframe operating systems, server operating systems, personal operating systems, even handheld operating systems, that is, everything above sort of the embedded operating system, they provide a means for loading a program from mass storage into memory. We know now that programs, processes, are typically composed of, well, at least one, but many times many more than one, threads, where threads are sort of an abstraction of a series of instructions that are being executed.

A program could very well just be single-threaded, as is the terminology, where it starts, and the program executes only conceptually one thread of execution at a time. So there's

only, for example, one program counter associated with that process because there's only one thread that is at a specific location at any point in time. And as we explained in the multi-verse, you can have multiple threads because a thread is able to spawn or fork another, essentially another conceptually asynchronous stream of execution so that it's able to go, this forked or spawned thread is able to go off on its own and do something else. For example, it may be responsible for reading the keyboard or updating the display or doing some other things that are sort of asynchronous to the main work that the primary thread of the process is doing.

So the operating system is the thing that oversees all that. It has, somewhere, some hardware, a timer which is generating interrupts at a given frequency. That interrupt is what's responsible for yanking control away from individual threads running in individual processes and getting control back to the operating system. When that happens, the operating system takes a look at what's going on and decides if the thread should continue running, if it's time to give it to a different thread in the same process, or if that process has used up its allotment of time, time to bring back to life a thread in a different process that was previously suspended. In which case it restores the state of a given thread as if that thread had never been stalled and just continues executing where that thread let off.

So that's the job of scheduling, which is the subject all by itself of books on operating system theories. Scheduling is amazingly complex, and it's easy to sort of stand back from a distance and say, okay, well, this just has to divide time up among all these threads. But it turns out that when you start actually doing it, there's all kinds of complexity with deadlocks and stalls and competing priorities. And if something has a low priority, and the things with higher priority never take a breath, then the things with low priority never get a chance to run, so that's not good. So it's like an amazing mess of special cases and, again, has been the topic of many doctoral theses over the years.

One of the other things that operating systems do is allow interprocess communication. That is, they support a means for allowing processes to communicate with each other, which is often something that co-working processes want to be able to do. You want to be able to have some sort of communications between them. Even though the operating system is all about interprocess isolation, in some cases you do need to facilitate communication. The operating system provides that mechanism because there is otherwise no direct means, for example, for one program to mess with another program's memory. You want to manage that carefully to prevent mistakes from happening, and also of course to prevent deliberate abuse from being possible.

One of the other things that all of this relies on is memory management, which we've talked a little bit about but never really directly. The idea is that, in every case, all these operating systems basically produce an abstraction of the machine's underlying resources and of time. Time and the passage of time is an abstraction, as well, because from a thread-centric standpoint, a thread just thinks it's executing all the time. The thread has no awareness that time isn't all its own, that it isn't running all the time because control is yanked away from it, and its state is saved. Then the state is restored, and it picks up exactly where it left off. So from its standpoint it's just running all the time.

In fact, we know that's not the case. There's thousands of threads in a typical system that all believe they're running all the time, when in fact on a single-core system only one is actually running at any time. In a multi-core system you can have as many actual threads running as you have cores. So the operating system is essentially creating an abstraction of the underlying resources. One of the resources is memory. Processes can, much as was the case back in the days of early timesharing, where it was possible for a process to be sharing memory with other processes. Back then you could only have as

many partitions of a certain size as there was room left over after the operating system had been accounted for.

In this day and age we've got this notion of virtual memory and the ever-present, for example, Windows users are familiar with the paging file, or sometimes still called the "swap" file, the idea being that processes believe they have a much larger amount of memory than they actually do. And this is hidden from them. This reality of how much memory they have is hidden by the operating system. If the process attempts to execute or read or write memory outside of what's currently loaded in the system, the operating system, using this virtual memory technology, will get control, literally at the instant of a program's attempt for read or write memory that is not physically loaded at this time. That creates what's called a "page fault."

The operating system gets control at that instant, sees what's going on, and says, ah, the program is trying to access memory which we've had to swap out to the hard drive because of limited resources. Other programs needed to use physical memory. So what happened was that memory was paged out, as is the term, onto the hard drive while another program that was running was actually using the underlying physical memory. So when a program attempts to use memory which doesn't physically exist, it's been paged out, the operating system gets control, sees what's going on, decides, okay, do we page this in, do we suspend the process, there's a whole bunch of decisions to be made. As I said, scheduling is itself a huge issue for operating systems.

Typically what happens is, because physical I/O is necessary and always takes time, the act of the process touching memory that isn't physically located or that isn't physically present in the system, suspends it. The memory is cued up to be written or to be read from the system, except that there's no doubt some other process using the physical memory that's in use right now. So that needs to be written out to the swap area so that the memory which had been swapped out could be brought back in. Like I said, this gets really complicated very quickly.

But the bottom line is, what ultimately happens is, the so-called swap file or paging file or virtual memory file, it's an oftentimes very large extension of the physical memory in the system. And we talked last week, or last time we were talking about this, about the notion of caching, where the processor had very fast access to its own registers. And then it would also have a cache of maybe 8, 16, 32K in the old days, megs now, maybe several megabytes. And that would be a copy of what was in physical memory that would be maybe hundreds of megabytes or a gigabyte. Well, you can see that that forms a hierarchy, from the register in the chip to the cache. And oftentimes there's a Level 1 cache and a Level 2 cache - Level 1 being smaller and instantly fast, Level 2 being larger and not quite so fast. And then you have physical memory.

Well, virtual memory is the last largest layer of this hierarchy, where it represents sort of this super physical memory that is really slow inasmuch as it's written out on mass storage, which is typically a rotating medium in this day and age. And so it's very slow to read and write, but it's very large. And so that sort of - the virtual memory forms the final layer of this multilayered memory hierarchy where we get smaller and faster all the way up to the individual register of the machine. And the operating system manages all of that.

One of the other things that the OSes do is support a file system so that the programs running on the operating system are able to, again, see an abstraction known as files which exist only because the operating system creates that abstraction for the programs that are running on the OS. It knows about the physical organization of the media, the mass storage on the system, but hides all of those details from the programs that are

running under it.

Leo: Steve is taking a small drink at this point. For those of you listening at home.

Steve: My throat was getting scratchy. Believe it or not.

Leo: Well, you said it wasn't a lecture, but this is a lecture, but a great lecture I'm really enjoying. It's fascinating.

Steve: So one of the things that happens is that, notice that we've got floppy disks, which are small, 1.44MB, for example, in the famous instance of the IBM PC, and have a physical organization with some small number of sectors around some small number of cylinders on two sides. But we might have a multi-platter hard disk that's got thousands of sectors arranged around millions of cylinders and multiple platters. Well, the operating system's drivers understand the specifics of the hardware that they're driving. All of that is hidden from the programs that run under the OS. The program sees this abstraction of named files, and so it's able to say to the operating system, hi, open for me this file of this name and let me read it.

And one of the main innovations that operating systems had was that this concept of I don't care where it is, I don't care what media it's on. And in fact, even as far back as remember UNIX and CPM, there was this notion of this could be paper tape, or this could be a hard drive. There was this disconnection with the I/O system and the physical devices that were running underneath the operating system. So this abstraction is provided by the operating system. And then there's also an abstraction provided by I/O drivers, where the operating system would have a defined mechanism for interfacing to, for example, file system devices. And different drivers would be used to interface the operating system to very different storage mechanisms.

So you'd have a floppy driver which is able to answer specific uniform requests from the operating system, translating them into the instructions needed to work with the physical floppy drive hardware. And you'd have a mass storage driver, for a given hard drive, which is able to take the same operating system requests as the floppy driver, but in this case translate them into very different specific functions for its hard drive. So again, we create these boundaries of abstraction where the entities on each side of the boundary have agreed upon a protocol that is uniform, typically documented, and not changing. So drives can get bigger. The drivers may need to change over time. But that agreed-upon protocol at the boundary of the driver and the operating system, or even the operating system and the application program, that doesn't change. And so that allows things on either side to evolve over time, which was another major innovation in operating system design.

And then finally, the last thing that's come along in recent operating system architecture, on top of file systems and I/O, is security, where there's an increased notion of both wanting to prevent mistakes from compromising the system and wanting to prevent deliberate malicious intent from compromising the system. So there's this notion, which is layered on top of everything, of privilege. The idea that certain files can be privileged, certain memory regions can be privileged, essentially there's this notion of ownership and rights which are applied to all of the objects within the operating system, right down to and including the individual thread of execution. Threads can own, objects can have rights, and everything's scaled all the way up from threads. And all the other abstractions

that the operating system creates can have rights.

And so, in addition to everything else the operating system is already doing, it also has to check the security. It has to make sure for, I mean, at the level of the individual action, it has to make sure that that action is permitted, that the thing asking for something to be done has rights to do that to whatever it is asking for it to be done to. So OSes have gone from starting off being basically a loader of code, for a system which was struggling to be used enough, to what we have today. I mean, we're all sitting right now, I mean, you're hearing this through probably a device with an embedded operating system.

Leo: Almost guaranteed. I don't think you can play it back any other way.

Steve: No, there's no other way to play it back. Leo and I are sitting in front of computers that are right now, as we sit here, have thousands of threads all running, all competing for resources, using the hardware, somehow staying out of trouble. And to me it's amazing.

Leo: Amazing. Absolutely. Absolutely. It's truly a miracle that these things even work. It's amazing. Are you done?

Steve: I think that's it.

Leo: Wow.

Steve: That's operating systems.

Leo: I tell you, you've got to put these all together on a DVD or something, and we'd have the complete set of building a computer from scratch. What fun. Now, next week it's a Q&A. So if you've got questions for Steve, if there's anything you heard today that you have a question about, or anything in the security world, you can get a hold of him by going to GRC.com/feedback. That's his feedback form. Easiest way to do that. GRC is Steve's site, of course. Now blog-enabled.

Steve: Yes, and I would encourage our listeners to go to blog.grc.com and subscribe so that they'll get a short note from WordPress whenever I have some information. I'll try to - I'm sure I'll be talking about important things here on the podcast, as well. And of course I've got my personal blog, steve.grc.com, if you're curious about the sort of things more in a column format.

Leo: And follow Steve on Twitter. He's SGgrc on Twitter. SGgrc. And that way, that's his early warning system. I presume anything big would come across there first.

Steve: Yeah, and, you know, navel lint.

Leo: And navel lint about Vitamin D and the Ford Pinto. GRC's the place to go, though, for Steve's great programs. Not just SpinRite, which is absolutely the world's best hard drive maintenance utility, you've got to have it if you've got a hard drive, but also all the free stuff he gives away like ShieldsUP! and Wizmo and all sorts of stuff. GRC.com. You'll find 16KB versions of the show there, as well as the 64KB versions in high fidelity, the transcripts, the show notes, it's all there. GRC.com.

You can watch video of this show. Steve doesn't post that, but we do at TWiT.tv/sn. You can even subscribe to that if you want to put it on your iPad, Pod, Phone, whatever it is you've got. Yes, you'll need threads to view it, I'm afraid. GRC.com. Steve, we'll see you next week. Next week, Q&A time.

Steve: Yup. Thanks, Leo.

Copyright (c) 2006 by Steve Gibson and Leo Laporte. SOME RIGHTS RESERVED

This work is licensed for the good of the Internet Community under the Creative Commons License v2.5. See the following Web page for details: http://creativecommons.org/licenses/by-nc-sa/2.5/