EPISODE 790

[INTRODUCTION]

[00:00:00] JM: An operating system kernel manages the system resources that are needed to run applications. The Linux kernel runs most of the smart devices that we interact with and it's the largest open source project in history. Shuah Khan has worked on operating systems for two decades, including 13 years at HP and 5 years at Samsung. She's worked on proprietary operating systems and a variety of Linux operating system environments including mobile devices.

Shuah joins the show to discuss her work within Linux and her experience contributing to open source. Shuah has made significant contributions to K self-test, a set of tests for Linux. Testing the Linux kernel is complicated, because there's so much depth to the code base and such a variety of ways that Linux can be used. There's also a variety of ways that the operating system can get tested. There's smoke testing, performance testing and regression testing. There are trees of tests, and as a developer you may only want to run a subset of the tests.

The conversation with Shuah ranged from the low-level practices of testing the kernel, to a high-level discussion of how the Linux kernel can reveal dynamics of human nature. It was great having Shuah on the show. I really enjoyed the conversation and it was at the Open Source Leadership Summit put on by the Linux Foundation. So thank you to the Linux Foundation for inviting me to that.

Before we get started, I'll mention two events that are having in the near future. One is the Software Engineering Daily Meet Up at Cloudflare. You can go to softwareengineeringdaily.com/meetup to find out more. It's April 3rd and it will be a conversation with Hasseb Qureshi, a cryptocurrency investor and friend of mine and he's been an engineer at Airbnb and many other places. He's really a great guy and interesting to talk to. So to find out about that event, you can go to softwareengineeringdaily.com/meetup.

The other event is the hackathon. You can go to softwareengineeringdaily.com/hackathon. This hackathon is for a product I'm building called FindCollabs. It's a product for finding people to

work with on your projects. We're having an in-person hackathon at App Academy and we're going to hang out. We're going to have some food. It's going to be on Saturday, April 6th, and you can find out more details at softwareengineeringdaily.com/hackathon. I would love to see you there if you're in the city or in the Bay Area. So softwareengineeringdaily.com/meetup, softwareengineeringdaily.com/hackathon, if you're interested in either of those.

Now, let's get on with the show.

[SPONSOR MESSAGE]

[00:02:55] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW]

[00:05:02] JM: Shuah Khan, you are a Linux core committer. Welcome to Software Engineering Daily.

[00:05:07] SK: Thank you.

[00:05:07] JM: I want to talk to you today about operating system kernels. Let's start by just explaining what is the rule of an operating system kernel.

[00:05:18] SK: Operating system is you have a piece of hardware, laptop for example, and everything that a human needs to do on that. Operating system provides drivers and then kernel core, you have memory in the laptop and how do you initialize the memory. How do you set up the hardware? How do you get the drivers, different pieces of hardware to a state working, for example, audio on the laptop or your screen? All of that is what operating will do for you. It's essentially making the hardware usable for you to do – For a user to do what they need to do with that piece of hardware.

[00:06:07] JM: You've spent a lot of time on the testing process for the Linux kernel. When you're talking about these different elements of an operating system, like media playing, for example, how do you test something like that?

[00:06:21] SK: That is a good question. I do contribute to Linux media subsystem. So how do you test? You have a driver part, obviously, the media drivers, and then you use various user space applications to do the end-to-end testing. There are different levels of testing we do. For example, we would just do – Does it boot, for one thing? That's the first level of testing – And operating system. Does the kernel boot? Come up to user prompt? Can you log in? Can you start applications, browsers and so on?

Say you are talking about a particular media driver. You have a USB stick that does digital TV or analog TV and so on. So you connect that. So does it power up with that? Then after that, you go looking for actually starting an application that would start streaming perhaps and see are

you streaming. Can the application stream? Can you switch between channels and so on? So there're different levels of testing.

So to get to that level, you have to go through different layers of verifying different pieces work correctly. We also do some drivers, DRM drivers, or video drivers, or digital media drivers. They also tend to do compliance testing. That means is the higher level application that is using the API, kernel API, do they use it correctly? There are tools that do compliance testing. Testing is a large area depending on what you are planning to do.

So as a developer, if I'm developing, fixing a bug, or developing a new feature, I have to first learn to understand what are the pieces that I need to use to test that particular part. Kernel self-test itself is a regression test suite. Over there, we have developers. As kernel developers, we come up with a test and say, "Hey, this is the test I want to run when I am taking a new patch, or I want to be able to just quickly apply the patch, boot the kernel and run this test, run these shells." It could be shell script or a C program, just run it to make sure that the patch doesn't do any regressions.

In some cases you have to verify that the new feature itself is doing what it says it does. So there're those two levels of testing. So kernel self-test is a suite of developer tests so to speak. We go and do that kind of testing.

[00:09:16] JM: Is there a sense of continuous integration or continuous testing in Linux kernel? I'm much more familiar with web development, and you obviously have continuous delivery in web development. How does that compare to operating system development?

[00:09:33] SK: We do that in the Linux kernel space. What happens is we have test rings that continuously take patches that are coming and run various kinds of tests. Like for example, just a compilation test. It could be kernel has several configurations. We have various configuration options that can be turned on. For example, you might have a patch that compiles just fine that the developer compiles and test with it. Then once it hits the integration ring, it might fail to comply on some architecture or some particular configuration. For example, i386. It could fail on that.

So we find those – We have several gits, master – Linux's master git on kernel.org, then we have linux-next, which is we call the continuous integration. So we're doing continuous integration every single day. We have a linux-next release that developers use to test stuff continuously. So does that help you understand?

[00:10:44] JM: It does, yes.

[00:10:44] SK: Yeah.

[00:10:45] JM: You mentioned this tool, K self-test. If I'm a kernel developer, what is my usage for K self-test?

[00:10:54] SK: It is a test suite, so not so much a test tool. So if you are a kernel developer, what you would do is you'll use the kernel repo and you can kick off a – It works from the kernel make file, main make file. You will run, you'll check out your kernel sources and then you'll compile the kernel and, say, makes that k self-test.

What that does is it will build all the test that we have and it'll run through all of them and then give you results for that. So the way I use it and then also a lot of developers use it is they will either run the entire test suite or they go into a particular subsystem and then run just to those subsystem test, like for example MM, somebody that made it change to the kernel MM space. They would just quote on that particular subsystem. You can run individual subsystem tests as well using – There are multiple knobs that you can use. So you can go look at one subsystem and say, "Hey, I want to just run test in that subsystem," or "I want to just run test on this particular driver." So that's the kind of knobs they use.

Then in some cases it could be installed on a target test system and tests can be run on the target. For example, kernel [inaudible 00:12:23] test forms. They use that. They take kernel self-test. They built it on their development system and then they take it over to a target system and they run it right after they boot the kernel.

[00:12:36] JM: How does that fit into my workflow as a developer? Maybe I'm tinkering on some eye-level element and then would I want to run k self-test on some lower level elements that are underneath that. How am I using these suits of tests tactically?

[00:12:53] SK: So depends on who you are, right? So if you're a kernel developer and you made a kernel change, you want to verify that kernel change before you send the patch up. In that particular case, you will go, you will make sure that you run the kernel self-test either particular to your area of the change you made and make sure that nothing broke.

As a user, say you just installed a new kernel and you wanted to make sure that your kernel – Nothing regressed on your side of things. So you could run it as a distro provider, or a user, or a [inaudible 00:13:35] developer. Some [inaudible 00:13:38] platforms, they'll use all the revisions of the kernel. For example, the LTS releases, long-term releases. They're not always on the main line kernel. If they want to verify nothing broke.

[00:13:53] JM: In that case, I would run the whole suite.

[00:13:55] SK: Correct. Correct. As users, you use the whole suite.

[00:14:01] JM: And the subset? When I want to use the subset?

[00:14:04] SK: The subset, if you tweaked to something, fixed to something or added a new feature. So most of the kernel developers, what we do is as a developer in process, when we write a new feature, we would – In some cases we would write new tests to go with that new feature. Then we make sure that the existing test as we made this new change to the kernel, that we did not break anything. That's the regression part of it, which we already have in the kernel self-test. We add a new test and say, "Hey, from now on we want to test this feature. First we want to test this feature that does it work the way it's expected to work. Then going forward, as other changes keep coming in, does this feature continues to work the way we want it to work?"

[00:14:56] JM: Just to give the listeners some perspective for why testing is such a big deal for Linux, Linux is of course running in extremely sensitive environments. You have mission-critical

things that are life and death and it's really important that bugs don't sneak in to lower-level aspects of the codes. Linux is also very complicated. There're many different kinds of Linux. There're different kinds of devices that run different versions of Linux. Because of that, the amount of tests and the different sub-trees of testing types can get quite detailed. Can you describe I guess the testing tree or how the tree of different tests across Linux is managed?

[00:15:51] SK: Within the kernel we have kernel self-tests. That's really one single place that we have tests, user visible test that can be run. Individual drivers and subsystems, they could have some self-test they kickoff and run. For example, you will have a test code, driver for example, or a subsystem might expose certain testing sort of interfaces to just the testing programs. So the test, kernel self-test, test use that to exercise the kernel. Of course, testing is very important. That's the part of validating, qualifying kernel is really important, because it is the foundation piece in any of the Linux ecosystem. You want to have those bits.

So we have various methods in the kernel to verify things. For example, we have – If I am changing a driver, for example, I would go turn on configuration options in the kernel that makes sure that my locking is sane. That means I'm not locking here and leaving the lock, leaving that lock, not unlocking it. Those lock and unlock are balanced. So I can go into the kernel and say, "I want to turn on those options," which I do.

Then in other cases we have other debug options that will go and say, "Look at [inaudible 00:17:37], for example," or just standardization. So we can turn that on and then it will go make sure these are all not something that you would turn on in the field, but you would do it during the development, or you would do it as a validation mechanism before saying everything looks good. So you would go and say, "Hey, I'm going to turn on the sanitizer and tell me if I have any memory that I am using after free or any kind of memory issues I might have." That's one example.

So we have various tools within the kernel, configuration options, compile time, configuration options that we can turn on and debug. So we use all of these mechanisms. In addition, some cases we have tracing. We can turn on tracing on trace events. Like for example, you have a host and then you are KBM, you have ABM running. So you say, "Hey, I want to take this device and assign it to the VM." So if we are interested in looking at, "Hey, how this process is working?

Is it correctly working?" So we can go and turn on trace events during runtime, and then turn them off if we want.

[00:18:59] JM: Tracing is an interesting one, because I've done some shows about distributed tracing. I'm assuming tracing is somewhat just not in distributed tracing is somewhat similar, where it's measuring the latencies through the different paths that a call might be taking or it's measuring at least that the call is making its way through the different lower levels, or I guess maybe you could define what tracing is.

[00:19:22] SK: Right. In this particular case, these are trace events, a way for us to get a feel for how a particular user action might be taking paths similar to that. It's kind of taking paths in the kernel and we want to make sure we are tracing the path it's taking. It could be used in two ways. It could also be used how fast something happens, performance tracing. The other aspect is, is it taking the right paths in the kernel that you expect it to take? So as a debugging tool. So those are two different users of that.

I think what I'm getting at is that we have multiple ways to debug and test the kernel and then we use, depending on what we are doing as kernel developers, we use all of those avenues to test the kernel during development integration and then debugging.

[00:20:26] JM: So if I had some change and then I ran a performance trace on it and there was a small increase in latency due to that change, how would you know if that performance penalty is like a bug or is something that would be considered a regression, because sometimes it's a tolerable increase in latency. It's not a big deal. Nobody is ever going to recognize it. Is there some judgment that's involved in that process?

[00:20:57] SK: Yes, yes, depending on how bad the performance impact is and why it's introduced. The way I go about it is I have to first root cause and say, "Why am I seeing the change?" So what happened? Is that change – Can that change be explained or is it avoidable? Maybe there is something in that path in introduced something that I might have an alternate mechanism that does not introduce that performance penalty. It all boils down to root causing and analyzing is it absolutely necessary to do so? In which case, the code needs to be

rewritten, right? I mean, there might be a better way to do that piece of code. So it depends on what kind of performance impact and how it can be solved. It also depends on the nature of it.

[SPONSOR MESSAGE]

[00:22:07] JM: HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

Go to softwareengineeringdaily.com/HPE to learn about how HPE OneView can improve your infrastructure operations. HPE OneView has easy integrations with Terraform, Kubernetes, Docker and more than 30 other infrastructure management tools. HPE OneView was recently named as CRN's Enterprise Software Product of the Year. To learn more about how HPE OneView can help you simplify your hybrid operations, go to softwareengineering daily.com/HPE to learn more and support Software Engineering Daily.

Thanks to HPE for being a sponsor of Software Engineering Daily. We appreciate the support.

[INTERVIEW CONTINUED]

[00:23:30] JM: What are some other judgment calls you have to make? So latency might be one. Okay, is this latency actually going to matter? Let's say you do a root cause analysis, you find that, okay, the root cause is actually like just a tradeoff that we made in the code and it's actually just going to increase the latency, but it's not a big deal, because it's something that's asynchronous and it doesn't actually matter. So we're going to just leave it in there.

But are there any other areas of testing that – Like here's an example, you might say, "Okay, we introduced this new module and we've written 100 tests for it. We could write a thousand more. Should we write a thousand more tests?" So that's an example of a subjectivity kind of question, like the question of do we have enough testing?

[00:24:23] SK: So testing is an interesting thing. You really have to verify even the test code, right? Test code itself could be buggy. So there is always a balance of how much testing. It needs to be targeted testing in my opinion, and you can have – That's how I view testing. I go and look at how you can verify – There're two kinds of testing, white box and black box. You probably heard about that, functional testing versus oral testing. Black box testing would be – A lot of kernel developers, we are doing functional testing and white box testing. We kind of know in some cases what to look for.

Some of those tests are important, because if something shows up in a white box type testing, functional type testing, like lock imbalance, it will ultimately show up in a black box user test case. When a user is using, they will run into it. So do they run into it after 10 hours of use or one hour of use? That's a different thing. Is it a race condition or is it going to happen all the time? So there is always a balance. To me, there is a mix of white and black box type tests that would make a good test suite. I can't quite put a number or anything. Okay, should you be writing thousand tests? You might be writing thousand tests that really aren't useful, or you could be running one single test that's really useful.

So I recently I had to do this. I maintain USB or IP driver, and we have had some security issues that came up in that driver. Then I fixed several of those. Then I was sitting down and looking at it and I thought, "Is there a one shell script I can run that could exercise all of the different paths?" and it took me some time to come up with it. I was kind of sitting there and I'm going, "Okay. This is what I do when I'm testing this." So, "Okay. Can I somehow put that in an automated form and then put it in a shell script?" which I did.

So now I run through that test and that actually finds a lot of things, like for example, if I go do some kind of action, like attaching a device. That's what the driver does, attach the device. Can I detach it, or can I attach? You have to think through the process of things that could happen. For example, it's in particular state. So you have to have – When I'm writing a test I kind of have this state transition in my head. I might not actually put it down, but I kind of go, "Okay, somebody could come in and ask for the device that's already in use." What do you do in that particular case?

So you kind of have to think through and then automate as much as you can. Obviously you cannot automate everything. So we have –As recently testing, I'm doing a feature on the media, resource sharing feature, recently. I have a batch series out for review. So I was looking at that patch series and I'm going, "How do I test all of the exclusion cases?" So I came up with a set of applications I can use. So I'm running all the manual test, because a lot of these applications I'm using are not low-level enough that I pretty much have to have their gui-based or you need to kick off streaming and such. It takes a manual effort.

So now I have a document. While I was doing that, I wrote this tablet form of if the device is in use by this application, can the other application use it or do you see busy? So I have this table sitting there that I need to go back in the next few weeks or a couple of months to see how I can automate it. So you always start with a manual testing and then say, "Look at the manual testing," and then say, "Can I automate this somehow using the tools I have, or is it possible I can automate?" So these are all the processes I go through anyway when I'm doing work.

[00:29:14] JM: What about security testing? How does that fit into your workflow?

[00:29:19] SK: So security testing is – Like for example, "Hey –" I mean, in terms of denial of services kind of security, kind of problem, right? So if you could get a system to a state it's not responsive. That in itself is a problem. So security testing spans a range of general kernel testing itself. The specific aspects of – For the driver I was looking at, anytime a user can get a system into a state where it's not responding or it panics or crashes, those are the security vectors you have to look at.

Networking, for example, networking, it involves getting packets from the user space for example. So can somebody send you enough ending packet into data, into the kernel via a network packet and how does the kernel react to it? That's one angle. So you have to kind of think through that. There is a lot of effort going on in the kernel right now fuzzing. We go and look at the fuzzing, and there are fuzzing tests that get run, then we have a backlog of bugs, of course. So we look at those and we see how to fix them.

In some cases it could be injecting errors. We have a way of injecting certain errors to see how the system reacts, and some of those are security related, right? Meaning, could you – And

also, can a user, a regular user, can they get visibility into the kernel memory and can – Security is multiple aspects. Can a user view privilege to data? So it's the various aspects of that.

I'm starting to look at some of the bugs that we have at the moment. Then Google – Sorry, Sysbots, they're doing a huge effort on finding some of the problems. So I'm starting to look at those and see what can I do in terms of –

[00:31:33] JM: Could you talk more about the fuzz testing?

[00:31:35] SK: Fuzz testing is automated – Lot of it is auto-generated code. Okay, one example –

[00:31:46] JM: It's kind of like chaos testing, right?

[00:31:48] SK: I believe so. So let me explain one of the test they do. They'll do a user – This is one example of a test that they do. So system calls. Kernel has a bunch of system calls. So a user program will just invoke all of the systems calls and pass them, say, invalid data. How do systems calls react to it? Any driver that has outside user visible ioctls, ioctls – So call all the ioctls with bad memory, or passing in bad values, or error injection really. How does that react to it? Those are the two examples I can think of at the moment of fuzzing that's happening.

[00:32:37] JM: What kind of stuff does that reveal? What kind of problems?

[00:32:41] SK: In one of the cases, this is a bug I actually fix. One of the cases is it would react boundary kind of conditions. What would – So in one of the cases when such testing happens, if you have passed an invalid flag, for example. It goes down a kernel path. It, for example, should be able to detect that it's an invalid and not acts as out of bounce memory, for example.

I'm just throwing this out. So it's not necessarily true, but are necessarily real problem in the kernel. Say a user can pass in a range for a value, 1 through to 55, or whatever. If the ioctl comes in and then that ioctl specifies the wrong range, does kernel start to access memory that it shouldn't access and go down a path and counter panic, or a warning, or a – The driver itself running to an error, that potentially could impact the other subsystems.

[00:34:05] JM: Now, would fuzz tsting help prevent something like heart bleed from making it – Heartbleed where if I remember correct, you basically had a buffer overflow problem that was a vulnerability because people could access buffers that they shouldn't have been able to.

[00:34:27] SK: Are you talking about some of the recent problems, that Spectre or kind of thing?

[00:34:32] JM: No, not Spectre. The one before that, Heartbleed? I don't remember it very well though.

[00:34:36] SK: I don't.

[00:34:37] JM: Okay. All right. That's fine. But what about Specter? Spectre was the – I don't know if you can articulate what that was.

[00:34:48] SK: That is related to hardware, right? I can't speak to it that much, because I don't – But a part of it is exposing. We have several CPUs with different threads – This is hardware, right? So will one process have – Can one process, a user process, can access another user process? Will have the ability to look at data that's not privy to it, or it has privilege to look at. So at a higher level, that is the core of the problem, right? But, yeah, memory management is a very complex piece. I can't say I know much about it.

[00:35:42] JM: Ah! Okay. Do you know much about like testing it or how well one can test memory management?

[00:35:49] SK: We have several tests in the kernel self-test suite that we do use to test, but I don't play in that area that much to know.

[00:36:00] JM: Okay.

[00:36:01] SK: That extent of testing.

[00:36:03] JM: Right. Okay. How do you test networking for a single node operating system? Linux is a single node operating system, but it's often networking with other nodes. I assume there are certain tests you would want to run that would involve multiple nodes. What's the process for testing those?

[00:36:31] SK: You're talking about networking between two different systems, or you're talking about –

[00:36:35] JM: Yes, between two different operating systems.

[00:36:38] SK: Oh! I see. You're talking about almost – You're talking more about say you have almost like heterogeneous kind of testing that you have different operating systems running their own networking stack, for example.

[00:36:55] JM: Exactly, two different operating systems talking to each other. Is that in the purview of a k self-test or –

[00:37:01] SK: No, it's not, but I have done it in the past. I used to work on NFS, kernel NFS part for HP-UX.

[00:37:13] JM: Network file system.

[00:37:15] SK: Correct. So in that time when I was working on that, we would actually have interconnectivityfest we call them, hack fests, interconnectivityfest. So we would – That is one of the objects, being able to say it can – Can interoperability – Does interoperability work at that level? Will you be able to – NFS is just an example, but TCP/IP stack for example. So will you able to take – Could you have a client running on a Windows system, for example, and then you have a server running on Linux. So how does the communication work? NFS is, again, packets, right? Any other packet, you would take a packet, you're sending messages back and forth. Is it interpreted correctly on both sides depending on the stack? So in that particular case, you would run tests to make sure that's one of the tests you could do, doing both client and server side. If your application – Say, is a server client model and if you want – I don't know if we talk about server clients anymore as much with the cloud, but that is the kind of testing that you

would do to make sure that even when you have different networking stacks involved on either side, that you are – It's a TCP/IP protocol, right? If you follow that protocol, it should work.

Then the other aspect of that I ran into, I had to deal with, is if you have – You might have the same operating system, different architectures could come into play based on beginning and [inaudible 00:39:15]. But that's all networking – You have to worry about those aspects when you're designing the messaging format, messaging packets. I'm not talking so much about TCP/IP. For example, you came up with your own – You're using UDP as a base, but you're using your own application level protocol on the top. So you have to worry about those aspects.

Is that what you're looking for?

[00:39:43] JM: Yeah.

[00:39:43] SK: Yeah, okay.

[00:39:44] JM: You're at HP for 13-1/2 years.

[00:39:46] SK: Correct.

[00:39:47] JM: What did you learn about operating systems at HP?

[00:39:50] SK: When I was at HP, I was doing HP-UX kernel code. I worked on the low-level PCI Express drivers, a lot of the CPI code.

[00:40:01] JM: That's the operating system HP built.

[00:40:04] SK: Mm-hmm. HPs version of Unix, right? I believe I think it's probably at some point [inaudible 00:40:10]-based on. Yeah. Then it's Big Indian, because that's a big thing with –

[00:40:19] JM: You said Big Indian?

[00:40:19] SK: Big Indian. Yeah. So I64, that's one of the early I64 servers I worked on [inaudible 00:40:27] as well. My fun experiment with that was when we were bringing up new system, I64 system. We took the ACPI and then we had to make sure that Big Indian worked. I mean, Intel didn't worry too much about the Big Indian, because they don't have any Big Indian systems. So had to go through and make sure – And I did a lot of PCI Express work, where HP servers, there was large 64 way, 32 way 16 way servers, even 8 way. We supported online replacement on those PCI Express online replacement. What that means is you could yank a card out and replace it.

So all of that goes all the way from when you get – There is a way sequence of things to do. You push a button and you say, "Okay, it goes into driver. Has to go into a [inaudible 00:41:21] state and then drivers had to detach itself," and then you go into a CPI and make sure that you actually get it ready for removing the device and then you put the new device back in and it has to recognize. So it kind of spans from all the way from low-level ACPI all into the user level. That was fun. I lost lots of thumb doing that, because you had to actually pull them out. So that was fun. So that's the work I did. I also worked on shut down paths. For example, how to gracefully shut down in various cases, the system.

[00:41:59] JM: Why is that hard?

[00:42:01] SK: It's just a way some of the internals of the operating system worked in terms of – Do I remember even now? So when you have a 64 way system or when you have multiple CPUs, you have – The internals of the operating system worked, you had to gracefully shut them down in terms of various – It's not hard. It's not hard, hard per se, but coordination between different CPUs.

[00:42:38] JM: Okay. What can go wrong?

[00:42:42] SK: You might – One CPU might choose to not – Okay. This is very internal to that particular [inaudible 00:42:50]. This is not going to span.

[00:42:51] JM: That's totally fine.

[00:42:54] SK: Gosh! I don't remember to the extent you're asking, yeah.

[00:42:58] JM: Yeah, don't worry about it.

[00:42:59] SK: Yeah.

[00:43:01] JM: Samsung. What about Samsung? So you worked there. Samsung makes mobile operating systems. I guess at that point we're talking about Linux. Tell me about the world of mobile operating system development.

[00:43:15] SK: So it's very different. I came from doing large 64 way type servers and I don't think I worked on desktop as much, but going all the way to – Going to really small phones, right? So one of the things that they encounter is there is a lot of vendor driver set sometimes, vendor drivers that come in to play. There is a lot of pressure of time to market type things. So one of the things I noticed is that it is hard for them to go keep up with the mainline in a lot of different cases. For example, when they release a product – And obviously, everybody wants their phone to be working. I mean, I think I'll get mad if my phone isn't working.

So it's harder for them to move to new mainline kernels as fast as they would want to, because they have reasons to keep up with the mainline, because mainline has new features coming in. We all want to move to the mainline, because mainline Linux kernel has more features, right? It's continuously – We're fixing bugs, we're adding new features. So we want –

At the same time you also want to keep your product stable. So it is a continuous balance of the two activities, and in some cases the phone vendors want to stay on a long-term release kernel.

[00:45:05] JM: So you just continually try to merge with mainline, or you're going to get merge conflicts, or you're just going to get errors in how the code interacts.

[00:45:12] SK: You could get merge conflicts, and then also in some cases phone vendors might have their own [inaudible 00:45:20] that they're keeping internally, because they might fix problems they are encountering. They might not have time to upstream those – Send it upstream to fix this. Yeah, the merge conflicts, plus something might break.

That is one of the things I was doing when I was at Samsung. One of my activities was to make sure devices that are important to Samsung product teams continue to work on the mainline kernels. So take mainline kernels and make sure that – That helped some of the product teams move to newer releases quicker, that activity itself.

[00:46:01] JM: When you say mainline, is it mainline Linux or mainline android?

[00:46:05] SK: I'm sorry. I'm talking about Linux kernel that we do from the Linux kernel.org mainline.

[00:46:13] JM: Okay. How does managing the divergence between Linux and android work?

[00:46:24] SK: Android is – There are two parts to all these phone platforms, right? You have the underlying system, which is the system software, which is kernel. Plus, if you look at your phone and looked at system details, you'll see a kernel revision, and three dot something or four dot something. That is the actual kernel revision that they might freeze on.

Like for example, we have – Do you know how the – You probably don't have visibility to all of the stuff we deal with in the kernel space, right? So Linux's git mainline, that is continuously moving. It has new features coming in, and we releases once in about 8 weeks. Once every 8 weeks, a new kernel release comes out.

So during those 8 weeks, 7 to 8 weeks, we are doing integration, continuous integration you're talking about. So there's a two-week window between releases. This 8 weeks is part of that. First two weeks after a new release comes in, that's when all of the maintainers and we are sending patches, poll request to Linux. Linux pulls them in and then the first two weeks is that. It's just ending this Sunday for 5.1.

After that, we are fixing stuff that we found in the codebase we just did, Linux just pulled in the two weeks. So that continuous depending on the comfort level, depending on how many patches are coming in, fixes coming in that will settle in the 4 to 5 weeks. Then the next release comes out.

Very often the products, like for example phones, or Samsung, or Google, or any other phone vendors, and even some of the distros, Ubuntu, they say, “Okay, this is what we want to base our next release on, or this product, this Samsung Galaxy 10, this is the kernel release we want to base it on.” They usually always use a stable release. Greg Koah-Hartman maintains the stable releases.

[00:48:41] JM: He’s been on the show.

[00:48:42] SK: I’m sorry?

[00:48:42] JM: He’s been on our show. That was a good episode. Very interesting.

[00:48:46] SK: Great. So he probably took you through that process.

[00:48:48] JM: Oh, yeah. I don’t remember very much, but it was very deep. That’s a detailed show.

[00:48:53] SK: Right. So what Greg does is as soon as 5.0 for example just came out like two weeks ago, he’ll start to maintain that as the stable release and there are a couple three other releases that are always stable releases that keep getting fixes from the mainline. So we fix say a security problem or a driver bug and we look at it and say, “Maintainers decide that,” or maintainers and developers who fix that problem will decide whether this should go into stable or not, and then it will go into the stable. So stable releases, we have like couple three stable releases. Then vendors sometimes request and say, “Hey, we would like this particular release to be long-term stable,” which is longer length than – I think it’s like 5 years I think. Its decision is made at the time how long it should be.

So the way it works is a very new feature goes into the mainline, which is Linux’s tree, and then stable trees get fixes from there. So product teams – I mean, individual companies and phone product teams, they decide based on – They try to get their drivers and so on that need to run into one of the mainlines, then they use that whichever one has all of their content in to get their product running. They will stabilize on that. Meaning, they will pick that as they release – To do

the release. After that they will keep moving their kernel too with the fixes and then they'll keep merging stable releases.

[SPONSOR MESSAGE]

[00:50:58] JM: This podcast is brought to you by wix.com. Build your website quickly with Wix. Wix code unites design features with advanced code capabilities, so you can build data-driven websites and professional web apps very quickly. You can store and manage unlimited data, you can create hundreds of dynamic pages, you can add repeating layouts, make custom forms, call external APIs and take full control of your sites functionality using Wix Code APIs and your own JavaScript. You don't need HTML or CSS.

With Wix codes, built-in database and IDE, you've got one click deployment that instantly updates all the content on your site and everything is SEO friendly. What about security and hosting and maintenance? Wix has you covered, so you can spend more time focusing on yourself and your clients.

If you're not a developer, it's not a problem. There's plenty that you can do without writing a lot of code, although of course if you are a developer, then you can do much more. You can explore all the resources on the Wix Code's site to learn more about web development wherever you are in your developer career. You can discover video tutorials, articles, code snippets, API references and a lively forum where you can get advanced tips from Wix Code experts.

Check it out for yourself at wicks.com/sed. That's wix.com/sed. You can get 10% off your premium plan while developing a website quickly for the web. To get that 10% off the premium plan and support Software Engineering Daily, go to wix.com/sed and see what you can do with Wix Code today.

[INTERVIEW CONTINUED]

[00:52:56] JM: Are there ever conflicts between these major companies? Like a conflict between Samsung and some other phone vendor, or are they mostly pretty aligned when it comes to operating system stuff?

[00:53:12] SK: It's open source at that point, right? It's Linux kernel community that they work in. So conflicts in what way are you thinking about?

[00:53:23] JM: I don't know. Judgment, priorities?

[00:53:25] SK: So as open source developers, we are looking to work in open source, right? For example, we might wear two hats, but at the same time we're always looking to work with – We don't think of it as – In a sense, we don't necessarily think of ourselves as belonging to a one company or the other, because that's not very productive. Because you want to – You want to do advanced kernel the way that scales and improves for the betterment of large general features as supposed to specific features, or a product specific feature, for example.

If there is a product specific driver, I mean it's still a component. It will come in. That will – In the driver space, there is a driver that's needed for a phone to run that will be part of the driver and then, for example, Samsung would contribute that code, and they will maintain that.

I can't think of a conflict that we will have about a feature that this is – People are willing to listen, say, "Hey, we want this, but it needs to be generic as well." Does that answer your question? Are you looking for something?

[00:54:58] JM: Yeah. No. So what I think is interesting about Linux is it's this enormous project. It might be the most people project significance intersection, the biggest project managed by the most people with the most impact in human history. I don't know of another. I can't think of another. What has your work in the Linux project taught you about human nature?

[00:55:27] SK: It's a good question. So it's, again, a collection of individuals, which is I find working in the Linux kernel – I have worked on a lot of projects before. I started working in the open source and then contributing to Linux kernel. My sense is that people are welcoming. They would want you to do the best and contribute. They will give feedback, because you are putting yourself out there with the large number of – When you send a patch out, you do not know what kind of comments you would get back. So you have to – I have learned. It's a growth process. You have to figure out – It's a self-discovery as well. You kind of go hard way until you actually

send your code out. Then when somebody says, “Huh! There is a better way to do this?” You have to understand that you have to say, “Huh! Yes! That is a better way to do it.”

It’s a self-discovery process when somebody points out something. Would that bother you, or how would you react to it? So it’s self-discovery aspect. You should be able to take feedback and criticism, and there is always – When you look at 6,000 or 7,000 or 10,000 people, there is always a percentage of the people – A large percentage of the people want to work with you and there could always be somebody that doesn’t want to, or you’ll see different kinds of personalities, just like any community.

[00:57:07] JM: Yeah.

[00:57:08] SK: So what I found that works well for me is I like to – The way I like to work is that I would take the feedback and look at the feedback and say, “Hey, at the end of the day, we all are working to introduce the best code, the one operating principle.” So we might have different opinions on what is best. So at that point, communication becomes very important. So if you want to survive in open source community, you have to have good communication in terms of being able to explain, being able to explain the problem you are solving. Then also, other interpersonal issues as well in terms of how would you react to feedback and criticism, or how would you go back and say, “Hey, I understand what you’re saying, but this is what I’m thinking.” So be able to have that conversation, and it’s a process.

[00:58:11] JM: I imagine delivering criticism and delivering feedback is also an art.

[00:58:17] SK: Correct. Those are – Yes, obviously. Yes. So it is tough. It’s a tough process. Both are tough. Giving feedback in a way that you want – To give a feedback in a way, it’s very constructive, and it has to be viewed the same way as well. So both are tough. So it teaches you a lot about just humans, that we’re all humans. We’re not perfect and we make mistakes, and we have to be understanding of each other and then keep going.

[00:58:52] JM: It sounds like a great feedback loop though personally for like personal self-improvement of communication skills.

[00:58:58] SK: Of course. Yeah, it is. I have learned a lot. I mean – Yes, it is a – I mean, I have taken – I mean, there is one thing I remember from a while back. I took a technical communication type class. How do you interact in teams or how do you – So one of the things that stayed with me and I still use it is if you ever have a situation where it's emotional situation, then you want to not match emotion with emotion. It's like adding fuel to the fire, right?

So you have to sit back and say, "Hey, what can I say that doesn't add, contribute? If somebody is getting upset about it about something –" I mean, it's actually something to use even personal life too.
Somebody is getting upset about something, you don't want to say something that upsets them more, because at their point communications stop. You have to stop and think how could you – And thankfully, in an email-based communication, it is actually easier in a way, right? Sitting in a meeting room actually talking about code, it's a little bit different in a way than doing it or email, because you have time to –

[01:00:27] JM: Though if you make a mistake –

[01:00:29] SK: Yes.

[01:00:29] JM: If you make a mistake, everybody –

[01:00:30] SK: There is a delete button, right?

[01:00:32] JM: Well, there's delete button, but somebody probably has an audit log.

[01:00:38] SK: No. I mean, email. If you are sending email, you can always look it over multiple times. I mean, at least look at it one more time before you hit the send button, right? So you can kind of look it over and say, "How would the other person take it?" We might not think about it.

So email gives you that little bit extra time, I'm saying, email-based communication or that little bit of extra time to think about it and say, "Hey –"

[01:01:10] JM: Definitely. I like how Google and Gmail recently – Now you have an undo button after you send the email, which is – I mean, it only lasts for like three seconds or something. But like if you actually make the emotional mistake, the undo button, you're like, "Oh! Okay. Okay. Undo. Undo. Undo."

[01:01:27] SK: Right.

[01:01:28] JM: You got a three-second window.

[01:01:29] SK: Or it could be the emotional mistake or it could be something. I have done this. I would just be thinking something and then I just come up with, "Oh gosh! This is a real problem. I mean, this piece of code," and it could be mine. Then I would just respond in Russian and then go, "Man! I should have thought about it a little more." It could be not just emotional. It could be something else too. You're not thinking through and you're kind of thinking – Going down the wrong path and you kind of just like, "Oh okay. This could happen," and they go, "No. That can never happen."

But now that – So that is. Then another aspect I found – I mean, I have done mentoring within Samsung and I'm used to mentoring at HP as well. I have been on both sides. I mean, I mentored and then I was a mentee as well. So I learned a lot in both aspects. So one thing I found is open source is great place to do collaborative work. I mean, we're doing so much collaborative work, but it is always difficult for somebody to get over the apprehension and fear of making their work public. So that is a journey you just have to make if you want to be. You have to want to do that, because it's so much easier to not have to do that. It's so much easier to work on a project that isn't open source and you can do the work you do and you have a few peers that look at the code and then it goes in, right? But it is a journey you have to make when you are setting out to be vulnerable to have your work critiqued by lots of people.

I see that in the process. I have mentored lots of engineers, and it's a process for them. It's an apprehension. First, they don't know where to start, because – I mean, I struggled with it too. Like for example, I told you, I don't know much about MM, or I don't know much about scheduling. Linux, it's a huge project and there is so many areas. Not one person can be – Nobody can be an expert at everything. So we all have our pieces that we are good at.

Then if you look at a subsystem for a couple of years, it moves, that is hard to even keep up. That's how much development goes on. So what happens as a result is there are lots of smart engineers. It's hard for them to kind of wrap around where do I start? This is a huge cookie. Where do I start biting from? So that's the first thing I noticed.

The second thing is getting over that apprehension. What if I make a mistake? How would people react to it? Am I going to look dumb? You have to go through that process. So, yeah.

[01:04:34] JM: Last question, how has your work within the Linux community changed your perspective on computer science?

[01:04:43] SK: Computer science – Well, the human impact of technology, or just open source and Linux is mind-boggling to me. We have changed in the last just 10, 15 years. It changed our lives so much that the way we communicate has changed. The way we interact with each other has changed. The way we conduct a business has changed and it touches our lives in ways that we don't even know, right? Often time, I would talk to people, people that aren't familiar with the technology or don't – I mean, they use their phone, but they don't really know what that means, right?

So I think I am – Then the whole AI and deep learning and all of these are based on computer science. So it is amazing to see in the last 20 years how many lives it is touching. So I have always thought that computer science is a combination of it's an art and a science, and there is a lot of creativity involved. So it's a different kind of creativity. You're not painting a picture, but it is similar to that. It's a creative process. So the rest of the – It's one thing to write an operating system and how it gets used is something that continues to amaze me.

[01:06:23] JM: Shuah Khan, thanks for coming on Software Engineering Daily. It's been great talking.

[01:06:26] SK: Thank you for the opportunity.

[END OF INTERVIEW]

[01:06:31] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show

[END]