AN • INTERVIEW • WITH Warner Losh part 2

by Allan Jude and Benedict Reuschling

BSD Now Episode 362, 2.11 BSD Restoration. Recorded for August 5, 2020.

Benedict Reuschling and Alan Jude speak with Warner Losh about Unix history and its interesting tidbits. They discuss his 2.11 BSD Restoration Project, of which the Unix Heritage Society is part. And what's that thing he wrote called devmatch? We hope you'll enjoy it all! Part 1 appeared in the July-August 2020 issue.

REUSCHLING: Let's return to the Unix Heritage Society you mention in your blog and talks—what is it exactly? How can people participate?

WARNER LOSH: The Unix Heritage Society is sometimes called TUHS because the full name is a mouthful. They preserve the different Unix artifacts that have existed over the years inside Bell Labs and also once Unix was out in the world. TUHS has built an archive that's freely available through their website, and they strive to not only preserve the artifacts of Unix history in terms of code and binaries, but they also seek out old troves of papers about Unix. For example, when someone retires or dies and the family donates the person's papers, they scan them and make them available. Anyone who is interested can discover what it was like when some of the historical developments were actually happening, and read about them in a larger context.

One example is a group of papers about networking at Bell Labs that aren't TCP/IP. There's something called data kit and it represents a mindset at the Labs. They were a phone company, and they had a circuit-switch mentality because that's how phone calls are created. You create a circuit from the originator to the other side, and it's hardwired the whole time. So, their networking technology—their data-networking technology—reflected that. You would create a circuit, and then you would send data over the circuit. And that's a very different mindset from the packet-switch paradigm that we have today and that ultimately wound up winning. In the Unix Heritage Society collections, you can read about this and dozens of other historical events or mindsets or cultural happenings.

If you've got bits of lost history, or anything that's old and Unix related, then that's potentially of interest. There is both a public and a private archive for TUHS. William Toomey is the person who started it, and he maintains a private archive of early commercial releases on which the copyrights have not as yet expired, but he archives against the day that he can release them.

So even if you have something that can't be widely distributed, you can still deposit it in the archive so that it won't be lost to history. There are different ways to participate, but, generally, being on the mailing list, taking part in the conversations, following the different discussions, and participating in projects that come up every year or two as new artifacts surface are the primary ways most people get involved.

JUDE: That's really interesting. You just wonder how many things are out there but are lost or just undiscovered and how we can make sure more of the stuff we create doesn't end up getting lost.

LOSH: That's right. There are also other efforts. Bitsavers goes in and scans documents. They look at old computer manuals from DEC and Sun and other companies, and they'll scan those—particularly the ones that help people write simulators. They also preserve different magazine articles, and recently somebody donated all the early DECUS newsletters. DECUS is a DEC users' group that held symposium conferences back in the day, published proceedings from them, and produced newsletters that they would send out monthly to the members, talking about different developments.

It's a good thing, because although we have received the wisdom from the past, we don't always know how that wisdom accumulated or what bits of knowledge were retained or discarded. Discovering that is really exciting; at least it is to me, and given the popularity of some of the subjects I've written about, maybe it interests a few other people too.

JUDE: I think we need to give more consideration to preserving the history of not just the work we're doing but more of our thinking around it because nowadays people don't write papers and produce proceedings quite as much as they did before. A recording of a talk from BSDCan or whatever is good, and it is something we're starting to preserve, but it doesn't necessarily always get into why things were done a certain way—which could be interesting questions down the road.

LOSH: Exactly. When you read a paper, you have a different audience from when you're giving a talk. And the talk has a time constraint, and you're talking to people who have a wide range of abilities and interests and so you try to keep the talk interesting and relevant in that context. You put different things in it than you do a paper where you want to say, I did this, this, and this, and you don't want to leave anything out—even if it bores your reader a little—because some reader, sometime in the future, will want to know—is that why this crazy thing is going on now? I much like the approach of preserving all the different forms of media that are produced.

One thing that worries me is we have mailing list archives, and we have forum archives—kind of. But the forums tend to be more ephemeral and come and go. As a result, a lot of the thinking gets lost. And it's a slow erosion over time. If you're talking about how to set the jumpers on an iCOBUS network card, nobody's really gonna think to preserve that. But if you are trying to recreate a FreeBSD 4 system and realize, oh, I need period hardware, or I need a network card, or, this is all I can find, then that sort of thing suddenly becomes more interesting. And maybe a FreeBSD 1 system would need that, whereas, with FreeBSD 4, everything's plug and play. But that's the kind of thing that you might not think of when you discard it. You might think that's unimportant and that you don't need that—but others might.

JUDE: That's interesting. When I was working on the encrypted boot loader stuff, one of the papers I spent the most time on was getting into how FreeBSD boots. You recently wrote an article on how you actually basically reboot or shut down the machine.

REUSCHLING: Properly.

LOSH: Safely.

JUDE: So, what is the proper way to restart a Unix machine?

LOSH: Well, on modern systems it's like shutdown. Shutdown will make sure everything gets shut down and will try very hard to make sure all the dirty buffers get written out. But it will also bound its efforts so that it won't try forever to write things out, and so the system will actually reboot. It's kind of unsatisfying. You can't just flip a machine off. On some systems, you can type synch and then turn it off immediately, because the synch is synchronous. Linux has a synchronous synch. When synch returns, everything has been written. If the system is otherwise quiet, you could turn it off, for example.

BSD just schedules the writes. And there are some complications with UFS soft updates where, when a buffer is written, it will re-dirty itself so that it gets written again when other data is written out, and so that it can present a consistent view of the media in case the system were to crash while everything is being fleshed out. That's one of the reasons why—if you're watching the shutdown—you'll sometimes see, synching, dirty buffers one, one, one, thirty-seven, twenty-eight, fifteen, two. You might think—what on earth would cause that? Every process has been killed, why—where do those thirty-eight buffers come from? And it turns out that it's the soft updates code, it's that there are hidden dependencies, and when that one thing writes out, all those dependencies become writable, and all of a sudden you have all these dirty buffers you got to flush out to disk.

On older Unix systems, there was no reboot command. There was not a way to even stop the kernel apart from the power switch. If you wanted to synch everything out, you'd type synch. And you would type synch three times. The first one had the effect, and it would schedule all the IO. The other two were an analog delay loop that the programmer was executing to allow time for the stuff to finish before you turned off the machine.

That thinking got instilled in people. And then reboot got implemented that was moderately safe for some definitions of safe. The first version I could find flushes the files out, waits five seconds, and says, okay, we're done. Which isn't exactly safe. Usually it's fine, sometimes it's not—you know—if there was a lot to flush out.

So, it's useful to understand how this evolved—why somebody said to type synch three times. Maybe you'd say, I'll just put semicolons behind it so that I only have to hit return once, you know—I'll be smart about it. Well, being smart about it doesn't really help because having the individual commands were what made it interesting. And you get all the apocrypha that accumulates around it. Well, the first synch schedules it, and the second synch blocks until the first synch is done—was a common response that I saw in feedback to my blog. And there may be some system somewhere that does that. I didn't examine all systems but I looked at all of the BSD kernels I could find and none of them do that. I looked at Linux kernels and none of them do that. I looked at all the System V kernels that I could find a copy of and couldn't find that in any of those either.

So, it's just a myth that seems to have sprung up on its own because nobody waits for all the current dirty buffers to finish before scheduling the rest. All they do on the synch call is literally lock all the buffers, schedule them for IO. Linux will then say to have a barrier and wait for all of them to complete. That was decided a long time ago. Most other systems don't do that. Although there might be some that I'm not aware of that have that as a behavior. There's a lot of diversity in this area because people were trying to make it reliable and a lot of people hacked on it and a lot of solutions came and went in a vacuum and nobody knew what anybody else was doing because all of this was commercial and proprietary.

It's only now that the sources have leaked onto the Internet. You can find these sourc-

es, and you can actually go and study what's going on with the different systems. And one of the difficulties is that some companies were better than others about keeping the source tightly held. For example, you can't find HP sources until it became System V. With earlier ones, you don't know what was going on. If you wanted to study these systems in the original and from the original sources, it could be quite difficult because you would be left with decompiling all this stuff, and that level of effort is beyond my level of interest in the topic.

REUSCHLING: There's a point where you just accept where it is and move on.

LOSH: Exactly. I have the man pages, and what's in the man pages is all I can know.

REUSCHLING: Switching gears a bit, can you tell us a little about Devmatch, what it is, and if you plan to do more work in that area?

LOSH: Yeah, Devmatch is a notion I've had for a long time that I didn't get a chance to implement until more recently. And it started with an observation that we have a bunch of drivers in this system, and every single one of them has these tables that the prog routine will go through and say, does my plug & play information match the first entry? No, does it match the second entry, no, and so on. And I thought, what if the driver could encode a pointer to the table and some metadata about the table's format and put that into the loadable driver? Then you could write a program or leverage an existing program.

Well, if I augmented it to go through and grab this metadata and use that metadata to walk through the tables that were there to create a master table so that when an event comes in you can look at it and go, oh, that event for this kind of device, that driver over there, or that module over there says it has a driver that will accept that device.

For a number of years, Hans Peter had a USB-specific version. He would go through and he would parse, he would put the USB tables in a particular ELF header, and he would parse that out and generate devd rules to load all of the drivers. Starting in FreeBSD 11 that became default. In fact, the period from FreeBSD 10 to FreeBSD 11 is the only time the FreeBSD kernel has shrunk in size. And the only reason that it shrank is that Hans moved all of the USB drivers out of the kernel and put them into devd rules so they would load on demand when you would plug in the different devices.

So, I did that. USB worked, PC card worked, pci kind of worked. It's mostly there now. There may be one or two missing pieces. I haven't done an audit recently. All my work was focused on loading it right after boot. You would have to know to load everything to mount, route, and start, but everything else could be dynamically loaded as part of the boot process.

Manu took that a step further for embedded, and he's run a little bit of code that works with FDT but would be easy to generalize to other things like pci. The hard part was building the code that walked through the binary data structures that are in the loader.hints file that KLDX ref generates. So, we could do something similar for pci, nobody's done that. That's something I would like to find people to work on.

I did a lot of boot loader stuff about the same time I was doing this to get in Lua, and by the time I was done with Lua, I was kind of done with the boot loader for a while. I've not pursued getting this into the boot loader, but that's one way we could make the system even better because then we could load almost every device driver that way. In fact, you can—there's a way you can look at the pci device to know whether or not you need to load it.

And so you could load just a few of them in the boot loader, just enough to get the route filesystem going and go from there. That logic might be a little bit complicated, but it is possible to do. The information exists. And also walking through different binary UEFI data structures is a little bit tedious and time-consuming. Another reason I've not tried to pursue that is you can find the device booting, and you can find the path to it and then find all of the devices along that path and know what to load.

But you know, making that all happen requires more focused effort than I have right now for this project. When I'm doing FreeBSD work, I want to be as focused as possible. So if anybody's looking for a cool project, a cool Summer of Code project—it might be about that size—you can contact me or contact Manu, we can set you up with what you need to do to take the next step in this. So that's what's going on there and what I plan to do about it.

Twelve and thirteen will have about the same level of support unless somebody shows up quickly. I don't think a lot is going to change, but you never know. Somebody else might have their Covid project or turn this into their Covid project.

JUDE: For a lot of devices like network cards and so on, it seems like devmatch makes good sense, and you can decide to load those later and not compile them into the kernel.

LOSH: Yeah, particularly when you have a lot of choices. For storage drivers, there are not so many choices these days. If you have all of the legacy support we had—then there are. And bringing those in on an as-needed basis would also help the kernel size. One of the things I wanted to do with this was to have a more minimal kernel so that we could improve boot time, maybe make it possible to load things on slightly smaller devices, although that hasn't really been a focus for me in a number of years. You know having the ability to do that, having that flexibility to do that, is a good, useful tool that we should finish building because people would use that tool if they had it.

JUDE: Yeah, and just in general, from a security perspective, it's better to have all the code you're not using not loaded.

LOSH: Well, the most secure code is the code that's not on the system.

JUDE: All right, well, thank you very much for taking the time to talk to us. Was there anything else you wanted to talk about before you go?

LOSH: I think I got everything off my chest that I had hoped to get off my chest. I guess one last thing is that if you are interested in 2.11-BSD project, contact me. There are things to do that people can help out with that would make it a better recreation.