AN • INTERVIEW • WITH Warner Losh

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 Death Match? We hope you'll enjoy it all! We present Part 1 in this issue and Part 2 in the next.

REUSCHLING: Welcome, Warner. First, for the people who don't know you as yet, would you tell us a little bit about yourself and how you got involved with Unix and BSD?

WARNER LOSH: You bet! I have been involved with Unix since my college days. I started on a VAX-11/750 running 4.2 BSD back in the early 1980s—it was 1984, I think. And I've been a BSD user for a long time, and a FreeBSD user since the company I was working with ported our software to FreeBSD 1.0 GAMMA as Jordan Hubbard also happened to be a client of ours.

I've done a lot with FreeBSD over the years. I was a security officer for a while and have served on several Core Teams. I've done PC card, CardBus, devmatch, and some general storage things lately having to do primarily with optimizing flash performance for my company's video streaming servers that you might be familiar with. I work for Netflix, and they graciously let me optimize the FreeBSD storage system.

REUSCHLING: So, we can blame you for a lot of good things?

LOSH: Yes, but if your set-top box doesn't play immediately, don't call me. We have a lot of good people for that who are better at troubleshooting those sorts of things than I am.

JUDE: You're such a BSD history buff! I look forward to talking with you about the project you're working on, which is apparently older BSD than the first BSD you started with.

LOSH: It was actually released after, even though it has a lower number.

JUDE: Right, it will be interesting to hear about that. Recently, you spoke at FOSDEM about the early history of Unix, and then later gave a related talk at BSDCan. Do you think Unix history matters even more today?

LOSH: I think Unix history matters for several reasons. Unix was an innovative system in its day, and it drove a lot of very interesting innovations, some that people seem to forget about and then rediscover. And by knowing and understanding history, you can understand some of the early issues and problems with different approaches, and you can also better understand the larger context of how we got to where we are today and why we do certain things and don't do others. And history adds an understanding of why there's a philosophy of Unix, and yet there are some things that don't quite fit the philosophy, like, for example, sockets and how that came about and how it evolved and how it how it turns out people actually tried to make networking stuff fit just a normal file descriptor and found problems with that. By understanding all of those happenings, you can understand why we have some of the things we have today.

Also, by looking at the early Unix—if you're a kernel hacker, looking through early Unix code—it's a lot easier to approach, a lot easier to read, a lot easier to understand what's going on. And in a lot of ways, the core algorithms that were used at that time are still present in the kernel today and modernized mostly for core preemption for faster CPUs and different technology. It'll help you understand what's going on, which will give you a jump on understanding the code if you go to read a modern BSD kernel.

JUDE: It's interesting how many of the concepts we have today came from Unix and basically haven't changed. Like pipes. Basically, every operating system has pipes now, and it wouldn't make sense not to have pipes. If it wasn't for Unix, how would we have gotten any of those things, even those that aren't really related to Unix anymore? So many core concepts came from Unix.

LOSH: Exactly. Pipes were this thing where people thought well, yeah, we'll try this, it might be interesting to have a bunch of small tools to string together on these old PDP-11s that didn't have much address space. The tools had to be simple because that's all you had space for, and that notion has carried over since then into a number of other operating systems. It's quite interesting.

REUSCHLING: Oh, yes, especially if you have a history background and know where it originally came from and how it was implemented. I teach a lot of students every year and the Unix history sections are unfortunately often where I lose some of the students, not completely, but some fall asleep. Do you know a good way to teach students Unix history without it being boring?

LOSH: I don't know if I have a good way of teaching students without being boring. I'm not a teacher. I don't have students in front of me so I don't know if these things will work. But one of the things I do when I'm just talking to people at conferences or over a beer, or whatever, is relate how the different historical developments fit together. If you read the history of the quarter century of Unix, you'll get a good sense of what happened when, but you won't know something like the people who were developing it had a friend at Bell Labs who was feeding them new tapes—kind of on the sly—so that they could keep things up-to-date. There are personal details that make it all very interesting, and those aren't always public.

Some groups like knowing that Dennis Ritchie spent some time at Berkeley, and that Unix got so established at Berkeley because Dennis and Ken spent time there. I think I may have just misspoken, it may have been Ken that spent a semester on sabbatical there!

Knowing that Unix is more than just the technology, that Unix is also the people and the connections between the people and how those played into how Unix spread and was adopted is extremely interesting. If you can help people relate to how the different pieces fit together, more than just the recitation of facts, that seems to work better. I know it works better in my talks with adults. I imagine it would work well with students.

JUDE: I think some of the most interesting parts of Kirk McKusick's talks are the personal stories. There are just so many interesting bits that make it interesting. And I think another thing is that to really appreciate Unix history, you have to first understand some of Unix so that you can then appreciate the elegance of the way it was developed and also how it so easily could have gone very differently if there were other people involved or if Person A hadn't met Person B or even if they both hadn't been at the same place at the same time.

LOSH: Exactly. There are a lot of person-to-person situations, and there were bakeoffs. One of the reasons 4.1 turned into 4.2 BSD was that there was a bakeoff between BSD Unix and VMS. And they did a bunch of dumps of performance data to see different workloads so that 4.2 BSD could be improved. A lot of things that they did in these bakeoffs was to spur Unix development, to make it faster, better, more robust. In the end, Unix wound up winning because it was faster, better, and more robust for the things they needed.

JUDE: To Benedict's point about teaching, a lot of it is about making it matter to people. The other part is looking at just how many bits of history we've already lost, whether that's when somebody had to rewrite a program based on only the disassembled original binary or that a tape has been lost forever and so on. It makes us think about the code we're writing now and make sure that we preserve it in a way so that in 30 or 40 years from now when somebody wonders how we did that, not only will they find the source code, but also the things we don't always think about, which are the tools around it.

For instance, to be able to build Free BSD 4, you kind of need a Free BSD 4 system as you're not necessarily going to be able to just compile it on whatever computer you have 30 years from now.

LOSH: That's true.

JUDE: We might have to think about preserving more than just the source code history from GitHub or whatever.

LOSH: Yes, having binary artifacts is important so that you can go back and recreate things. It's quite useful to be able to run the original binary even if you have source. Some of the early Unix source is written in Dennis Ritchie's style, which is pretty fancy, and a lot of modern compilers just do not like it at all. If you spend a lot of time and effort, you can find all the right switches to make it swallow that code, but there have been subtle language changes, and not everything works if you do that. Having the original binaries and emulators that can run those binaries is quite important for recreating history as well, should you want to do that.

JUDE: Yes, and speaking of recreating history, tell us about your project to recreate BSD 2.11 I guess patch-0.

LOSH: You bet. For those who don't already know, you know it as 4.2BSD that was in the fourth series from Berkeley. They did a 1BSD, a 2BSD, and a 3BSD. 1BSD and 2BSD were for the PDP-11; 3BSD and 4BSD were for the VAX.

Berkeley tried very hard to run away from the PDP-11 versions, but they were so popular that they wound up doing several releases. And we have those releases. We have copies of the master tapes, but it went from 2 to 2.8 for some reason that I don't know. I think it was because it was the 80th tape. There's a BSD2.78 which has notes around it saying there have been about 70 tapes released. So maybe that's why, maybe not. But there's a 2.8, a 2.9, a 2.9.1, a 2.10, 2.10.1 and we have all of those releases. And 2.11 went out through USENIX, not through Berkeley. USENIX managed all the tapes, and there was actually some data corruption on the early tapes. Actually, the corruption was not on the tapes but on the system that made the tapes, so there were blocks of nulls where they should have had the source code to the games.

Nobody preserved the early tapes. USENIX said, oh, there's some corruption, let's come up with a new thing. And so, they remastered the tapes and the earliest one that survives is at patch level-195, which is 18 months or so after the original release. So, all of the changes in that period were preserved.

Then we thought, oh, you can just run it backwards, you know, un-apply the patches—as we're all used to modern patches, you just do path minus r, everything's cool, right? Well these patches were posted to Usenet in a number of different formats. And when they were posted, some were in a style like—run this shell script, or extract this tar ball. And when you did that, the shell script would remove files, or the tar ball would overwrite files. We don't have any extra context to go back to. We can't run the sausage mill backwards to make pigs like you can with most modern systems.

Every so often I would ask people, hey, does anybody have an original tape? And people would just say no. As a result, I just kind of let it sit there.

Then the coronavirus hit, and it caused me some anxiety and sleepless nights. You can either stay up and worry about things or you can distract yourself. So, this started as a distraction. I thought, well, let's take a look at the patches. What information was destroyed? It turns out that a lot of files were brought in to replace stuff that was in the previous version—so I could just grab the previous version. And that got me a certain distance, but there were about a dozen files that weren't quite right, didn't compile, didn't work.

I was then able to assume, well, they got this from 4.3BSD. And for about half of the files, I could get material from 4.3BSD because there was an updated 4.3BSD and the changes were bigger. So, okay, now I've got that.

But one of the things that they changed from 2.10 to 2.11 was the archive format. There was an old PDP-11-specific archive format, plus the new, modern archive format that they decided to switch to because it would make things easier. They could do more things on the VAX, and cross compile and take it to the PDP-11 because the PDP-11s were really slow machines in comparison to the VAX.

So, they did that, and I was able to pull in that code and merge it with the BSD2.10 and the BSD4.3 code. And eventually, I was able to have a string of patches all the way back to the beginning that was consistent—and by that, I mean that all the information that's in the patches, all of the reconstructions I did, didn't conflict with anything. But that's kind of a low bar—to not conflict with anything. You can have an empty file not conflict and still it can be totally wrong because there was content to that file.

And so, I started thinking, well, you know that's an interesting outcome so far, but let's see if I can actually build the system. And I got into bootstrapping the system and ran into all kinds of issues around that. I could talk for an hour and a half about them, but a lot of the issues are kind of boring.

The biggest issue was that they changed the assembler enough so that the system I started with couldn't assemble the old system at all. I had to figure out how to use the version-7 assembler to assemble BSD binaries because the system call formats had changed between version 7 and BSD. I couldn't just run the version 7 binaries; no, that would be too easy.

I had to figure out how to do that and use the version 7 compiler to compile the BSD system calls to link that all together using a PDP-11 simulator—I wanted to do that without having an actual system. So, the first step of the bootstrap was to build that and then use the oldest 2.11 system we have—the patch level-195 system—to build a couple of things. And then I had the bare minimum tool chain I needed to generate the old binary format because one of the things that changed between patch level-0 and 195 was that patch level-0 had all the old limitations.

There were eight-character identifiers and a bunch of other limitations that they wanted to get away from. They changed the binary format so they could have longer symbol names. And that presented a problem. I couldn't use the new compiler to build binaries for the old system because then you're mixing binary format types.

Once I had the basic tool chain bootstrap, I was able to use that to then bootstrap everything else. Last night I was able to create a boot tape for the original system for the first time, and I'm testing that out. And when I'm sure that's good, I'll be able to release it.

But in doing this, I realized, oh, this piece is missing or that piece is missing, or that compiles but doesn't work. And I've gone through and debugged all of those things in the six months or so since I've had sleepless nights. I'm getting very close to being able to announce, hey, here are the tapes, you can go try for yourself—these are the setups you need to use with the simulators, and here's the source code.

Once I have that, I want to make each individual change publicly available on GitHub someplace so that people can go through and look at the history and understand the history of the system. There were a lot of developments and a lot of innovations that happened during the patches that are really hard to access now. Getting at each individual piece is difficult, because they are in so many different formats, and there's not a convenient way you can browse through history. When looking at history with GitHub's history browser or the

GIT command's history browser, I can find where things happened, and it becomes a much more accessible history. Then it's not history that's locked up. I liken it to having a place people who are interested in this can come and look at it and be able to then understand what's going on and see the progression. And they can see the interesting tricks—the PDP-11 has a very small address space. It's a 16-bit machine. It has a 64kilobit address space which through MMU Magic doubles, you have 64kilobits for code and 64 bits for data. And then there's some overlays you can do to swap code in and out, kind of a precursor to dynamic paging that you have in 32-bit systems.

Looking at those techniques, you can see ways to make code fit in smaller spaces and ways to simplify the code that doesn't sacrifice functionality and allows it to fit in the space constraints available.

So that's why I got started on this project. Plus, people told me I couldn't do it, it wasn't possible, there's no way I'd succeed, which also is a motivator.

That's the BSD2.11 project, and if you're one of my Twitter followers (I'm bsdimp on Twitter), you can see the little teeny, tiny, incremental steps of progress, or if you read my blog, you can read the bigger chunks of progress. I don't know when this discussion is going to be airing or read, but in the coming weeks, there will be pointers to the posted artifacts. And when the GitHub stuff is done, that'll come up.

JUDE: When that's done, you'll have each of the patches as a GIT commit?

LOSH: Yes. Each patch is a GIT commit and was sent to a Usenet news group, and each of the patches has a header on it that will be the commit message that describes what the patch does plus any interesting commands you need to run. Also, as part of this, I'll have a series of scripts to recreate each of the patches.

To go backwards, it's a lot of manual work so I automated it. You know, I got back to patch-50 and I found a problem in patch-175. It stuck me that was 150 patches I'd have to redo by hand. Well that got old in a hurry and so I automated it and I automated the bootstrap process.

Now, with just a couple commands, I can go from historic artifacts to a tape that has the original—my reconstruction of the original 2.11BSD—on it. And that allows me to go almost to the end and find, oh, I'm missing the boot loader for this type of hardware that I know was in the original release—let me go find it. But let's just regenerate it all so I don't have to sit there and type. I can regenerate it and go get some lunch or grab a beer or whatever. And I have been publishing that to GitHub as I go along. And one of the things I'll be adding to that will be a set of scripts that will go forwards (to augment my scripts going backwards) all the way through the current patch level which is patch-469. And you could go all the way through patch-469 which was released earlier this year and be able to look at things and bring patches forward.

I also hope to create a second set of patches that are just pure patches. If I create a series of patches, then you just download the patches, you run the patch and maybe run a couple of commands to go with it, because even though there's a build everything command in 2.11BSD, when the binary format changes, you have to build the assembler first and install it, and then you have to build the AR program and the loader, and you have to do that in a particular order.

Then you have to build other things, some with old stuff, some with new stuff, so that everything winds up working and you don't break your system and you don't wind up with a system where you can't compile anything to fix it, so you've got to lift something off of tape to restart.

There is a group of people who use what's called a PI DP-11. People have taken historic PDP-11 front panels that you'll find on say a PDP-11/70 and created some glue software so that you can put a raspberry pie in it and interact with it like you could the old PDP-11s. Some members of this community are potentially interested in helping to create the patch chain all the way up to current because—you know—they've had to do this over a span of 30 patches. Since the tape they have is for patch level-430, and they need to go from 430 to 470, that's kind of a pain when you have to apply each one and it's not just patch and rebuild. It's the concept that remains of using scripts or removing this or moving there or building it in this order.

There are a number of people who have created what I'm talking about, going all the way back to the beginning for the last 30 or 40 or 50 patches. One of the hopes is that I can take all the patches and apply the script and actually wind up with the same system that's the snapshot of the other system, or the tape that's there today so as to validate that what I have is a reasonable reconstruction.

Because there are a lot of places in the instructions, it's not as simple as to just run the script and everything will be fine, or here are the five commands you run. They're not at the bottom where the normal commands are, but they're buried in the text. And several of the things that I thought were missing were actually mentioned in the text. Had I read every single line verbatim and remembered it, I would have picked that up, but since there were about 200 patches to go through, I did not do that. That takes a lot of time, and some of these patches are quite verbose.

It has been a kind of a labor of love as well as an interesting series of challenges, and it has tested my knowledge of early binary formats. I wrote a blog that talked about the early binary formats, and that blog explored what the format was on the PDP-11, and it also said, now that you know that, this is what's going on today. And you can take that simplified view and map it onto the more complex view that we have today where we have shared libraries. We have different sections that are treated differently—this is loaded into read-only memory; this is loaded into rewrite memory. And all of these things can be brought in by anything. And so instead of three simple sections, you can have hundreds of more sections in an executing program. A good reason to study history is to understand the current stuff better.

JUDE: Yes, and as you were saying, because the computers were so much more constrained at the time and there was only one CPU and so on, a lot of the code is easier to understand, and looking at that first can help you grasp the concept before you're looking at the more advanced versions in modern operating systems that are much more complicated because there are a lot of moving parts all happening at the same time.

LOSH: Right, understanding an algorithm that is not locked helps you understand how the locking happens in newer incarnations of that algorithm. Oftentimes, when you're first approaching it, the thing you see the most is I have to lock this, do this, unlock that. There is a relationship between the locks, and you can easily get distracted in all of that and not see what's going on underneath. You do not see the big picture because of all the things you have to do to make it perform. It tends to obscure what's going on behind the scenes.

JUDE: As you mentioned earlier with the limited address spaces and so on, sometimes a lot of those tricks maybe still make sense on smaller embedded systems where you only have 16 megabytes of memory to work with. You can't just do what we do nowadays and assume there are gigs and gigs of RAM, it's fine.

LOSH: Exactly.

JUDE: You know, a hundred kilobytes here, a hundred kilobytes there start to add up.

REUSCHLING: Yes, it's all coming back to us. It's a bit like looking at a dinosaur and realizing there are a couple of bones missing. And I try to extrapolate what they would look like from a running mammal that's living nowadays.

LOSH: Or, we have a part of the bone, and we compare that part of the bone and a modern chicken bone or a modern rhinoceros bone because the two animals look similar and say, oh, can we use that to recreate something? This part fits in that so maybe it looked like the modern version.

REUSCHLING: Yes, exactly. So, coming back to the museum part of this question, you mentioned the Unix Heritage Society a couple of times in your blog and your talks, so what is that exactly? And how can people participate in that?

(Part II will appear in the next issue!)