FreeBSD® JOURNAL

May/June 2023

arm Developer Program

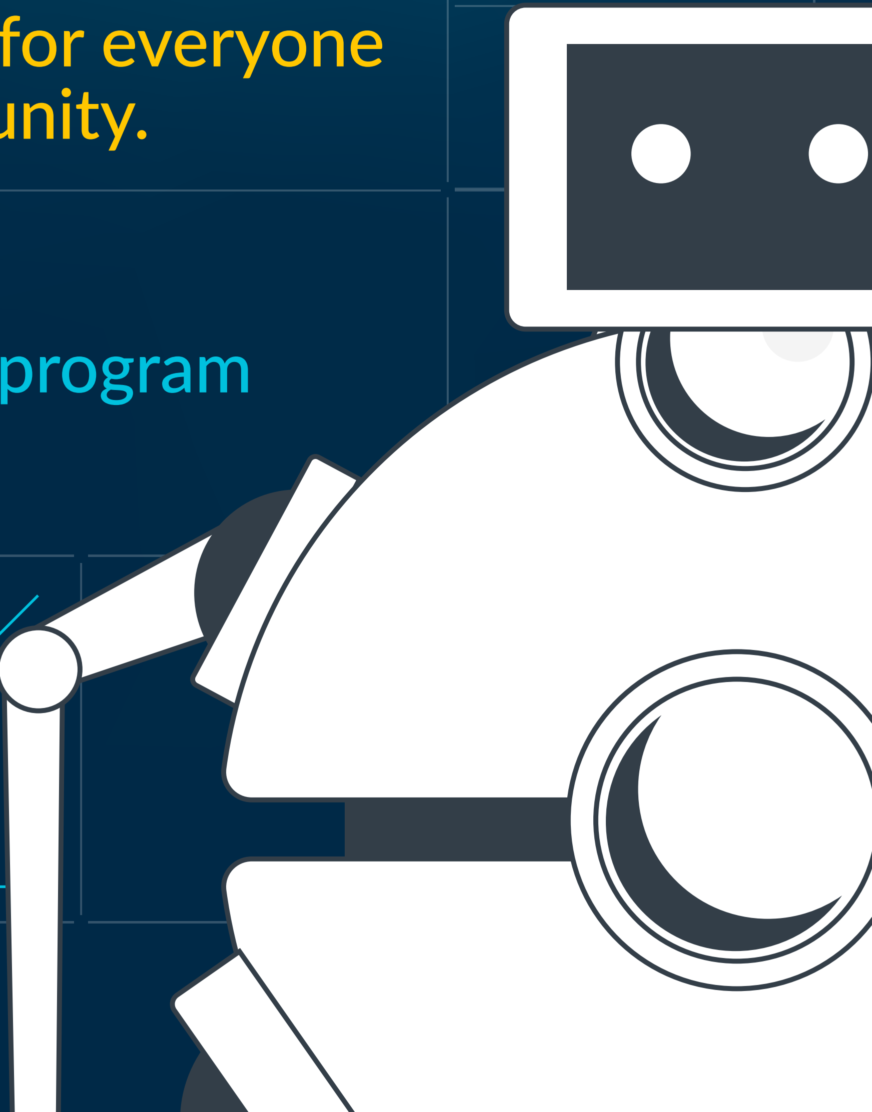
Get fresh insights directly from Arm experts.

Connect with like-minded developers.

Build on your expertise and become an Arm Ambassador.

There is something for everyone in our global community.

arm.com/developerprogram



S&W PUBLISHING LLC
PO BOX 3757 CHAPEL HILL, NC 27515-3757

Editor-at-Large • James Maurer
maurer.jim@gmail.com

Design & Production • Reuter & Associates

FreeBSD Journal (ISBN: 978-0-61 5-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-51 42 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Foundation

Welcome to a special edition celebrating 30 years of FreeBSD! FreeBSD began as a small group of volunteers building on a rich inheritance from UNIX and BSD. Thirty years later, FreeBSD has grown into a large, open source project with its own vibrant history. That history is full of individual stories with various twists and turns. There have been moments of worry and doubt (thanks AT&T), laughter (Christmas trees, who knew?), conflict, and triumph (13 major releases and counting). Each of us has played a part in FreeBSD's story to date and we, along with many new faces, will shape the next 30 years.

This issue contains a variety of articles reflecting on that long history. There are many personal anecdotes and hidden nuggets in these stories. This collection is by no means exhaustive, but it does give a glimpse into the breadth of work that many, many people have put into FreeBSD over the years. Join with us in celebrating not just the software FreeBSD has produced, but all the individuals who have contributed their time, talent, and treasure to FreeBSD's success.

John Baldwin
Chair of *FreeBSD Journal* Editorial Board

| EC11-18E |

Beckhoff through and through: TwinCAT/BSD

The robust unix-like operating system TwinCAT/BSD:

- vertically fully integrated Beckhoff solution
- BSD license for the entire value chain
- for all Beckhoff IPCs, from ARM to XEON
- implements TwinCAT real time
- easy to update through public update server
- freely configurable, familiar TwinCAT 3 Engineering
- can install own software (e.g. database, VPN, web server, etc.)

Scan to learn more!

New Automation Technology BECKHOFF

May/June 2023

FreeBSD® JOURNAL

30 YEARS FreeBSD

I'm Not from Yorkshire, I Promise!

BY POUL HENNING-KAMP

My first draft of this article is filed away under the filename "Unix_from_Yorkshire.txt", because the longer it became, the more it read like a UNIX version of Monty Python's "Four Yorkshiremen" sketch:

Running an entire oil-company on a Zilog Z8001 16-bit CPU with 4MB RAM?

Installing UNIX from 103 single-density floppies, where number 92 was bad?

1200 bits per second modems?

No source code?

Binary patching of TZ rules?

Email addresses with exclamation marks in them?

X.25 instead of Internet?

But you try and tell the young people today that... and they won't believe ya'

And they probably would not read it either, because while such tales make decent entertainment when beer is quaffed, it makes for dull and monotonous reading in daylight.

But I guess I should present myself:

I'm phk@FreeBSD.org, my laptop "critter" has run FreeBSD-current for almost 30 years, and for more than a decade I committed to the FreeBSD src repository every 18 hours on average, only to disappear when some Norvegian Newspaper had HTTP performance problems.

How to Make UNIX Not Suck

I wound up in FreeBSD because UNIX truly sucked, because I was good at making it not suck as much and was determined to do so.

The UNIX I was used to, around 1990, was bastardized by some or other mini- or micro-computer vendor, who had jumped on the UNIX bandwagon, wanting to lure in new customers with promises of "portable UNIX", and then hell-bent on locking them in, with their own "enhancements".

As a rule, the vendor support was useless. In many cases they were newer to UNIX than their customers.

There was not one of the dozen or so UNIX machines I had been root on, where I had not been forced to disassemble some program or the kernel, in order to find out what was going on, and more often than not, to patch in binary, just to make things work.

I downloaded and tried 386BSD 0.0 when it was released in spring 1992, but it was too fragile to be useful for me. I did print out a lot of source code at work, to read on the train home.

In summer 1992, 386BSD 0.1 was released and was followed within days by 386BSD "0.1-newer", and that looked a lot more promising.

To give it a workout, I installed it on a discarded "UNISYS Professional UNIX Workstation", (A wildly overpriced 486 PC with a QIC-120 cartridge tape drive and a UNIX binary license, for a list price north of $10K at today's prices) hauled it into the PBX room and set it to work logging and accounting traffic.

As a harbinger of things to come, my first `comp.unix.bsd` posting two weeks after the release was about disk-drive geometry trouble, due to the infamous 1024-cylinder limitation.

In September, my first patch was the nineteenth and final patch in the first installment of Terry Lambert's "unofficial 386BSD patchkit":

```
patch00019
PATCH: CLEAN UP SLIP INTERFACE TO KEEP FROM HANGING
AUTHOR: Poul-Henning Kamp (p...@data.fls.dk)
DESCRIPTION:

Here is a patch to clean up the interface between the tty-drivers, in
particular the com driver, and the sl# interfaces, this is not a work-around
but a genuine bug-fix.

Symptoms: after a number of "com#: silo overflow" SLIP ceases to work.

Overview of the problem: the slip interface will disregard any notice
from the tty-driver on problems (parity errors, framing errors or overruns),
which basicly means the one might as well throw the packet away right away.
Also overrun in the packetizing will go relatively unnotized.
```

As it says in the title: I'm not from Yorkshire. I'm from the unfashionable and poor southwestern corner of Sjælland, the largest island in Denmark.

But in 1992, I had TCP/IP connectivity on my home-PC.

I called the call-back security-box at work, waited for it to call back, pressed the button on my V.22bis modem, logged into the 386BSD machine at work, started SLIP there, started SLIP on my PC, and I was on the Internet via my work's connection to the Danish Unix User's Group's network "DKNet".

Life was good at 200 bytes per second, but then things got weird.

Bill Jolitz, who had ported the NET/2 sources and added what it took to make it run on a i386, did not respond to the many people who offered him help and patches, the promised 0.2 version never materialized, and the patchkit grew and grew, topping out at 138 "official unofficial" patches, 3 1/2 of which were mine.

In spring 1993, Chris G. Demetriou et al. gave up waiting for Bill Joliz and founded the "NetBSD" project, starting a new CVS-tree consisting of 386BSD+patchkit.

Then, three months later, 30 years ago, Jordan Hubbard et al. did the exact same thing, under the name FreeBSD.

Why?

I have no idea: I was on the other side of the planet, and, at that time, a very minor contributor.

To me NetBSD seemed to be more idealistic: They wanted a very portable, pure BSD UNIX, running on all sorts of hardware: HP, Sun, Digital, IBM PCs, most of which I had neither access to nor an interest in. The NetBSD crew also seemed very academic and more concerned with doing things "right" than getting them to work soon.

In all fairness: They have very competently done exactly what they set out to do, so a big hat-tip from here to our marginally older sibling.

FreeBSD focused exclusively on the i386 PC platform, which is what I had access to, so I gravitated towards the FreeBSD project where people also seemed to be "production-oriented" like myself.

In late November 1993, FreeBSD 1.0 was released, and I had enough patches in it to be listed under "Additional FreeBSD Contributors", and I even received a free courtesy CD-ROM from Walnut Creek CD-ROM some weeks later.

On February 26, 1994, almost precisely 29 years ago as I write this, Jordan gave me the 21st commit-bit in the FreeBSD project.

Your Mission, Should You Choose to Accept It

According to the CVS log Jordan did this: "… so that he can help me with the ports cleanup," and as the 1.1.5.1-tree CVS commit-log as my witness: I did make two commits to ports before I nonchalantly sauntered into src and committed a bunch of my time-nuttery patches to the kernel and NTPD.

Next, I committed an unsanitary concoction of shell and Tcl scripts that could munge a GCC 2.5.8 compiler source tree into the form expected by the FreeBSD Makefiles in case anybody else wanted to upgrade from GCC 1.39.

At the same time, I was negotiating a new job as on-site SW-engineer for a very large document imaging system for "Giro" post-cheques, to be delivered to the Danish Post Bank on a very aggressive schedule by "TRW Financial Services" from Oakland, CA.

Julian Elischer, who I knew from 386BSD & FreeBSD, worked there, and got himself a referral bonus by pointing the "DGB" project my way. That probably made me the second person to land a job because of FreeBSD, Jordan being the first, working at Walnut Creek CD-ROM.

I landed the job and landed in California where I happened to settle into an apartment in Pleasant Hill, just a few miles from Jordan and Walnut Creek CDROM. That soon came in handy.

Nominally, I was in the USA for training, but I soon found myself responsible for installing Solaris and 3rd party software on five presumably performant Sun Sparc 1000 servers with a literal ton of disks, configuring a couple of Cisco 7010 routers, an ATM switch made by Fore Systems, and dozens of ethernets to the 135 PCs running NT.

The USL-BSD lawsuit settled out of court in February 1994, with a 3-month grace period which also applied to FreeBSD 1.0.

So, we had to start over, and did, as soon as we could get the "4.4 BSD(Lite)" release from UCB, and then we had to reintegrate the "unencumbered" bits and pieces of FreeBSD 1 as best we could, just to get feature parity with FreeBSD 1.0.

Rod Grimes did most of that work and, as I remember it, he had the new CVS tree ready for us sometime in May 1994.

Not long after, I was inducted into the core team and almost immediately appointed release-engineer for 2.0, together with Jordan.

In FreeBSD, the releng@ is the designated pilot responsible for rowing the locomotive onto firm ground, and Jordan and I set a precedent that almost unlimited powers come with the job.

As release engineers, we had to write the code to build the release, pack the release into media-agnostic distribution files, the code to start the installer, the installer itself, the code to partition the hard disk, the code to extract the distribution files from all sorts of media and network services, and the code to do the magic boot-block swizzle stuff.

But we only had to do that once. Subsequent releases were a breeze by comparison, and it took over a decade before somebody finally had had enough and wrote **`bsdinstall`**.

Jordan wrote almost all of the **`sysinstall`** program, and it lived up to our design goal of asking all questions up front instead of sometime later when the answer was needed--like Windows or Solaris did, much to everybody's annoyance. What's the point of installing from a CD-ROM or tape, if you have to babysit it all the time?

However, the disk-partitioning editor fell to me to write, causing several people to argue that I should be prevented, by injunction, if necessary, from ever writing user-interface code again.

In FreeBSD, the releng@ is the designated pilot responsible for rowing the locomotive onto firm ground, and Jordan and I set a precedent that almost unlimited powers come with the job.

See the `make world` with Releng Cruises

One thing that took a lot of time and commits was "sterilizing" the release process so that no "dirty laundry" leaked from the system on which the release was built into the distribution files of the release.

For instance, a fair number of Makefiles would have:

```
CFLAGS+= -I/sys
```

instead of the correct:

```
CFLAGS+= -I${.CURDIR}/../../sys
```

We did not quite get to reproducible builds, because back then it was surprisingly common to compile transient information such as `__DATE__`, `__TIME__` and similar things into programs.

Three weeks before the release, I slammed GCC 2.6.1 into the tree, hoping it would solve some of the many issues we had, with a commit message (fe7dee47009525) making it clear that this was not up for discussion. GCC 2.6.2 came out a week before the release, I rushed the changes in, and FreeBSD-2.0 came out with a non-crummy C-compiler.

ITAR dual-use export regulations forbade export of cryptographic software without an explicit license from the US government, and we had neither the time, the money, nor the lawyers to even contemplate that, and we were unlikely to get permission to export the sources, and certainly not in a relevant timeframe.

The futility of this regulatory scheme was obvious: A copy of the relevant source files were in South Africa, from where Mark Murray made them available via anonymous FTP, and anybody in the whole world, including the USA, could freely fetch those files and use them, because the USA did not forbid import, only export.

Personally, I would just have ignored ITAR, pointing to the global availability if trouble came later, but Jordan vetoed that because, while the FreeBSD project might get away with that, Walnut Creek could not put "contraband" on their CD-ROMs if they wanted to sell them abroad—or in case anybody else did.

The single place where this really mattered was the `crypt(3)` function which is used to scramble passwords for storage: In traditional UNIX, it was derived from the DES encryption algorithm and thus the source code could certainly not be exported.

Somebody, and I am not entirely sure who, had written an ersatz `crypt(3)` function during FreeBSD-1 days, but it really was no good, and we were certainly not going to ship it in 2.0. Since nobody else had done anything, I did, two weeks before the 2.0 release.

They produced a new batch of CD-ROMs with a "January 1995" date, and distinguished the cover by giving Beastie the bright green sneakers, he has proudly worn ever since in FreeBSD-context.

I had previous experience with brute-forcing traditional UNIX passwords. In one job I found that the company policy of "letters and digits" made everybody use their license plate as a password. In a later job, I found out that a lot of people at Olivetti's research center in Ivrea used "Gloria" as a password, and having met her, I did not blame them.

First, I had to do some experiments to see what I could get away with. How much code assumed the length of the scrambled password? Up to 80 characters was not a problem. What characters could you get away with using? Pretty much all the visible ones. And so on.

I made hacking passwords a lot harder. I made the "salt" much bigger to thwart precomputed dictionaries. I designed the algorithm to take much more CPU time, 34 milliseconds on a 60 MHz Pentium, and to be very hard to implement in FPGAs.

To mark the new kind of passwords, I picked the prefix **1**, committed the code with a suitably scary commit message—and continued to the next item on the still far too long TODO list.

The resulting code, far from perfect, became known as "md5crypt". Because it used the already RFC-published MD5 cryptographic hash function, we were free to export it, source and all, and we never got into any trouble.

BSD, BSD, BSD, GNU, BSD and Beer

That is: If you ignore the license.

There had just been one of the periodic GNU vs. BSD license mail storms and just to tick off everybody equally, I revived my old "beer-ware" license and slapped that on md5crypt before I committed it.

I have never fully grokked why md5crypt was copied all over the place, but it was. All sorts of FOSS imported it, from Perl to Apache. A version with the serial number filed off appeared in GNU libc. Somebody wrote md5crypt in COBOL. Somebody produced a JavaScript version, including the underlying MD5, using some kind of C->JS transpiler.

Closed source also embraced it: Macromedia Flash used it. Cisco put it in IOS, and eventually, even Microsoft supported **1** passwords.

But as it spread, it also landed in front of lawyers doing "due diligence" before some FOSS-based company or other got bought, and some lawyers had a really hard time figuring out precisely who owed who how much beer.

IBM bought Whistle Communications, makers of a FreeBSD-based communication appliance. A polite junior IBM lawyer called me, asked for permission to record the call, apologized for taking my time, and ran me through the Very Serious Legal Questions prepared by the Senior Lawyers. We both giggled a lot.

But I also received a registered letter from Very Serious Senior Lawyer demanding that I reply "forthwith", in writing and notarized, to the following questions: Was I the sole author of the md5crypt function? Did I have full legal control of the copyright to offer licenses? Was I offering the beer-ware license to $BigCorp? If so, would it be the specific division of $BigCorp, all of $BigCorp, $BigCorp's distributors and agents, or was it the end-users of $BigCorp products who owed me beer? And if so, how much beer did they owe me, how was it to be delivered and who would be responsible for customs, levies, and fees thereon? How did the amount of beer owed relate to the amount of use and/or perceived usefulness of the software to the legal entity in question? Could other alcoholic beverages be substituted if beer was not available in the relevant jurisdiction? Were people unable to send me alcoholic beverages, for legal or any other reason, prevented from using my software? He had clearly thought a lot about it. I did not reply.

Interestingly, not one single lawyer ever asked me if I were willing to make their job easier by offering md5crypt under a standard FOSS license. I guess that is not how lawyers roll?

Finally on the 22nd of November 1994, we released FreeBSD 2.0 right in time for the dot-com boom already in progress.

Walnut Creek CD-ROM pushed out FreeBSD 2.0 on CD-ROM, aiming for December 1994, missed the fabrication deadline, edited the date to "January 1994", produced the CD-ROMs, discovered the mistake, produced a new batch of CD-ROMs with a "January 1995" date, and distinguished the cover by giving Beastie the bright green sneakers, he has proudly worn ever since in FreeBSD-context.

After some more releases, I got out of releng@ on good behavior, but Jordan was stuck with release engineering as long as he worked at Walnut Creek, because FreeBSD became a major product for them.

Critter Always Runs -current

I continued hacking FreeBSD in my copious spare time and bought a Gateway Handbook/486 so that I could spend the daily "BART" commute productively. That was the first "critter" and the best laptop I ever had, but the name was passed on to the dozen or so laptops I have had ever since, all of them running FreeBSD-current.

In 1995, RAM prices had gone through the roof because everybody bought PCs, and with only 4MB in "critter", I noticed that FreeBSD was not using RAM too efficiently. I dived into it and found a spot where the "VM revolution" had passed UNIX by.

The traditional implementation of `malloc(3)` can be found on pages 173-177 in the K&R Old Testament, and it works great on a pre-VM system where entire processes are swapped in and out. But with virtual memory, keeping the metadata about free chunks of memory in those chunks of free memory means you might page in a lot of unused pages, just to free even more memory.

Somebody between K&R and FreeBSD had "solved" that by not really bothering with reusing free memory, until the kernel refused to hand out more with `sbrk(2)`.

After some hacking around, partly inspired by a very old computer's filesystem, I came up with `phkmalloc` which was one of the first truly page-organized `malloc(3)` implementations, and it made my tiny laptop noticeably faster.

One side effect of the design was that several classes of wrong use, notably double frees and freeing modified pointers, could be deterministically detected before they created any havoc, so I added several debugging flags to `phkmalloc`, where `A` meant `abort(2) on any trouble`, `J` meant `fill with Junk` and so on.

When I set `AJ` globally on "critter", `fsck(8)` dumped core on the next reboot, and so did far too much other code over the coming years, including `inetd(8)`, `cvs(1)`, `getpwent(3)`, `BIND` and `OpenSSH`.

The Great Wizard Convention of 1998

Summer 1998 was peak-dot-com-mania, and thanks to sponsors swimming in virtual money, we were able to gather the entire core team physically, for the first and possibly only time ever, at the USENIX Annual Technical Conference in New Orleans.

Bruce Evans from Australia resisted the idea for a long time, eventually revealing that he was in fact almost totally deaf, but he was persuaded to come anyway and got at least one Wayne's World style "We're not worthy! We're not worthy!" greeting.

Bruce died a couple of years ago, but he was already a legend back then. Andrew Tanenbaum had thanked him for Minix386, Linus Torvalds had thanked him in the original Linux announcement, and we in FreeBSD were eternally thankful for his countless code reviews, patient explanations, and wry humor. May he rest in peace.

For good measure, I had proposed a talk about `phkmalloc` and it was accepted for the "FreeNix" track, which were clearly meant as the kid's side-show to the grown-up's serious UNIX conference.

I am chronically unable to remember names and faces, but I instantly recognized Kirk McKusick's trademark mustache in the front row. His picture had been over numerous articles in USENIX and European Unix Users Group's magazine over the years.

I had never mingled with UNIX nobility before, and I decided to skip foil 20, where Kirk's `fsck(8)` program topped the list of buggy software `phkmalloc` had spotted so far. But when I got there, I was so high on adrenaline, I just plowed through, until solid laughter from both the first row and elsewhere in the room stopped me. Kirk was not the only person in audience with code on my list.

All the "victims" I talked with were nice about it, and thought it was perfectly fair to show that list to document that this was a real and present problem.

Kirk turned out to be a particularly cool guy and I would get to work on UFS2 with him a few years later. At Kirk's and Eric Allman's "secret wizard party" I learned that UNIX nobility knows how to party. If the punch had been poisoned that evening, you would all be running Windows NT today.

For me personally, the highlight of the trip to New Orleans was running into Dennis Ritchie at the breakfast buffet, and talking UNIX history, timekeeping, device nodes and networking with him, for so long that we missed the first talk of the morning. Getting his blessing on what I did with DEVFS meant everything to me.

I hope someday, there will be a definitive book about the history of FreeBSD, written by a competent historian.

Disks, Partitions, Buffers and VNODEs

The `sysinstall` disk editor had revealed to me that the interface between filesystems and storage devices was fairly kludged together. For instance, the disk partioning was implemented in the individual device drivers. Hard disks mostly did, but ramdisks, floppies and optical media did not.

If you look at pre-VM UNIX, it's all neat and sorted, but the way virtual memory was added on top, made me coin the slightly derogative term "phd-ware". A pile of proof-of-concept source code dropped on the doorstep, like an orphan at the monastery before sunrise.

When I started, we had three instances of phdware in the storage-domain: Heidemann's extensible VNODEs, Seltzer's LFS, and Kirk's FFS/UFS. To his credit, Kirk came back later, and he maintains FFS/UFS to this day.

At work, I wrangled a bass-ackwards disk layout. Veritas Volume Manager forced us make 9 slices per disk (for performance) and then mirror the slices pairwise across disks (for reliability), before finally concatenating the mirrored slices into ten partitions for our image storage filesystems.

It would have been so much more sensible and much easier to setup, manage and operate if we could have mirrored the disks pairwise first, slice those mirrors, and then concatenate to partitions, but Veritas could not do that.

To make matters even worse, we had five servers but only four disk cabinets, three disks per SCSI-bus, but two SCSI-busses per power-supply.

Within a year of starting production, that particular architectural "limitation" in Veritas was the root cause that wiped out half a million images of financial transaction documents.

Fortunately, they were still available for rescanning, but production was delayed for most of a week.

That incident, more than anything else, caused me to desire and design a Lego™ style volume management system, without any kind of silly restrictions.

But before I could implement it, there were a lot of weeds to whack, which I did, and so much so that "Danish axes" became a standing joke on the core team for some years. I like to think of it as retiring technical debt before that became cool.

I cannot claim to have worked from a master kernel architecture blueprint, it was more a long sequence of "well, that could be smarter…," in many cases circling back over the same parts of the kernel multiple times to implement another set of improvements.

Finally, the core parts of GEOM were developed in 2002 on a DARPA contract under the "CHATS" research program, thanks to the young and ambitious Robert Watson, who also coauthored the paper about jails with me.

Later, I again found voids in my one-man-company's calendar for the second half of 2004 and decided to see if "community funding" could keep my FreeBSD-habit going. That succeeded beyond all expectations, manifesting in 530 commits that cleaned up ttys, linedisciplines, filedescriptors, VNODE-switch, VFS-switch and more.

Core's Dirty Laundry

But I also had another role in the project—I was on the first core team.

The core team was the nominal governing board of the project, but there were no formal definitions of any of those things, we had no bylaws, and we had three huge internal handicaps:

First, we were global, and the culture differences were very real.

In Denmark, we joke that if three people wait more than 5 minutes for a bus, they will have formed an association by the time it arrives. In other countries, forming associations and organizations were very formal events, involving lawyers, notaries with stamps, filing fees, or downright governmental preapproval.

Second, and much worse, we were all primarily very good coders, a demographic rightly not famous for interpersonal skills, and we were all used to being the smartest person in the room, building and probably also zip code. Take it from me, lynx are no good at cat herding.

Third, the FreeBSD core team grew organically on the general principle of "he does good work, pull him in!" But out of some kind of misguided respect, we never retired anybody, even if they had wandered off and we hadn't heard from them for years.

Eventually we peaked at 18 nominal members, of which much less than half could be expected to cast a vote when it came down to that, often causing the losing side to argue that no quorum was reached.

And with these three handicaps, the core team had work to do, most importantly, handing out commit bits, something we should never take lightly. In a few ugly cases, it involved taking them away again, when strongly encouraging the misfitting or misbehaving person to volunteer it back had failed.

If You Can Keep It…

A particular "personnel-issue" brought the legitimacy crisis of core@ to a breaking point, and while various minor patches were proposed, they would not, and could not, have fixed the fundamental problem: Core's unwillingness and inability to make unpleasant, but necessary decisions.

Eventually, I proposed that we should simply let the committers elect a new core team, but that idea did not appeal to people from countries with a history of less than fair elections. It would only lead to "politics", "campaigning", and "corruption" they extrapolated.

I suspect the argument which finally sold them on my idea was when I said that I would not be a candidate if an election were held, but otherwise I would cling to my core bit to the bitter end, or until they pried it away from me, in bright daylight and in public.

This would not be a hard promise for me to fulfill, as my personal life was quite chaotic. My (then) wife's mental illness had forced me to become self-employed and work from home to take care of our two toddlers.

One way or another: I eventually got my election after we had agreed to a very minimal set of bylaws for FreeBSD.

At the BSD Conference in Monterey in autumn 2000, the first core team election result was officially announced, a competent 9-person core team was elected, and the "press-release" email contained:

> Departing Core Team member Poul-Henning Kamp said, "I'm very proud of what we have done together in the Core Team over the last 8 years. The new Core and the fact that they are elected by the committers, means that the project will be much more responsive to change in the future."

In the past 24 years, the committers have taken their job as voters seriously, we are on our 12th duly elected core team, none of which have been any worse than we were on "core.0" and many of them have been obviously better.

To me, democracy was my most important contribution to FreeBSD.

And That's Enough of That…

I think I managed to avoid a copyright claim for planking "Four Yorkshiremen" in this second attempt, but it still feels fundamentally wrong to me, because there is so much more to FreeBSD's history than the haphazard selection of the egocentric memories I have laid out above.

The number of people I have mentioned by name is the least I could get away with, because there are literally far too many who deserve to be mentioned to fit into an article of this length.

I hope someday, there will be a definitive book about the history of FreeBSD, written by a competent historian, with no or little beef of his own in the pot, where everybody who contributed, including myself, gets their fair bit of attention.

Until them, to all FreeBSD Friends, past, present, and future:
Thanks for all and keep caring!
/phk

POUL HENNING-KAMP is phk@FreeBSD.org, his laptop "critter" has run FreeBSD-current for almost 30 years, and for more than a decade he has committed to the FreeBSD src repository every 18 hours on average, only to disappear when some Norvegian Newspaper had HTTP performance problems.

How Time Flies: A Personal Timeline

BY GREG LEHEY

FreeBSD is 30 years old!

Where do I fit in? I wasn't one of the founders, but I was waiting in the wings. Here's how things happened from my perspective.

Pre-FreeBSD

By the time FreeBSD came around, I had been in the industry for almost exactly 20 years. In May 1973, I started working for UNIVAC in Frankfurt/Main. I later joined Tandem, also in Frankfurt. From 1987, I worked with Tandem's new Unix systems (System V) and formed the European Unix Technical Support team. In 1990, in true "eat your own dog food" tradition, we installed the first desktop Unix machines in Tandem Europe, Intel 80386 machines with a whopping 8 MB of memory running Interactive Unix System V/386.

In March 1992, I left Tandem and became a consultant. No more Unix source access for me! In fact, hardly any Unix: I couldn't take my company equipment with me, I thought, though I discovered I was left with what may be the very last Tandem LXN system. But that wasn't appropriate for desktop use, and Unix was very expensive in those days. By coincidence, a product called BSD/386 came out at almost exactly the time I left Tandem. Only $1000 with complete sources!

Yes, it wasn't System V, but it should be acceptable. Surprise, surprise! It was much easier to use than System V. I wrote an article that was published in the June 1992 edition of the German iX magazine.

I also kept my eyes open, of course. In mid-1992, Bill Jolitz released 386BSD, which seemed surprisingly close to BSD/386, not just in name. I downloaded it, installed it, watched it crash and moved on.

1993

And FreeBSD? When did people find out about it? 19 June 1993 was really just the date when people decided on a name. I had been following developments on USENET, but especially after my experience with 386BSD, nothing made me feel that I should try FreeBSD, especially with the horrendous sums that network traffic cost in those days.

But on 27 September 1993, I was in Walnut Creek to meet Jack Velte and Bob Bruce, the owners of Walnut Creek CDROM: I had published a CD-ROM called "Applications for Unixware," and I wanted them to market it for me. While I was there, I saw somebody working on a computer on the other side of the hall. Jack told me it was the development machine for FreeBSD. I was interested, and he promised to send me a CD as soon as it became available.

1994

My FreeBSD 1.0 CD-ROM arrived! I must have installed it and tried it out, but I didn't intend to use it. But then, I found a bug in BSD/386 *sort*. Was it there in FreeBSD as well? No! And what's more, the binary worked perfectly well on BSD/386. My opinion of FreeBSD improved, of course.

1995

In May 1995, I was in Singapore working as a consultant for Tandem Computers on a project for Singapore Telecom. We were all using Tandem's version of Unix System V, but most people in the office used Microsoft, and in those days, networking was something new and strange. They appeared not to have heard of the Domain Name System, and they connected to other machines by IP address.

One day, I wanted to print out something. Over to the printer, found a sticker with its IP address on it, and printed to it. The print job went out, but the printer didn't print anything. I later discovered that it had belonged to the Tandem office in Jakarta, but it had failed and been sent to Singapore for repair, after which it remained in the office. But they hadn't changed the label, and my print job went to Jakarta.

And FreeBSD?
When did people find out about it?
19 June 1993 was really just the date when people decided on a name.

We need a name server! I found a little-used PC, installed FreeBSD, and bind on it and hid it under a desk. Presto! The zone *sing.tandem.com* came into being.

Later in the year, in October, a number of people from Walnut Creek CDROM came to visit us in Germany. I had just submitted my first book, *Porting UNIX Software*, for publication, and showed off the drafts. Jack Velte was impressed and said, "Can't you write a book about how to install FreeBSD? Doesn't need to be long, just about 50 pages or so."

That sounded like a good idea, so I set to work on it. I had had issues with *Porting UNIX Software*: the proofreaders disagreed with my wording and my markup, producing a final result that had significant errors. So, one of my conditions would be that I would submit the final PDF myself. I almost certainly started work with BSD/386, but clearly, I needed to understand FreeBSD better, and so I moved the work to a FreeBSD machine. I also wanted to make a point of creating the whole book only on a FreeBSD platform.

What was it like writing a book on FreeBSD in the mid-1990s? I have heard suggestions that it must have been much more difficult than nowadays. But no, that's not the issue: it's not the tools but the content that causes the pain. And that hasn't changed. The book went through a total of 5 editions, and in each edition, I had to rearrange things to make them sound more logical. Yes, the machine was slow—a 66 MHz Intel 486, I think—and it took a couple of minutes to format each chapter. But I devised some *Makefile* tricks that made it possible to build one chapter at a time and still maintain the cross-references to the other chapters.

1996

On 24 February 1996, I submitted the final copy for *Running FreeBSD 2.1*, which Jack changed (only on the outside cover) to *Installing And Running FreeBSD*. It wasn't exactly the 50 pages Jack had asked for: it had swollen to 200 pages of editorial content and another 100 pages of man pages, contents, and index.

Was Jack happy? Of course not! He wanted something much bigger. I hardly had time to breathe before I started on the next edition, to be titled *The Complete FreeBSD* (Jack's title). He wanted 1000 pages!

In October we had a conference ("FreeBSDCon") in Berkeley, at which I demonstrated Vinum.

We didn't make it. On 19 July 1996, less than 5 months later, I submitted what we decided was the first edition of *The Complete FreeBSD*, with only 844 pages. How did I do it? Man pages are your friend: there were 542 of them.

In this time frame, I gradually migrated to FreeBSD and spent some time working on an ISDN subsystem. This had an unexpected side effect: while working on the logging, I received a phone call. Answered it, but the caller hung up immediately.

But I had a log entry, and it contained the caller number, something unusual in those days. So, I called back and asked if I could help. Yes, he wanted to buy my house!

1997

Selling our house was a long-drawn-out matter. We had decided to leave Germany because of the climate, but where should we go? Ultimately, we decided on Australia, and the move there took us 6 months to complete.

In September 1997, I was playing around with *ccd*, a tool to concatenate disks. It didn't work well, and I thought "there must be something better." I had just returned from China, where I had taught some courses on Tandem's NonStop-UX product, which included the Veritas Volume Manager (VxVM). That's what we needed! How hard can it be?

I set to implement a clone of VxVM, which for obvious reasons I called Vinum. That brought me much closer to the FreeBSD development community, of course, and clearly it also kept me busy for some time. The development of Vinum was helped by Cybernet, who were looking for a RAID-5 implementation, so I implemented that as well. It must have been some of the most complex code I have ever written. On the day that I finally thought I had nailed it, three kookaburras started flying round the house, laughing their heads off. I've never experienced that before.

1998

In June 1998, I went to the USENIX annual technical conference in New Orleans and met a number of the FreeBSD developers. I discovered that my design for Vinum conflicted with plans for implementing GEOM.

Later in the year, Vinum was ready for inclusion into the FreeBSD source tree. I presented a paper at the AUUG annual conference in September, at which Peter Wemm, the CVS-Meister, was also present. He helped me to commit the Vinum code, my first-ever commit. A good thing, too: for some reason, probably network flakiness, the commit failed. The second time it was successful, but in the meantime, we had managed to commit an extra file: *emacs.core*, which caused general excitement.

Later in the year, Wes Peters talked me into writing articles about FreeBSD in the now-defunct *Daemon News*. We alternated every month: each of us wrote 6 articles per year, and I continued until 2004.

1999

FreeBSD has come of age! In October we had a conference ("FreeBSDCon") in Berkeley, at which I demonstrated Vinum. Compared to the meeting New Orleans only the year before, things seemed much more professional, and for the first time, we got the feeling that there were people out there who were interested in FreeBSD without being part of the project. We have followers!

2000

When did I stop using BSD/OS (as BSD/386 was now called)? The last CDs I have are for release 3.0, which was released in February 1997. My guess is that I stopped when I moved to Australia: I had to set up new machines and FreeBSD was just as good as BSD/OS.

It seems that BSDI thought so too: in March 2000, they "merged" with Walnut Creek CDROM, under conditions that I never found out. There were initial announcements that the FreeBSD and BSD/OS code bases would be merged into a single product, though there was also vocal opposition to the idea, notably from Mike Karels. And as we all know, it never happened.

Also in March 2000, I received an offer that I couldn't refuse: join Linuxcare, a company created to support Linux. What did that have to do with me? They wanted to branch out into BSD support, and they already had one employee, Ceren Ercen, who proved to be a friend of Mike Smith. The good news was that I could work from home, and I'd be paid to do BSD work.

In April, a core team member committed changes that broke Vinum! It wasn't that I had not been discussing the matter with him, but it seems that he couldn't be bothered. Much discussion, eventually coming up with Jordan Hubbard's opinion that the current core structure (the 17 oldest members, whether still active or not) had passed its use-by date. In the end, people came up with the idea of an elected core team, and I was elected to the first team ("core.2"). And gradually, the team was more international: 3 Australians, 3 Americans, 2 British and 1 Japanese.

The big technical issue of the year was performance, related to multiprocessor systems, which were just becoming a thing. To our horror, we had discovered that Microsoft had out-performed FreeBSD on web server applications. We understood why: the Unix kernel was built for single processor machines, and only one process could be in the kernel at any one time. To enter the kernel, a process had to acquire a lock called `Giant`, waiting, if necessary, until its owner had left the kernel. This resulted in a lot of processor idle time.

Fortunately, round this time (the "Dot-com bubble"), many project members, myself included, were employed by companies prepared to fund this kind of work. Though BSDI had merged with Walnut Creek CDROM, we still didn't have access to their code base. They had done some work in this area for BSD/OS 5.0, and they were prepared to let us use the code. On 15 June 2000, a number of us got together atYahoo! In Sunnyvale and hacked out the details, one of which was a project name: SMPng. In principle, we would incorporate the SMP-related BSD/OS code into FreeBSD. In the end, only Matt Dillon, Jason Evans, and myself ended up on the implementation team. I had been working with multiprocessor systems for almost my entire professional career, so I was really keen to participate. And since this was my day job, funded by Linuxcare, I had plenty of time.

It soon eventuated that we had different expectations. Matt left the team, John Baldwin joined, and Jason became project leader. He committed the first version of SMPng on 7 September 2000, less than 3 months later. And by then I had understood Mike Karels' objections to the code merge: even merging the relatively small number of differences between BSD/OS and FreeBSD was like pulling teeth.

In October 2000, we had the first BSDCon (and not FreeBSDCon) in Monterey, California. This was also the first time ever that all core team members got together. I also discovered that Justin Gibbs had formed a FreeBSD Foundation. I was puzzled by the duplication, and it wasn't really clear to me where the boundaries lie, nor why there should be two bodies. A quarter of a century later I still don't understand.

2001

In 2001 the members of the Core Team learnt what the New Order was like: not specifically technical, but administrative. We were no longer a small group of hobbyists, but a bureaucracy. The core team no longer set technical direction; instead, we were mainly dispensers of commit bits, and our main involvement was with squabbles between committers.

In early 2001, Linuxcare began to disintegrate. Our "OzLabs" department, with such Linux luminaries as Andrew Tridgell and Rusty Russell, was disbanded. Paul Mackerras had good contacts with IBM, and he managed to get most of the group working for IBM very quickly. That would have to be pure Linux, of course, and I spent some time looking for alternatives, but didn't come up with anything good, and ultimately joined IBM in May 2001.

That required learning Linux more intimately. People decided that I would be the person to create a clone of IBM's JFS version 1 to run on Linux. Why a clone? It was a different part of the company, and we didn't have access to anything more than the header files. It was quite an experience, but that's another story. From my point of view, I got to see Linux from the inside, and it didn't inspire me. I also ran Linux on one of my normal machines, and from time to time, when some FreeBSD misfeature irritates me, I experiment with converting to Linux. None of these experiments were successful: as of 2023, I still run only FreeBSD in my house network.

During this time, I did a number of presentations and tutorials about FreeBSD, including the last of my "installing FreeBSD" tutorials and papers at the USENIX ATC in Boston and the AUUG conference in Sydney. Vinum and SMPng had required lots of debugging, of course, and I built a tutorial around that, which I presented multiple times, first at the EuroBSDCon in Brighton in November 2001.

To our horror, we had discovered that Microsoft had out-performed FreeBSD on web server applications.

The debugging tutorial was interesting. Part of it involved a demonstration of the techniques, of course. On the first presentation in Brighton, I didn't have much of an idea of what to show, but when we came back after lunch, I opened my laptop and... it panicked! So, I had a panic to analyze without any preparation. And it was an interesting one: I had rebooted my laptop, but the system had checked some random memory location to see whether it needed to restart services. It turned out that the memory hadn't been completely drained, and much of the memory still had the old contents. The system had thus assumed that no reboot had taken place, tried to access memory elsewhere, and ran into trouble. By the evening, one of the participants had worked out a fix and committed it. A very satisfactory outcome.

On another occasion, though, things didn't work as well. At the first AsiaBSDCon in Taipei, I decided to do a real demonstration with two laptops and Warner Losh's help. In front of a packed main auditorium, we connected the machines, put one of them into debug... and the other one hung: it was trying to access kernel sources on the first laptop. Both machines needed a reboot. Egg on my face.

2002

Gradually my activity in the project slowed down: I no longer had an employer who was prepared to fund my development activity, and somehow that had become a requirement for any significant work on the project. But I continued with other aspects, like the Daemon's Advocate articles and presentations and tutorials at conferences. Now that Walnut Creek was gone, O'Reilly also agreed to publish the fourth edition of *The Complete FreeBSD*.

Life in the core team was not much fun. We had a surprising number of rogue committers, in particular a high-profile case where two committers fought each other in the open. We ultimately came to the conclusion that we would have to expel one, Matt Dillon, though it was by no means clear that he was more at fault. He went on to create the DragonFlyBSD project.

The stress showed: in relatively short succession, Jordan Hubbard and Mike Smith resigned from the core team, and also from the project. To quote Jordan,

"Finally, it also bears noting that while being part of the FreeBSD project is many things, it should always be "fun" to at least some degree for its participants or there's really not much point in being involved."

And Mike echoed his sentiment: "It's not fun anymore." Somehow that symbolized the changes that had happened in the first ten years.

2003

From my perspective, things continued to wind down. In April, the fourth edition of The Complete FreeBSD was published, and I presented my debugging tutorial a couple of times. And I continued to commit to the source and ports trees, but by this time I was more a hanger-on. My time had passed.

Twenty years on: 2023

Now, the time frame of this overview looks like the distant past to some. Many project members, even a core team member, weren't even born when the project started. How do things look now?

They've changed, of course. More than I expected. From the outside things didn't look that different, I stood for election for the core team again in 2022 and I was elected. And from the inside much has changed, mostly, but not all, for the better.

Part of the "early days" were concepts like "we're ahead of the crowd" and "eat your own dog food." But they don't seem to apply any more. I wrote *The Complete FreeBSD* in troff, using only FreeBSD tools. When asked to submit this article, I was given the choice of plain text or (Microsoft) "Word". Nobody would have thought of that in the olden days, though "Word" was around then too. And many of our communication tools are external to the project. I'm still trying to understand the Brave New World.

GREG LEHEY was born in Australia and was educated in Malaysia, England and Germany. He spent most of his professional career in Germany, returning to Australia 10 years before retiring. He has worked for numerous computer manufacturers, the German space research agency, nameless software houses, a large user, and also for himself as a consultant.

Over the course of 35 years in the industry, he has performed most jobs. About the only thing he hasn't done is write commercial applications software or understood Microsoft software.

He has been involved in the free software movement since 1989 and has been a member of the FreeBSD developer team since 1998, serving 3 terms on the FreeBSD Core Team, of which he is a current member. He has also been a NetBSD developer and the President of AUUG, the Australian Unix User Group.

He is the author of *Porting UNIX Software* (O'Reilly and Associates, 1995) and *The Complete FreeBSD* (Walnut Creek, 1996 and O'Reilly and Associates, 2003). He retired in 2007 but is still active in the FreeBSD community.

FreeBSD and the Early Unix Communities

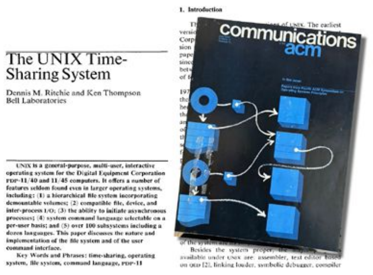
BY WARNER LOSH

FeeBSD is a collaborative and supportive community that allows users to help each other build wonderful products on a common, open platform. We can trace the community's roots and dynamics back to the vibrant mid-1970s Unix users groups. This aspect of Unix history is often glossed over; however, without these mutually beneficial early communities, we would not have a FreeBSD story or even a Unix story to tell today.

Out of these early communities (and others for similar hardware and software of the 1960s and 1970s) grew the open source community that thrives today. These early communities, such as IBM's SHARE and DEC's DECUS, formed because computers of the time were quite expensive, and users often had similar problems that might not align with the manufacturer. Community members shared not only modifications to the vendor software, but also interesting programs that solved problems the vendors had no interest in solving.

Batteries Not Included

Unix started out life as a reaction to a canceled project on cast-aside hardware as a skunk works project at Bell Labs, AT&T's research department in the early 1970s. At the 1973 ACM Symposium on Operating Systems Principles (SOSP), Ken Thompson and Dennis Ritchie presented the first public talk on Unix. The July 1974 edition of *The Communications of the ACM* published the first paper.



The UNIX Time-Sharing System

Dennis M. Ritchie and Ken Thompson
Bell Laboratories

UNIX is a general-purpose, multi-user, interactive operating system for the Digital Equipment Corporation PDP-11/40 and 11/45 computers. It offers a number of features seldom found even in larger operating systems, including: (1) a hierarchical file system incorporating demountable volumes; (2) compatible file, device, and inter-process I/O; (3) the ability to initiate asynchronous processes; (4) system command language selectable on a per-user basis; and (5) over 100 subsystems including a dozen languages. This paper discusses the nature and implementation of the file system and of the user command interface.

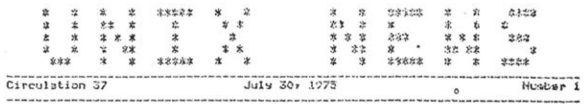
Key Words and Phrases: time-sharing, operating system, file system, command language, PDP-11

1. Introduction

For complicated legal reasons, AT&T had to give away, for the cost of duplication, any cool, non-telephone technology it created for non-commercial use. Requests for Unix started pouring in after this publication.

The first copies of Unix went out in early 1974. AT&T was also prohibited from offering support, so recipients were on their own. AT&T guaranteed the tape could be read, but little else. AT&T was also prohibited from selling any products that weren't telephone services, so these tapes were not a supported product. Technically, users had to figure out how to use and install the system on their own, but Ken Thompson was involved in many early installations. Once the system was installed, it was up to the users to figure out for themselves how to solve any problems they encountered. This stood in stark contrast to large computer makers like IBM or DEC, who provided extensive support to their users. If you wanted to run Unix, you really had to want it.

Shortly after the first tapes went out, the first user group meetings started in New York City. People would gather around and talk about Unix, how to use it, bugs they were encountering, etc. People from AT&T would attend some of these meetings in an unofficial capacity. After the first few meetings, word spread about them, attendance grew, and a newsletter, Unix News, was soon published to record the interesting bits of news and gossip for people who couldn't attend. The first issue went out to 37 people, but it listed the names and addresses of about 50 Unix users—like a directory—so users could contact each other, exchange information, and help each learn how to use this new operating system.



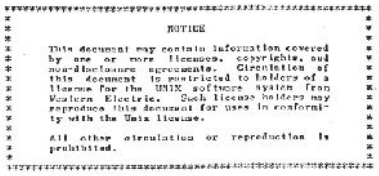
Circulation 37 July 30, 1975 Number 1

Although it's hard to get exact numbers for Unix's early popularity, we can make some educated guesses. In the first half of 1974, maybe 10 tapes went out. By the start of 1975, there were more than 50 installations. By mid-1975 there were almost a hundred. By the end of the year there were close to 200. Unix grew substantially inside of AT&T during this time as well.

These user groups weren't limited to the United States. Groups in the UK and Australia produced their own newsletters. In Canada and Europe, these groups operated as special interest groups inside of DECUS. Even the early *Unix News* address list included people from 10 different countries. Unix quickly became an international affair from a very early date. We know many of these details because sometimes the newsletters would republish pages from other groups using Unix.

First Legal Battles

AT&T's legal department was good at protecting the Unix name and rights, but it wasn't very good at community relations. First, they objected to talking about Unix without a strict legal notice. The Unix license requires that the source, methods and concepts remain confidential, so the legal department forced *Unix News* to use a legal disclaimer.



NOTICE

This document may contain information covered by one or more licenses, copyrights, and non-disclosure agreements. Circulation of this document is restricted to holders of a license for the UNIX software system from Western Electric. Such license holders may reproduce this document for uses in conformity with the Unix license.

All other circulation or reproduction is prohibited.

Next, they didn't like the name *Unix News*, since it infringed on their trademark of Unix, so they forced them to change their name to *;login: The Unix Newsletter.* It seemed like a needless fight to move the name to a subtitle.

Within a couple of years, other newsletters sprang up in the UK, Australia, and Europe. Each included this legal disclaimer. The confidentiality colored many of the early discussions about Unix, and limited publication of cool aspects of Unix in the early days, despite the fact that most of the methods and concepts were published in the CACM article. As you might imagine, this rubbed people the wrong way.

Tapes a Plenty

Unix users exchanged software from the very beginning. The preferred form of transfer was magnetic tape or removable hard disk (both of these could hold on the order of 10MB). Within a year of the CACM publication, a vibrant software swap had sprouted. The first two organized distributions were from Toronto and Harvard. Toronto's distribution contained a number of additional drivers and some bug fixes. Harvard's included optimizations to the system, improvements to the interactive responsiveness, and some security hardening needed for university computing. Both of these were discussed in the first *Unix News* in mid-1975 when there were only about 100 external Unix systems (and a few hundred users).

Also, less than a year after the CACM article, Unix had been adopted to run on a multiprocessing system (MUNIX). The MUNIX paper discussed now-familiar ways to make a kernel that depended on global variables that could have locking added to make it thread safe. This software was also available on tapes from the Naval Research Station in Monterey Bay. One processor handled all the data acquisition, while the other handled displaying the data and status to the user.

Net Unix, from the University of Illinois, connected PDP-11s to the ARPAnet, the precursor to the Internet. Net Unix, also produced within a year of the CACM article, was discussed in the first *Unix News*, and would make the PDP-11 the most popular computer on the ARPAnet by the following year. Since it was easily available in source form, many networking protocol researchers based their work on Net Unix. It would go on to influence the TCP/IP stack that BBN gave to Berkeley, which formed the basis for almost all early TCP/IP networking stacks in the 1980s and 1990s.

First Public Use

The Boston Children's Museum appears to be the first museum to have used Unix in its exhibits. Mere months after the CACM article, the museum had already childproofed this complicated system so that even children could interact with it. The museum exhibit allowed children to explore programming through a fun graphical game.

How on earth could this have happened? The museum, which had close ties to Harvard, was running the Harvard modifications of Unix. The museum would share several more versions of its Unix innovations over the years.

Porting Unix

Three groups completed ports of Unix to completely different machines at about the same time. Richard Miller and Ross Nealson at the Wollongong University ported Unix to the Intersil 7/32 (a 32-bit machine). Tom Lyon and Eric Schmidt, students at Princeton, ported Unix to IBM's VM/360. And AT&T did its own port of Unix to the Intersil 8/32.

The Wollongong port was by far the most interesting. Professor Miller was hired on to the University with the promise that a PDP-11 would be purchased to allow him to run Unix. Instead, when he arrived, he discovered that the school had purchased an Intersil 7/32 instead, and there was no money for the PDP-11. So, he carefully modified the C compiler to produce Intersil code. But he had no way to run this C compiler, since it required Unix. So, he drove the 90 minutes to Sydney to use a PDP-11 there, running Unix a couple of times to bootstrap the compiler. He then ported the rest of Unix to the Intersil 7/32, first as a user library, then with the Unix kernel as a user process and then a native port—all of this in under six months, with the system being robust enough to go live after nine months. It was an incredibly inspiring effort from someone who had never used Unix before, getting the tape and beginning this undertaking.

Community Commentary

A COMMENTARY ON THE UNIX OPERATING SYSTEM

This booklet has been produced for students at the University of New South Wales taking courses 6.602B and 6.657G.

It is intended as a companion to, and commentary on, the booklet *UNIX Operating System Source Code, Level Six.*

The UNIX Software System was written by K. Thompson and D. Ritchie of Bell Telephone Laboratories, Murray Hill, NJ. It has been made available to the University of New South Wales under a licence from the Western Electric Company.

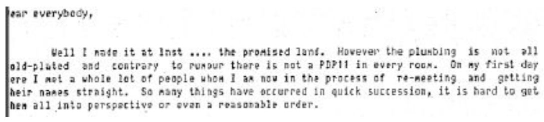
This document may contain information covered by one or more licences, copyrights and non-disclosure agreements. Circulation of this document is restricted to holders of a licence for the UNIX Software System from Western Electric. All other circulation or reproduction is prohibited.

J. Lions
Department of Computer Science
The University of New South Wales
© Copyright 1977 J. Lions

with complements
John Lions

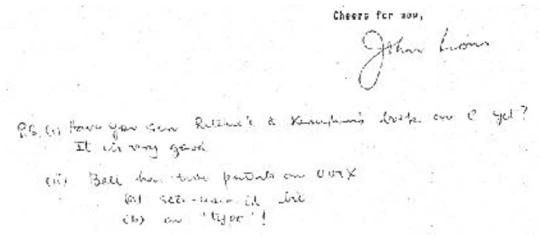
John Lions, also at Wollongong, wrote a book about the Unix kernel to use while teaching his Operating Systems class. Initially, AT&T gave him permission to distribute this book, but later withdrew the permission because the book was too good. AT&T's legal department yet again created bad PR for AT&T in the Unix community.

Things turned out well for Professor Lions, however. His fame from the book helped him to secure a sabbatical at Bell Labs. Of course, John wrote back to the Australian Unix User Group with the story, who published it in their newsletter. He wrote with news of his travels to this group, much as one would write an old friend:


ear everybody,

Well I made it at last .... the promised land. However the plumbing is not all old-plated and contrary to rumour there is not a PDP11 in every room. On my first day ere I met a whole lot of people whom I am now in the process of re-meeting and getting heir names straight. So many things have occurred in quick succession, it is hard to get hem all into perspective or even a reasonable order.

(lots of information about the forthcoming 7th Edition, and other info about AT&T omitted)

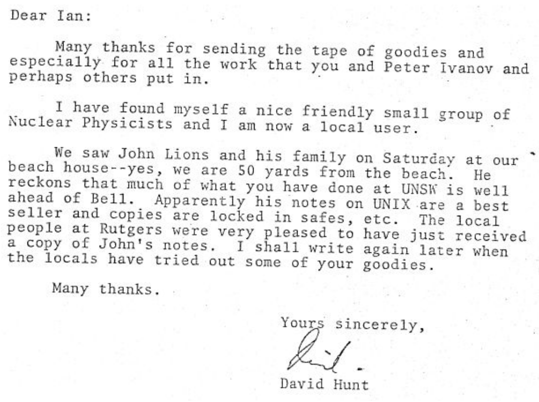

Cheers for now,

John Lions

P.S. (i) Have you seen Ritchie's & Kernighan's book on C yet?
It is very good.

(ii) Bell have two patents on UNIX
(a) set-user-id bit
(b) on 'typo'!

One of Professor Lions' graduate students, David Hunt, went on sabbatical from Rutgers University. Ian Johnstone, the editor of the AUUGN, published Hunt's report about safeguarding copies of the Lions commentary.


Dear Ian:

Many thanks for sending the tape of goodies and especially for all the work that you and Peter Ivanov and perhaps others put in.

I have found myself a nice friendly small group of Nuclear Physicists and I am now a local user.

We saw John Lions and his family on Saturday at our beach house--yes, we are 50 yards from the beach. He reckons that much of what you have done at UNSW is well ahead of Bell. Apparently his notes on UNIX are a best seller and copies are locked in safes, etc. The local people at Rutgers were very pleased to have just received a copy of John's notes. I shall write again later when the locals have tried out some of your goodies.

Many thanks.

Yours sincerely,

[illegible]

David Hunt

Conferences and More Tapes

The informal USENIX meetings grew in size and complexity. By 1977, gatherings featuring formal presentations were happening on both coasts in the US. In September 1977, over 100 people attended the gathering in Menlo Park. Usenix collected innovative software from about 30 different early Unix users for the oldest surviving tape, known as "The Third Software Distribution." This innovative tape included original software to make it easier to produce future tapes.

The content the user communities created for Unix helped Unix to grow. The user group newsletters spread the word and created connections between people. In fact, user groups published articles from each other's newsletters frequently, increasing and strengthening the connections between the user group communities. The generous sharing of code and welcoming atmosphere would serve as the basis for today's thriving open source communities, like FreeBSD.

Enabling Unix

Unix wasn't the only operating system for the PDP-11. DEC produced several other operating systems, aimed at different niche industries. For real-time control, they offered RT-11. For business and numerical applications, they offered RSX/11M. DEC also had the best FORTRAN compiler at the time (much better than the one in Unix). Many universities had data collection PDP-11s running RT-11, with other PDP-11s running RSX/11M to do the data analysis. So, there was resistance to running Unix, initially, because it couldn't run programs written for these environments. Many resorted to running the DEC OS during the day, when professors were typically around, and Unix at night, when the weirdos that hung out and used Unix were around.

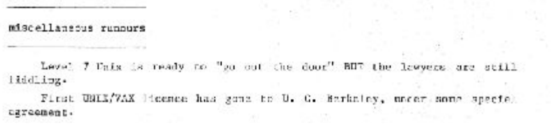
At the time, DEC distributed many of its programs to universities in source form. So, within a couple of years of the CACM article, they had ported FORTRAN and other programs to Unix using emulation libraries that translated what would be RT-11 or RSX/11M system calls into Unix system calls. If you had all the right licenses, and could provide the right paperwork, you could get copies of these programs from the university that did the port.

However, that was too cumbersome, so some clever people noticed that PDP-11 had many ways to make system calls. Fortunately, RT-11 and RSX/11M used one method, and Unix used a different method. These clever folks then added trap handlers in the Unix kernel, translating them into signals to the program. They used this to create emulation programs for RT-11 and RSX/11M, so users didn't need the source code to run the DEC software on Unix. In addition, university professors who had programs running on RT-11 could run them unmodified on Unix with the help of these emulation programs. Bill Webb at the University of British Columbia wrote the earliest emulators in 1976.

Suddenly, Unix could run 24 hours a day because it did everything these proprietary OSes did, paving the way for it to expand beyond computer labs to wider scientific applications. Unix users had created something useful that not only enabled the community to grow larger, but also broadened its sphere of influence. Unix could do more than these proprietary OSes, thanks to the cooperation of its users and their generously shared code.

DARPA Funding

I came across this interesting notice in one of the AUUGN issues:


miscellaneous rumours

Level 7 Unix is ready to "go out the door" BUT the lawyers are still fiddling.

First UNIX/VAX licence has gone to U. C. Berkeley, under some special agreement.

These two sentences, presented as just rumors, announce two of the biggest developments for Unix. "Level 7 Unix" announced the Seventh Edition of Unix, the last version of Unix from Bell Labs. The Seventh Edition would launch AT&T's Unix commercialization efforts and bring Unix to non-academic users.

The "special agreement" was AT&T giving Berkeley an early version of the Seventh Edition, ported to the VAX known as "32V". Berkeley would enhance 32V and turn it into the Berkeley Software Distribution (BSD). The US Government, through DARPA, funded Berkeley to add advanced features to Unix for the VAX. It funded their addition of the network stack to Unix. Although AT&T had a version of Unix for the VAX, nearly everybody ran the Berkeley version of Unix. Even Bell Labs would base its later research versions of Unix on BSD.

Coda

The highly collaborative community continued developing innovative and beneficial systems and software. Berkeley's releases brought a robust Unix to many universities and companies. It too was widely ported. The companies porting it, like Sun, would often make major contributions back to the community. Tiring of the ever-increasing legal burdens from Unix, Berkeley rewrote the few remaining parts of the Unix system they hadn't already enhanced and released an unencumbered distribution dubbed Net/2.

Net/2 was quickly ported to the IBM PC. In addition to a free version anyone could download, companies produced their own versions that also included support. BSD Unix was poised to take over the world. But in a final act of self-destruction, AT&T's legal folks sued over these releases. The resulting confusion and uncertainty led to the rise of Linux, which would put AT&T out of the Unix business several years later. This story, though, is too long to tell in more detail.

Nevertheless, the BSD community in general, and the FreeBSD community specifically, survived. A spirit of mutual competition developed between FreeBSD and Linux, leading to an active exchange of ideas and techniques. The same spirit of community and sharing has survived the last 30 years and continues to thrive today. FreeBSD's community of mutual support carries on the traditions of the early Unix user groups on a scale that's many times larger than those early efforts. FreeBSD is used on hundreds of thousands of computers worldwide.

It has attracted corporate users who give back as well to keep their storage and network appliances competitive. Isilon and NetApp use FreeBSD for their storage appliances. Netflix uses it to deliver all its videos. Juniper's routers use it to move network packets. These corporate users have also contributed back to the project, as do numerous other users, both old and new. The community remains strong and has welcomed several new generations of Unix users since the original ones who founded the *Unix News*. We all owe a debt of gratitude to the Unix pioneers whose early perseverance and innovation have blossomed into the robust and supportive open source community we enjoy today.

WARNER LOSH has been contributing to open source since before the FreeBSD project existed or the term "open source" was formally defined. He's recently been delving into the early history of Unix to discover its rich, hidden legacy. He lives in Colorado with his wife and daughter in a strawbale house heated by the sun, a small boiler, and the occasional antique computer.

Early FreeBSD Ports

BY DOUG RABSON

From its inception, FreeBSD focused on providing solid support for the i386 architecture. The PC platform was commonly available and relatively inexpensive and concentrating our existing resources on this helped to make FreeBSD on i386 stable and performant. However, after a few years, we decided to broaden our support; the first target was the DEC Alpha platform which was a good choice given its 64-bit architecture. A few years later, we added IA-64 support, then being positioned by Intel as a successor to the i386.

FreeBSD on Alpha

In early May of 1997, Jordan Hubbard asked for volunteers to work on porting FreeBSD to the DEC Alpha platform. The Alpha was a 64-bit, load/store architecture, which was quite different to i386 with more registers and a RISC instruction set. Available hardware used the familiar PCI and ISA bus interfaces.

I volunteered for the project. Clem Cole at DEC loaned us some hardware which arrived in July 1997 (apart from Peter Wemm's machine which I seem to remember got lost in customs). At this point, I think it started to sink in how much work this project was going to take, with changes needed in many different areas.

Device drivers in FreeBSD, especially for ISA, were going to need changes to support the new architecture where hardware access was quite different.

Drivers

Device drivers in FreeBSD, especially for ISA, were going to need changes to support the new architecture where hardware access was quite different. To make this work and share as much code as possible between i386 and Alpha, we needed an abstraction layer.

I had some ideas for this which eventually turned into FreeBSD's newbus framework. My plan for this was to discover devices automatically, starting from the top-level system device bus which was typically PCI and ISA for both i386 and Alpha. The kernel would dynamically build a tree of devices and then match these with available drivers. I also wanted to be able to combine this with earlier work on the Kernel Linker (KLD) to allow drivers to be added after the initial boot. I worked on a prototype for this in late 1997 but didn't get as far as supporting any 'real' hardware drivers.

File Formats

In 1997, FreeBSD was using the a.out file format which was a very simple 32-bit format. Most other Unix-like systems were using ELF format by this time which provided more flexibility and already supported 64-bit platforms including Alpha.

Moving to the new format involved changes in the build system to support the new format and also to support dynamic linking, which is implemented in userland using the Runtime Linker (RTLD). Lots of people worked on this including John Polstra, Jordan Hubbard, Peter Wemm and myself over several months in early 1998.

Booting

This was initially an area of uncertainty for the project. The Alpha 'pre-boot' environment was fragmented with Digital Unix (DUX) and VMS using the SRM Console, NT using AlphaBIOS and Linux using both of these as well as their own MILO.

Any operating system on Alpha needs something called PALcode to handle virtual page translation, caches, interrupts and transitions between user-mode and kernel-mode. There were variants of PALcode for DUX, NT and VMS. While the DUX PALcode was likely to be our best choice, it wasn't clear if we could use it without an expensive licence. As it turned out, all the Alpha hardware we ended up supporting had support for the SRM Console which came with DUX PALcode, so this ceased to be a problem.

The i386 boot code was limited in size to just 7.5k. The limit came from the UFS filesystem format which reserves an 8k area for the bootstrap, and we needed a 512 byte sector from that to allow the PC BIOS to boot. This was just enough to read the a.out format kernel file from the root filesystem into memory and start it.

Alpha would have a similar limit; the SRM Console used the first sector of the disk to identify a contiguous range of sectors to load, which left 7.5k from the UFS boot area in the same way as i386. On Alpha, the 7.5k limit probably wasn't going to be enough due to the lower code density of the RISC architecture and the extra complexity needed for the ELF format. We ended up rewriting the bootstrap so that the 7.5k boot stage loaded a larger boot program (in modern FreeBSD systems this is /boot/loader). This gave us enough flexibility to fully support booting ELF format kernels on both i386 and Alpha as well as other new features such as pre-loading kernel modules, network booting, and more. Mike Smith worked on the multi-stage boot and Peter Wemm implemented module pre-loading. Somewhere along the way, a Forth interpreter was added—I think Jordan Hubbard was responsible for that.

Userland

When we started this project, the FreeBSD source tree was not set up for easy cross-compiling. This made building user-mode utilities a little challenging. John Birrell worked on getting most of the FreeBSD source tree to build on a NetBSD host and had a system with a fairly complete FreeBSD userland running with a NetBSD kernel.

The NetBSD system call interface was a little different from FreeBSD, so John's early userland work used NetBSD's ABI. A native FreeBSD kernel would need utilities that used the FreeBSD

ABI, but this was a significant step forward. Once we had a working kernel, it was a relatively straightforward process to move from the hybrid NetBSD/FreeBSD system to a working native FreeBSD system.

Kernel

This was probably the largest part of the project and involved filling in all the machine-dependent parts of the kernel which provide low-level support for virtual memory, interrupt handling, process context switching etc.

I approached this by building an Alpha cross compiler and just attempting to build a kernel, seeing what failed to compile, then filling in the gaps, either with empty stubs or by importing code from NetBSD where it was similar enough to work for FreeBSD. This was a fairly tedious process which took several days, but eventually ended with a non-functional kernel binary.

The next, longer, part of the port was to attempt to run this kernel, seeing how far it got until something broke and then fix that problem before trying again. To run each test, I used a tool called SimOS. This simulated an Alpha-based computer complete with simulated hardware such as disks and serial ports. SimOS supported debugging the simulated kernel with gdb; this was extremely helpful since I was able to single step through the very early kernel initialization process which sets up the kernel virtual memory, etc. before moving onto the machine-independent initialization sequence.

To shorten the porting process, I used code from NetBSD/alpha where it made sense. Unfortunately, I omitted the NetBSD copyright in a few places. This had to be fixed in public after the code was committed which was quite embarrassing. This is one of the very few times where FreeBSD's commit history was altered—we removed the revisions with incorrect copyrights.

One area where using NetBSD code wasn't going to work was in the virtual memory support where FreeBSD was quite different. The Alpha page tables were similar to i386 with a tree-based structure using three levels (where i386 used two levels at the time). I copied the i386 code and changed it to add the extra level.

Initial support for Alpha was committed in July 1998 with support for the SimOS emulator and real hardware support followed over the next few months. The release notes for FreeBSD 3.0 mention this: 'A port to the DEC Alpha architecture has entered "ALPHA" (haha) status'.

FreeBSD on IA-64

This project got started in 2000 when Paul Saab brought a set of IA-64 documentation to Usenix ATC and asked me if I would be interested in porting to this new platform. At the time, Yahoo! was a large-scale user of FreeBSD on i386 and some of their workloads were running up against limitations of the 32-bit platform.

The IA-64 architecture was interesting in several ways. The instruction encoding was in 128-bit instruction bundles, each of which contained up to three 41-bit instructions which may execute simultaneously. This allowed for a large register set with 128 general purpose registers and 128 floating point registers.

The general-purpose registers are divided into two groups--32 'static' registers and 96 'stacked' registers which the processor would allocate from a large pool of registers and automatically save and restore on function call and return. Each register is 64 bits plus one NaT ('Not a Thing') bit used for speculative execution.

Conditional execution is via 64 predicate registers which are each a single bit and hold the result of compare instructions. Each instruction can be conditionally executed based on the value of a predicate register.

Indirect branches (e.g., function pointers) are supported using 8 branch registers which can help with branch prediction.

Virtual memory management is controlled by a Virtual Hash Page Table (VHPT) which contains a subset of possible virtual to physical mappings. A software TLB miss handler is used to find translations which are not in the VHPT. The VHPT supported two formats, a 'short' format which could be used to emulate traditional tree-based page tables or a 'long' format which was a simple hash table with collision chains.

Booting

The IA-64 hardware used the EFI pre-boot environment. I added very basic support for EFI to the multi-stage boot loader. In the IA-64 EFI environment, programs were relocatable; this needed to be done in the EFI program itself which was difficult to debug.

Initial support for Alpha was committed in July 1998 with support for the SimOS emulator and real hardware support followed over the next few months.

Userland

Porting the FreeBSD user-space tools and utilities was fairly straightforward—the FreeBSD build supported cross-compiling by this time and only needed the addition of IA-64 versions of low-level library code for things like string comparison, memory copy, and system calls.

Kernel

We were lucky enough to have access to an HP IA-64 emulator (SKI) from which Marcel Moolenaar made a FreeBSD port. This included an instruction-level debugger which was very helpful in debugging early kernel initialisation and trap handling.

The kernel port was a little more difficult than Alpha. This time, there wasn't another BSD port which could be used for reference, so all the low-level support was new code. The IA-64 architecture required two stacks, one for registers and one for regular data. Trap handling was significantly more difficult than most other architectures due to the extra register state and the complexity of speculative execution and the stacked registers.

The long-form VHPT format ended up being a reasonable fit for FreeBSD's virtual memory system. The machine-independent VM system makes requests to the platform's pmap system to make virtual to physical mappings. These were just added to the VHPT.

32 Bit Compatibility

During development of the port, we used Perforce for source code control and there was only an i386 binary available at the time. I wanted to be able to use this on the target platform during the port, so I ended up implementing i386 compatibility which used the built-in i386 support in the IA-64 processor. This built on earlier work on Linux and SVR4 emulation which had made a clear separation between the syscall ABI and implementation.

Legacy

The Alpha port prompted a great deal of necessary supporting development which has helped to shape the modern FreeBSD kernel. The transition from a.out to ELF format was a necessary step for the Alpha port, but since ELF rapidly became the de-facto standard, moving away from a.out on all platforms saved us from having to spend large amounts of effort supporting and extending an obsolete format. The multi-stage boot loader has proven to be a flexible platform, making new architecture ports easier and supporting booting from modern file systems such as OpenZFS. The newbus device framework facilitates driver compatibility across architectures and supports dynamic device discovery which is required in modern systems where devices can be added or removed at any time.

Adding support for Alpha forced us to tackle 64-bit compatibility problems across both kernel and user. The load/store architecture uncovered other problems such as the assumption that read-modify-write operations on memory to set flags or increment counters could not be affected by hardware interrupts. This was solved by adding a set of 'atomic' operations to the kernel. The atomics framework was extended by John Baldwin to support IA-64's acquire/release semantics and is used extensively to support multi-cpu platforms.

The IA-64 port was inspired by the need to get past the limitations of the 32-bit i386 platform while retaining compatibility with legacy software. While these goals were achieved, the platform itself did not reach the price/performance of the simpler i386 architecture. IA-64 eventually found its niche in large-scale Supercomputing, but it was not a good fit for most FreeBSD workloads and was superseded by AMD's x86-64 extension to the i386 architecture, which is pervasive in modern compute environments.

Support for both platforms has since been removed from FreeBSD. The Alpha architecture was a casualty of the Digital/Compaq merger, although it continued to be available as a product until 2007. FreeBSD support was removed in 2006. Support for IA-64 survived a little longer; Marcel Moolenaar made many improvements over the years to support multi-processor and NUMA variants of the platform. Support was removed from FreeBSD in 2014 and the platform was discontinued in 2021.

DOUG RABSON is a Software Engineer with more than thirty years of experience ranging from 8-bit text adventure games back in the 1980s to terabyte-per-second distributed log aggregation systems in the 2020s. He has been a FreeBSD project member and committer since 1994 and is currently working on improving FreeBSD support for modern container orchestration systems such as podman and kubernetes.

Recollections: An Interview with David Greenman Lawrence (dg@)

BY TOM JONES

The FreeBSD project started out with contributions from many hands, but the early days of the project and the people behind our favourite Operating System haven't been covered in much detail. As part of the 30th Anniversary Issue of the *FreeBSD Journal* I set out to speak to those involved at the start of development and learn how they became involved.

This first installment is with David Greenman Lawrence, an early contributor who helped give FreeBSD its "high performance server" reputation. Further installments will follow in subsequent issues.

TJ: Can you explain generally what you were up to in the late 80s/early 90s in the period before the start of the FreeBSD project?

DGL: This was the period of my early twenties, and I was involved in a lot of different things simultaneously and going in a lot of seemingly unrelated directions. For example, I was the Technical Director for a video production company that produced arts programs for Cable Access. I co-founded a company that was involved in establishing US trade in Portland's "Sister City" in the Russian far-east (this was in the early days of the new Yeltsin democracy in Russia). I founded a company that installed TVRO home satellite systems (the 12-foot dishes for C-band satellite, not the tiny Ku-band dishes of today), while also working as an independent contractor providing satellite uplink engineering services for a TV broadcasting company. I was also an independent contractor with expertise in DEC PDP-11 and VAX systems (mostly repairing customer hardware). And finally, I was a computer hobbyist with an interest in Operating Systems development. I hacked RSTS/E and VAX/VMS for fun. This was rather difficult, however, as DEC only provided limited source code for RSTS/E and nothing more than assembler listings for old VAX/VMS utilities. I learned to be a pretty good DEC machine code hacker! For a short time, I was also the President of the "Portland Computer Society" (a 501(c)(3) non-profit organization of local computer hobbyists).

Fun times, but it was my hobbyist interest in Operating Systems development that ultimately led to playing with the source code for 2.9 BSD, which I ran on one of my PDP-11 systems. This led me to Bill Jolitz's release of 386BSD 0.0 in 1992 and getting involved with the development of the "386BSD Patchkit" (a project led by Rod Grimes, a fellow Portlander). I built my first Intel PC - a 386SX with 4MB of RAM just for this purpose. However, Bill Jolitz wasn't really much of a "team player."He generally rejected and even publicly ridiculed patches and improvements that people offered and even eventually objected to the use of his "386BSD" trademark in connection with the Patchkit and an "interim" release of the patched system (which we were calling "386BSD Interim-0.1.5" at the time). He insisted that we stop calling it that, and, ultimately, this forced our group of hobbyists to organize our own development project, completely independent of Bill Jolitz.

TJ: How did you come across FreeBSD/the efforts that led to FreeBSD? There is a lack of accounts where people discuss how they found information, was it USENET? (if so, how did you connect?). Were any of the PC BSD efforts covered in more conventional media (magazines, etc.)?

DGL: Julian Elischer posted a message on comp.unix.bsd (dated November 17, 1992) about a series of mailing lists that he had set up for discussing 386BSD (Bill Jolitz's baby). I read the posting and subscribed to some of them—probably on the same day. Julian also set up some logins on "ref" (which was running 386BSD) for people to hopefully use constructively. I may have already had an account on that machine, however, prior to the creation of the mailing lists. I'm not sure because my real time access to the Internet was very spotty and limited at the time. Netnews and email were being delivered by UUCP, for example. I think I had to dial up another system using a modem for telnet access. Anyway, Julian's 'ref' resources wasn't the first contact with other 386BSD'ers. I know I had contact directly with Bill Jolitz and some other 386BSD enthusiasts. I knew Rod Grimes because of another Portland area project called "RAINet" (the "Research And Information Network"), which was an early-days attempt to get local computer hobbyists (Inter)connected to the newfangled "Global Internet," mostly using SLIP over dialup modems. Many of the local computer hobbyists at the time—some of whom worked at Intel or Tektronix, or attended or taught at Portland State University, knew each other through various events and social circles (including the Portland Computer Society that I mentioned earlier).

It was my hobbyist interest in Operating Systems development that ultimately led to playing with the source code for 2.9 BSD.

TJ: Julian's machine isn't something I have heard of before. I'll certainly dig more in that direction in future projects.

DGL: Also, Nate Williams had set up a mailing list for the major players in 386BSD called interim@bsd.coe.montana.edu. Most of the discussion occurred about 386BSD, after 0.1, while we were waiting for Bill Jolitz to release 0.2—which seemed to be delayed indefinitely.

TJ: Could you tell me a little about how you came to write sendfile(2) and other optimizations that allowed ftp.cdrom.com to serve so much traffic?

DGL: Wcarchive (AKA ftp.cdrom.com) became my obsession and it was the driving factor behind much of my development with FreeBSD. Rod Grimes built the first PC-based version of it in 1993, but I soon became involved in dealing with the machine's daily reliability problems. I took over the management of both the hardware and software shortly after that. One of the first challenges was the very limited bandwidth available on Walnut Creek CDROM's 1.5Mbps T1 connection to BARRnet. Even in those very early days, wcarchive was a very popular FTP distribution archive for the most popular Shareware of the day. The T1 was maxed-out 24x7 with packet loss that exceeded 50% most of the time. In fact, the T1 was so overwhelmed, that the congestion caused major issues at BARRnet, adversely affecting other customers. Bob Bruce (cdrom.com owner) proposed upgrading the T1 to a (45Mbps) T3, but that idea turned out to not be practical (or perhaps even possible) at the time.

With extensive testing and kernel profiling, it was obvious that there were significant performance and scalability issues in both the TCP/IP and the socket layer code.

I think it was someone at BARRnet/BBN who suggested the server could be moved to the Stanford University data center in San Jose (the main BARRnet/BBN POP in the region), which would mitigate the congestion issues with the T1. So, in September 1994, I built a new machine and deployed it at Stanford. Of course, no one really knew just how oversubscribed the T1 had been. BARRnet/BBN assumed the load would be just 1.5Mbps. Much to their dismay, traffic immediately jumped to over 30Mbps as soon as we brought the server online at the data center which caused a bit of a panic with the BARRnet/BBN network engineers. We continued to operate the server, but it was clear that BBN really didn't see the financial case for hosting such a bandwidth-expensive resource, and I think they had to put on some bandwidth caps to keep the cost under control. Ultimately, within a year or so, I had to find a new home for the machine—where there were no limits. FreeBSD's reputation was at stake here after all. An average user downloading from wcarchive couldn't tell the difference between the server just being overwhelmed and the network being congested, and I was also keen to set new records and push the boundaries of what a single server running FreeBSD was capable of doing. So, I found a new home at San Francisco based ISP "CRL Network Services". In February 1996, I moved the machine to CRL where we were given 100Mbps of bandwidth to start.

In the years that followed, it was a constant struggle to keep up with the demand. There were many hardware and network upgrades, but it was really the performance of FreeBSD where I focused. With extensive testing and kernel profiling, it was obvious that there were significant performance and scalability issues in both the TCP/IP and the socket layer code. Some of these issues could be optimized (and I did), but the Holy Grail really was much bigger than that—I needed to get rid of all copying of the file and network data as well. While there were some "zero-copy socket" tricks that had been implemented in other operating systems, these were kind of a mess architecturally, were difficult to use in the application, and still left performance overhead related to user-kernel context switching and many other issues. What was needed, I thought, was a magic system call that did pretty much everything—you just give it a file descriptor and a socket descriptor and it would send the contents of the file out to the network.

While attending a technical conference in 1998 (probably Usenix ATC), I told several colleagues about my sendfile() idea. Someone suggested that Sun Microsystems may have implemented something like the sendfile() I was proposing. I was very curious about the API (for compatibility), but when I reached out to Sun to find out about their API, I quickly found that the rumor wasn't true. I think someone at Sun suggested that something like sendfile() might have been implemented in HP/UX. Anyway, I didn't have any contacts at HP (or know if that rumor might also be false), so decided to move forward with my own API. What I came up with had more arguments being passed on the stack than any other syscall in FreeBSD, which concerned me a bit, but it was nonetheless the most efficient way to do it. sendfile() was a difficult syscall to write. It seemed to touch just about every subsystem in the kernel—file I/O, VM system, network buffers, etc. Anyway, the initial version of sendfile() reduced total CPU time on the server by about 75%, or in other words, made wcarchive about 4 times faster. It was a huge win.

TJ: Can you tell me what drove you to get more involved with the organization and management of the project?

DGL: I've been thinking about this question, and I finally figured out why I was having trouble answering it: The "What drove you" phrase makes an incorrect assumption--it implies that there were some forces that pushed me toward an organizational and management role, but that really wasn't the case. While some managers are pushed into their positions, I would call those unfortunate people "reluctant managers," and more often than not, they usually are not well suited for their job. For me, it was more of an attraction to a needed organizational role, and it was in my nature to fulfill that.

With that said, managing and organizing a freeware volunteer project is very different from managing employees in a for-profit, private enterprise. In an all-volunteer project with no budget and limited donated resources, a manager has almost no authority to command people to do things. If you try to command a volunteer to do a thing, they're likely to just get angry with you, ignore you, or go away. Instead, you have to be much more subtle by guiding people in a direction--through discussion, consensus, collaboration, and be willing to do (at least) the proof-of-concept first yourself. With a proper display of ambition and direction (or sense of direction), others will follow.

Oh, and "herding cats" also comes to mind. ;-)

TJ: Were there specific problems you wanted to address as a member of the core team?

DGL: Well, of course, my main focus in the project was the development of the software (and the kernel in particular). I was focused on reliability and performance, but also on the architectural direction of FreeBSD. I felt that having a Core Team was pretty important to managing the legendary quality of the BSD codebase. There was definitely a contingent of anarchists in the group, however, that objected to any sort of formal order and preferred FreeBSD to be just a cabal of unorganized developers, but that isn't what I wanted.

TJ: How did the project change during your time on core?

DGL: I was a Core member for a total of 10 years. The first 8 years as a founding member of the unelected/self-appointed Core Team, and then 2 years on the first elected Core Team.

The first Core Team started out as a collection of very talented software developers that had a common admiration for BSD Unix and a keen interest in continuing the legacy as an open-source project. In the early days, it was mostly about just trying to make FreeBSD work reliably enough to be useful. As FreeBSD became a thing in its own right, the size of the project grew with over 300 developers who contributed all sorts of things that I never would have imagined in the beginning. Jordan's wonderful idea of the FreeBSD ports tree, for example, was huge in furthering the adoption and ease of use of the system and it attracted a large number of additional (ports tree) contributors as well. This is all good, except that as the development team grew, so did the diversity of opinions, frequency of disagreements, and challenges to the project's leadership. With the Core Team being self-appointed, there was an ongoing question about where the Core Team's authority comes from. This ultimately led to the Core Team reorganizing into an elected body—elected by the developers. I have to say that this didn't really change anything with respect to how things operated day-to-day within the Core Team, but it did perhaps give a little boost of legitimacy to our authority.

The project changed in many other ways as well. With FreeBSD maturing into one of the best server operating systems available, the project was increasingly able to attract donations and specific project sponsorships from various corporate users. These were a bit difficult to make and accept in some cases, because FreeBSD was, in fact, not a legal entity. Even the FreeBSD trademark had to be held by Walnut Creek CDROM, and I personally owned the freebsd.org domain name. This all changed, however, when Justin Gibbs (a Core Team alumnus) founded the FreeBSD Foundation in December 2000, as a US non-profit 501(c)(3) legal entity.

TJ: What is the lasting legacy of FreeBSD?

DGL: There are so many legacies to choose from. The fact is the average human on planet Earth is using software developed in part by the FreeBSD Project every day--from shortly after they wake up in the morning until they go to bed at night. If you use a mobile phone based on Android or iOS, then these platforms borrowed significantly from FreeBSD for their libraries and user applications. If you use Microsoft Windows, Apple MacOS, or iOS, then you're using a kernel networking stack that mostly came from FreeBSD. If you sit down at night to watch some streaming movies on Netflix, then that content is being served to you by servers running FreeBSD. If you're a gamer with a Nintendo Switch console, then you're using a platform powered by FreeBSD. If you do online banking, or trade stocks on the stock market, or ship a package around the world, then you're probably doing it with servers that are running FreeBSD (although I could never get them to admit this publicly!).

In all the cases of borrowing code or completely basing platforms on FreeBSD, there is a reason why companies have chosen to use FreeBSD instead of Linux. It's the most important legacy of all and something we inherited from BSD before us. If you ask a

FreeBSD's lasting legacy is the very concept of truly FREE software.

software professional what the de facto standard most permissive software license is, they'll tell you "the (N-clause) BSD license." From BSD, it was the 4-clause license, but FreeBSD took that a step further and cut that down to just a simple 2-clause license. Pretty much what it says is that as long as you don't claim to have written the code yourself, you can use the software in whatever way you want. You can change it to suit your needs and keep your changes proprietary. You can make a ton money from it and not give a penny back to the original authors. It means that it is truly FREE software in every sense of the word. I can tell you that this was forefront in my mind (we ALL had this in mind) when the project adopted the "FreeBSD" name on June 19, 1993. Although it wasn't planned or even thought of at the time, it was a happy coincidence that this also occurred on the anniversary of "Juneteenth"—a day celebrated as the day that slavery formally ended in the United States—June 19th, 1865.

FreeBSD's lasting legacy is the very concept of truly FREE software.

TOM JONES is a FreeBSD committer interested in keeping the network stack fast, during the day he manage FreeBSD development at Klara Inc.

FreeBSD at 30 Years: Its Secrets to Success

BY MARSHALL KIRK MCKUSICK

This year the FreeBSD Project is celebrating its thirtieth year of providing a complete system distribution. The goal of this article is to understand what it is that has made FreeBSD one of the few long-term, viable, open-source projects. Most projects with long-term successes are sponsored by companies that base their products around the open-source software that they actively nurture. While FreeBSD has companies actively using and supporting it, they have come and gone over the years and no single company has been the primary long-term proponent.

Origin

Many open-source projects start with code written by one person and begin building from there. FreeBSD started from a solid code base, the 4.4BSD-Lite open-source distribution from the University of California at Berkeley. The Berkeley Software Distribution (BSD) had been in development and distribution for over a decade and the BSD distribution started from the Unix distribution from Bell Laboratories that had been in development for a decade before BSD. Though BSD was not open source, its code was widely licensed and had many contributors from both academia and industry. Nearly all of BSD was ultimately released as open source in the 4.4BSD-Lite distribution.

When the FreeBSD organization was set up, the organizers decided to establish a group of seven people called the Core group that oversaw the project.

The BSD kernel introduced important operating-system interfaces still used today:

- the socket networking interface and the original and widely used implementation of TCP/IP,
- the set of system calls used to operate on filesystems, the virtual filesystem (VFS) interface to support multiple filesystem implementations, and the fast file system and network filesystem (NFS) implementations,
- the mmap memory model, and
- the interface to manage processes (signals, process groups, job control, etc.)

The BSD distributions also established the model of complete system distributions that included the operating system, a core set of libraries and utilities, contributed software (that would eventually become FreeBSD's ports), and complete manual pages and system documentation.

Leadership

Most open-source projects are started by a single person who then becomes the czar-for-life leader of the project. A well-known example is Linus Torvalds who created and still leads the Linux project. Projects usually go dark when the leader loses interest and stops working on it. Contributors often get frustrated if the leader is not good at reviewing and critiquing or accepting input from others.

When the FreeBSD organization was set up, the organizers decided to establish a group of seven people called the Core group that oversaw the project. The original Core group was self-selected. The people who set up the project deputized themselves onto the Core team. They were "Czars for life." The Core team decides project direction and awards and removes the privilege of being a committer; committers are the people who are allowed to make changes to the project repository.

While this approach was better than having a single leader, it still had the problem that committers could only rise to a middle level in the project, thus leading to frustration and abandonment if their ideas were not accepted. To remedy this, the FreeBSD project decided to make Core an elected position. Core was also expanded to nine people. The entire Core is elected every two years. Core members are nominated from and elected by the committers. Any active committer can run for Core. Candidates are self-selecting and no nomination is required. The effect of this change is that newcomers can rise to leadership roles. As a result, the project leadership evolves over time, and the project is much less susceptible to collapse if its leader departs.

Development

From its inception, the FreeBSD project used centrally located tools (source-code control and bug reporting). This tooling enabled remote development from the start. Though common today, at the time FreeBSD was started, the usual approach was to have a single person who maintained the distribution, and changes by others had to be sent to them for inclusion. As the project grew, the person maintaining the master copy of the source would get overloaded and limit the speed with which the project could move forward. It also made it difficult to keep track of who was working on what when bugs would arise and needed to be assigned. Happily, the modern tool sets available today like gitlab and github mitigate these issues.

The FreeBSD project has also benefited greatly from adopting ideas and code from the NetBSD and OpenBSD projects. NetBSD has lead the way in efficiently supporting multiple architectures which was very helpful as FreeBSD began expanding from its initial focus on the Intel architecture to support additional architectures. NetBSD also has provided many tests that have been incorporated into the FreeBSD continuous-integration testing. OpenBSD has focused on system security and FreeBSD has incorporated many of their security improvements. OpenBSD has also provided several of the key security components used in FreeBSD such as the ssh remote access and login program and the software components that support https encryption.

Distributions

Many open-source projects are simply a collection of code that must be downloaded, compiled, and installed to be used. They often depend on other libraries and infrastructure which must also be found, built, and installed. In recent years, projects are beginning to provide containers that can be spun up, though they are an inefficient use of resources since they include the entire software stack all the way down to and often including the operating system, thus duplicating vast amounts of software already on the machine.

Early in the FreeBSD project history it began distributing CD-ROMs with the complete system on them that could be booted on PC computers. Users could boot up the system from the CD-ROM to try it out and then install it on their hard disk if they wished to do so. And--since it was derived from the BSD system from which it started--all the commands and libraries that they needed were already there. Prolific documentation was provided, making installation easy even for non-experts.

Hardware Support

Most open-source projects try to support everything, which usually means much hardware performs poorly and often fails under load. From the start of the FreeBSD project, the decision was made to curate hardware and decide what worked well with FreeBSD. Once the hardware was selected, significant effort was made to write robust and complete device drivers to run it. FreeBSD published a list of hardware that they recommended and supported that hardware by fixing reported problems and updating drivers as newer versions of the hardware were released. This curated list made it easy to put together server machines that ran well under load. FreeBSD became the system of choice for companies running dial-up servers and later Internet and web server providers because they had great performance and ran reliably.

Communication

Since nearly all the FreeBSD developers were working remotely, it was important to set up mailing lists to discuss core design decisions. Topic areas included networking, filesystems, core architecture, etc. A frequent issue with mailing lists, especially when most folks on them have never met, is that discussion can get off-track and distinctly nasty. Flamewars were not uncommon in the first few years of the project, so the mailing lists began to be actively monitored to tamp down bad behavior and ensure civil discussion. Sadly, many projects even today have toxic mailing lists. Once a project gets a reputation for bad behavior, it often results in it entering a death spiral. Alternatively, it is possible to go to the opposite extreme and become so controlling that folks abandon the project as they feel overly constrained. And for projects like FreeBSD that have developers worldwide, it can be difficult to find rules that work in the large diversity of cultures of its developers. The problem is never solved; ultimately there needs to be an ever-evolving methodology on how to keep the project moving forward on an even keel.

Documentation

The FreeBSD project started off with a solid base of documentation based on the documentation in the 4.4BSD-Lite distribution which was in turn derived from documentation in the UNIX system from which BSD evolved. Early in its evolution, FreeBSD embraced contributors that focused on system documentation. Folks writing code were encouraged to work with those writing the documentation to ensure that the documentation was complete and correct.

The project set up a documentation committer group for the folks doing the documentation. This group was given all the rights and privileges of code committers. They could run for Core, had equal voting rights, and their own group leaders that handled adding and removing documentation committers, setting up the documentation structure and tools, and overseeing the document repository. Under their direction the documentation was structured with a framework that allowed it to easily support multiple languages. Many of the documentation committers started out by doing translations of documents into their native language. This translation task often helped them get up to speed both on how the documentation tools worked and how FreeBSD itself worked.

Since nearly all the FreeBSD developers were working remotely, it was important to set up mailing lists to discuss core design decisions.

The Ports Collection

The 4.4BSD-Lite distribution had a collection of contributed software that consisted of about fifty utilities and libraries that had been developed outside Berkeley but were included in the BSD distributions. These included things like the X window system, the gated routing daemon, the emacs editor, etc. FreeBSD started with this set of core contributed programs and greatly expanded on it with what became the ports collection. Unlike the BSD distribution which installed all the contributed programs, FreeBSD ports provided them separately so that individual sites could install only those that they needed. The ports collection ensured that the program would compile and run on FreeBSD with reasonable defaults. It also ensured that fixes for bugs found in the BSD environment were up streamed to the maintainer of the software and that changes made up stream were brought down to the FreeBSD port. Most users could just use the compiled version of the port though those needing site-specific changes could make them and then build their own binaries. The port collection made it easy to use other open-source software on FreeBSD. Having a ports equivalent is done by most open-source distributors today but was new at the time.

The ports collection has continued to evolve over the years. Recent innovations are the addition of pkg system to manipulate ports. The pkg system handles registering, adding, removing, and upgrading packages. The other key component is Poudriere that is a utility for creating and testing FreeBSD packages. It uses FreeBSD jails to set up isolated compilation environments. These jails can be used to build packages for versions of FreeBSD that are different from the system on which it is installed and to build packages for a different architecture than the host system. Once the packages are built, they are in a layout identical to the official mirrors. These packages are usable by the pkg system and other package management tools.

Port, documentation, and development committers are all given equal say in how the project is run.

FreeBSD provides a base platform that can be modified to build a customized OS along with all the infrastructure needed to build a full OS distribution including not just the base system but also a collection of the ports. The OS can be customized to support an appliance as all the bits for how to build the release image for the customized OS along with automated building of packages via Poudriere for the customized OS are public and well-documented. None of the Linux distributions are as turnkey as FreeBSD in this regard. For example, it would be much more difficult to build your own Debian-fork on top of a modified kernel and system libraries, etc.

Project Culture

Port, documentation, and development committers are all given equal say in how the project is run. Notably, they all can run for Core and get the same voting rights. In most projects, the developers have more say and others are treated as inferior. The FreeBSD project has worked on building a culture of inclusion from its start. The culture values "plays well with others" above anything else. It does not tolerate a diva just to get their documentation, port, or code (though sometimes it can take a while to get to the point of a diva leaving or getting kicked out). The FreeBSD project is not set up to train people how to write or program. Folks joining the FreeBSD project are expected to know their trade. Documentation writers are expected to know how to write technical documents. Port and source contributors are expected to know C and any other relevant languages along with the tools used to write, build, debug, and profile them. That said, FreeBSD has been involved with mentoring students through programs such as Google's Summer of Code. Indeed, many of the students in Summer of Code have gone on to become committers on the FreeBSD Project.

The FreeBSD project is welcoming to new folks. It is not necessary to survive a gauntlet of hazing or needing to ingratiate yourself to the project leader to become a project committer. There is a well-documented process on how to become involved with the project.

Project Support

When FreeBSD started, its infrastructure was a machine in a developer's home. As it grew, its infrastructure was supported first by Walnut Creek CD-ROM and later by Yahoo. Being dependent on a company's goodwill was a recipe for disaster, so the FreeBSD Foundation was created to raise money whose initial use was to provide the machines and hosting for FreeBSD infrastructure. While Foundation support for projects is common today, FreeBSD was one of the first projects to set up a foundation to support the project. The Foundation was originally run by its (unpaid) board of directors. After a few years, it was able to hire its first part-time employee. Today it has nearly twenty staff and contractors supporting infrastructure, development, marketing, tooling, fund raising, and other project-related services.

Licensing

FreeBSD uses a Berkeley license which does not require companies to make their code available to others. The use of the Berkeley license has played a big role in FreeBSD's success, particularly with companies that have their proprietary code in the kernel. FreeBSD is heavily used in the appliance and embedded operating system market where companies need to put their intellectual property inside the operating system and thus cannot use Linux due to its GNU Public License (GPL) that requires source code for all changes be made available.

Conclusions

FreeBSD is still going strong. Its strength comes from having built a strong base in its code, documentation, and culture. It has managed to evolve with the times, continuing to bring in new committers, and smoothly transition through several leadership groups. It continues to fill an important area of support that is an alternative to Linux. Specifically, companies needing redundancy require more than one operating system, since any single operating system may fall victim to a failure that could take out the entire company's infrastructure. For all these reasons, FreeBSD has a bright future. In short, FreeBSD is awesome!

DR. MARSHALL KIRK MCKUSICK writes books and articles, teaches classes on UNIX- and BSD-related subjects, and provides expert-witness testimony on software patent, trade secret, and copyright issues. He has been a developer and committer to the FreeBSD Project since its founding in 1993.

WeGet letters

BY MICHAEL W LUCAS

Mr. Lucas,

Your love of FreeBSD is obvious and lifelong. This issue of the FreeBSD Journal has turned into a trip down memory lane, so the editorial board asked me to write you and ask how you got started. Why do you do keep hanging around us? Why do you write all these books?

—John Baldwin,
FreeBSD Journal Editorial Chair

Dear John,

I never expected to write a "Dear John" letter, but life is a bottomless font of disappointment. You do realize that this is the thirtieth column I have provided to your Journal, do you not? Thirty of these meticulously reasoned clear-sighted epistles over five years. People get shorter sentences for abusing kittens. Your question provides abundant evidence that you have not read a single one. Fortunately, for your tenure as Editorial Chair, your remit is filling the pages with technically accurate information and not ensuring the quality of editorial blather. It doesn't matter how erudite the Letters column is, so long as you *have* one.

Are you aware that I have previously been asked this very same question by many organizations and publications? Including your own Foundation, so that's at least one group that's not directly tied to international law enforcement? People keep asking. Presumably that's because nobody reads the answers, in each instance liberating me to provide tissue-thin lies if not outright calumny. I accepted the Lawrence Technologies interview to challenge my ability to maintain a straight face while improvising whoppers. Wonderland Press has interviewed me repeatedly, but only in the spirit of marketing. Everyone understood that honesty would interfere with selling books. The story about the monkeys? Utter fabrication. This time, however, with an entire editorial board of the greatest esteem and probity exercising their usual immaculate oversight, I feel compelled to at last reveal the truth.

Yes, the truth!

Pinky swear.

My life-long love affair with FreeBSD had to begin at birth, obviously. Fact-checkers might note that was years before Dennis Ritchie and Ken Thompson came down off the mountain to prophesy the holy word of Unix, but the potent concepts already stirring within their minds lured my vacuous, unformatted brain into their radiance. Contemplations of a simpler multitasking operating system were quickly brushed aside by life's disgusting necessities, however, and I wasn't able to turn my attention to computing until I got my grubby paws on a secondhand Sinclair ZX80. No, not a ZX81 or one of those fancy Timexes. A ZX80. That's where I learned how to program Perl. Fact-checkers will also note that Perl was not yet a language, but everything I know about algorithms I learned from the ZX80 BASIC interpreter, so I figure Sir Clive owes the world quite the apology. That's also how I acquired my knowledge of C and shell, and the ZX80's 1K of RAM provided quite the education on memory exhaustion. Adding the ZX80's esoteric peek() and poke() operators taught me all I need to know about interpersonal relations.

Meanwhile, I learned how to write. Grammar school teaches trivialities like syntax and spelling, but these have nothing to do

My life-long love affair with FreeBSD had to begin at birth, obviously.

with proper writing. My teachers pointlessly obsessed over getting me to draw my Zs and the Ss the right way around, when all I cared about was learning to express my inner self. Working on my own, I figured out the "inner self" business in the fifth grade and promptly got to work closing that mess right back up. My reputation was already soiled, however, condemning me to become either a writer or a television propagandist. Fortunately, I have a face for radio and a voice for paper, so I was spared the indignity of broadcast media.

Then it's just practice, the same as any other stupid career. A college classmate would say, "Hey, I'm throwing a backyard barbeque Friday night and inviting a bunch of folks of the gender and orientation you find attractive, and they all have poor taste and lower standards so you should show up" and I would say "thanks but no, I'm working on this piece that will get rejected by a hundred thirty-six markets before I bury it in the Box of Failure. I'll pay postage for every rejection, of course. By the way, the crate the stove came in is filling up, so I need a new Box of Failure, let me know when you buy a fridge."

Stuffing the Box of Failure to overflowing is its own reward. You don't get paid for it.

I needed a job.

In 1995, one of my "friends" was the DNS administrator for one of the brand-new Internet backbones. They needed a disposable body to answer phones, yell at the phone company, and run poorly documented commands as root. It paid terribly, but my experience consisted of running "trn" and "elm" as well as the occasional failure at FTP so that seemed fair. It was even on the night shift, which meant I didn't have to spend extra energy debugging why my peek() and poke() operations failed to provoke people correctly. People are buggy and have no interface for dispassionately accepting bug reports. I did learn to find the joy in making callers dump core, however.

Back in the exciting days of the early Internet, we had these things called "print magazines."

Nobody explained the dangers of having the root password before handing it to me—specifically that if you break it, you must fix it. In their defense, warning me would have ruined their fun. I needed to actually learn this FreeBSD thing before I yet again wiped a server and had to reinstall 2.0.5. Walnut Creek CDROM, FreeBSD's earliest commercial backer, had published Greg Lehey's *Complete FreeBSD*. I acquired one and began studying.

Scope creep is not only for projects. It is also for junior systems administrators. The "friend" who got me hired taught me how to do her job and promptly departed for an employer that still offered hope, which showcased her wisdom until she emailed and asked for help finding yet another job and I cheerily avenged myself. By then I had learned about NNTP and ldd(1) and realized that systems administration was the closest thing our society has to black magic and if only I understood library versioning I could become the modern Aleister Crowley. It's not that I wanted the endless wild parties, nor the ability to borrow vast large sums without consideration of repayment, but the thought of absorbing that much public vituperation made me believe I could make a difference in people's lives. Fortunately, time has beaten that youthful foolishness from my heart and left me my present happy wholesome self, perfectly well adapted to the carefree work of network and systems administration.

Back in the exciting days of the early Internet, we had these things called "print magazines." They were like printed-out blogs, glued together with a shiny cover. One was called "Sys Admin," demonstrating that spelling is an optional social convention. My quest to understand the pit I'd ignorantly dug myself into led me to subscribe, which was like RSS except they show up at your house every month even when you forget to check the feed. I fondly remember reading an article that contained useful information, once I deciphered the appalling writing. My gut reaction was that I could write better during a colonoscopy. I turned the page to see a *Write For Us!* box. Annoyed that I was working on my third Box of Failure while some doofus who could barely nail a verb to a noun had gotten published, I spewed something about CVS, CVSup, and building world and sent it to the editor.

Spite is its own reward, yes, but sometimes it offers special bonus rewards. They sent me a contract, a check that covered that month's mortgage payment, and a request to be permitted to send me more checks. They even printed that article in their September 1999 issue and put my name on the cover. Every few months afterwards, I would indulgently spit out a couple thousand words on some topic that annoyed me, polish it into formal magazine text, and let the editors send me money.

If your writing is less awful than other people's, strangers will appear out of nowhere and ask you to do more of it.

In the late 1990s, tech publisher O'Reilly decided to branch out into web-based publishing. They convinced one Chris Coleman to collect articles for the brand-new online *BSD DevCenter*. I'm sure it sounded simple when they proposed he take the job, but Chris quickly discovered that the world contained about two FreeBSD authors and Greg Lehey had learned better. Chris introduced himself and offered to exchange words for cash. Fortunately, Chris persuaded Dru Lavigne to join us, or the *BSD DevCenter* would been renamed *Lucas Whinges Like A Frustrated Toddler* and nobody would click on that.

The Big Scary Daemons column was basically "what program is annoying Lucas this week, and how can it be bludgeoned into submission?" Since the column was on the web, it wasn't like my articles were *real*. It freed me to write random gobbledygook, including daft things like the "sharing swap space between Linux and FreeBSD on multiboot systems" column that people still try to discuss with me even though multiboot has gone the way of the 5¼-inch floppy.

I established the O'Reilly column just in time for Sys Admin magazine to implode. Sending me those checks wrecked the publisher. Oh, well.

In early 2001, Bill Pollock asked Chris if he knew anyone interested in writing a FreeBSD book for No Starch Press. Chris threw out my name and fled before Bill could sucker him into it.

I signed the contract for *Absolute BSD* just in time for O'Reilly's BSD DevCenter to implode. I'm not saying I am frequently seen fleeing publishers going down in flames, but it's not uncommon.

Absolute BSD led to *Absolute OpenBSD*, then *Absolute FreeBSD*, *Cisco Routers for the Desperate*, *Network Flow Analysis*, and more. I had innumerable other book ideas, but my experience with PGP & GPG showed the warehoused oblivion awaiting unpopular books and the market for a book on PAM, sudo, or ed(1) was minuscule. My notes languished in my scrapbooks, surrounded by conference call doodles: obscene occult sigils, solitaire games of tic-tac-toe, pleas for euthanasia. You know, the usual. When self-publishing became cost-effective, that let me put out the less commercial books like *SSH Mastery* and *FreeBSD Mastery: Jails*. No commercial publisher will touch niche novels like *$ git commit murder* and *$ git sync murder*, but I now have the tools so nobody can stop me from trebucheting these BSD-themed works into the public eye. My fifty-second book will escape into the wild about the time this issue appears. Fortunately, that's insufficiently notable for Wikipedia. I don't care if I have an entry therein, but I would object if said entry contained even a soupçon of the precious truth.

There. The truth. You have it.

I consider my obligation to the editorial board fulfilled.

The aforementioned "truthfulness" compels me to mention, however, that I did notice the question hidden within your letter. *Why do you do keep hanging around us? Why do you write all these books?* I cannot conceive a more obvious disguise for asking *how can we make you go away?* In that regard, I must again disappoint. I am not only aware of the sunk cost fallacy, I embrace it. Besides, someone warned the Linux folks about me. I fully expect to remain here until this esteemed Journal pays off the debt of gelato it promised me in my first column. And promptly implodes.

Have a question for Michael?
Send it to letters@freebsdjournal.org

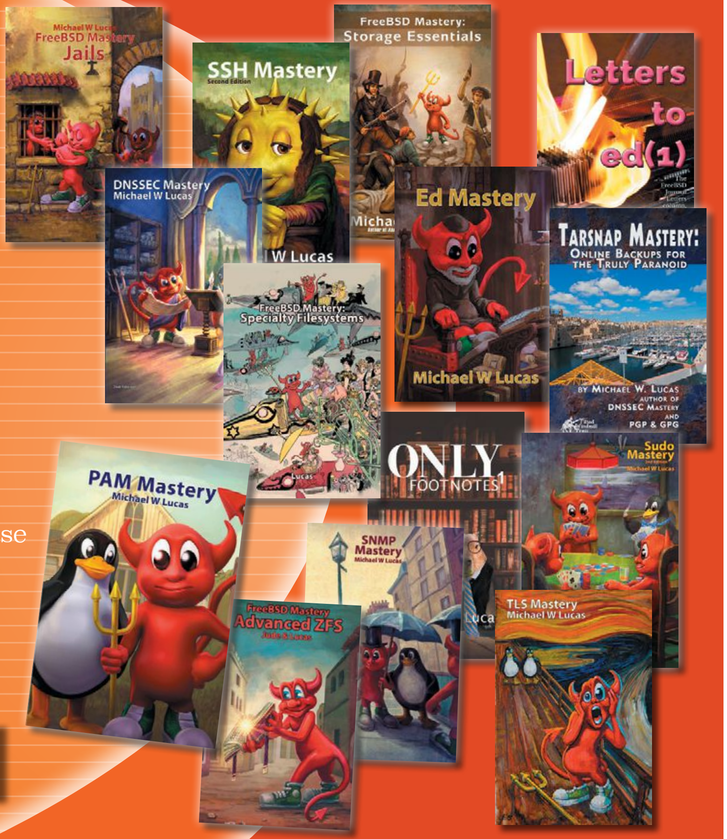
MICHAEL W LUCAS (https://mwl.io) has been scribbling this column for five years. He's currently writing a book on running your own mail server and scratching his back on the doorframe. *Letters to ed(1)* collects the first three years of this drivel.

Books that will **help you. Or not.**

"While we appreciate Mr Lucas' unique contributions to the Journal, we do feel his specific talents are not being fully utilized. Please buy his books, his hours, autographed photos, whatever, so that he is otherwise engaged."

— John Baldwin
FreeBSD Journal Editorial Board Chair

https://mwl.io


FreeBSD Mastery Jails
SSH Mastery
FreeBSD Mastery: Storage Essentials
Letters to ed(1)
DNSSEC Mastery
Ed Mastery
Tarsnap Mastery: Online Backups for the Truly Paranoid
FreeBSD Mastery: Specialty Filesystems
ONLY FOOTNOTES
Sudo Mastery
PAM Mastery
SNMP Mastery
FreeBSD Mastery Advanced ZFS
TLS Mastery

Happy

30th

FreeBSD®

Thank you to the FreeBSD Community for your hard work and dedication in creating our favorite open source operating system! Cheers to 30 years!

FreeBSD
FOUNDATION

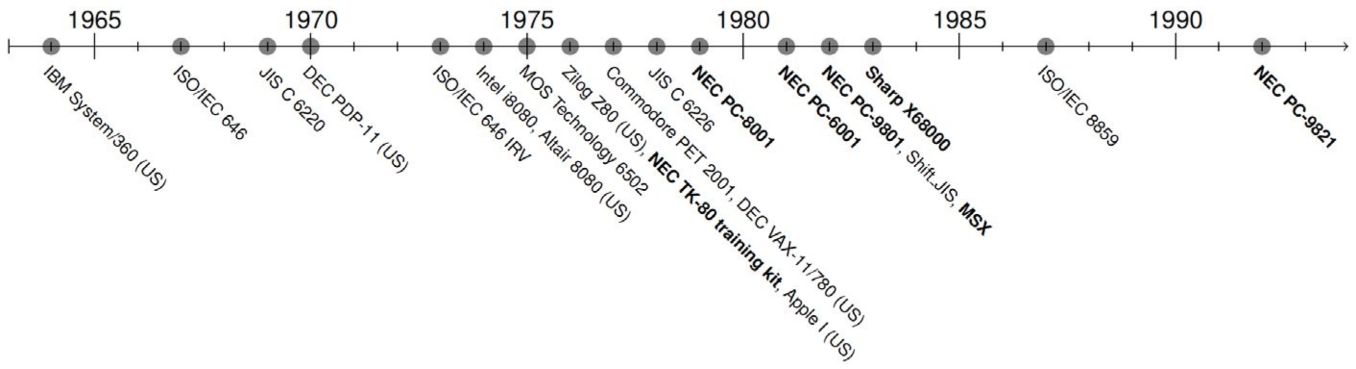
FreeBSD in Japan: A Trip Down Memory Lane and Today's Reality

BY HIROKI SATO

Japan is one of the regions where BSD and BSD-derived operating systems have been popular. Most of the evolution in computers and software technology has been in North America and Europe, however, Japan has had a significant domestic market for computers and ambitions toward the future backed by success in the semiconductor industry in the 1980s. In this same timeframe, commercialized Unix and BSD operating systems evolved. While much happened in the Japanese computer industry and user communities, these stories are rarely told because the records are only in Japanese, there are piles of failures, and there was less significant impact outside Japan.

Although the topic is dense, this article will focus on the history of BSD and Japan and the use of FreeBSD today. You will learn why and when Japanese people started using BSD and how they now feel about it.

1970-1980: Japanese-Domestic Computers



Mainframes and Large-Scale Computers

To understand Japan's computer industry, you must know how it has grown. In terms of commercial computers, mainframes were most popular in the US from the 1950s on. IBM System/360, in 1964, is one of the most famous models. Japanese companies started to develop computers after the research phase, and since the required technologies were premature in the 1960s, compatible models were developed under business partnerships with RCA, Honeywell, and General Electric. The players in Japan were Fujitsu, Hitachi, NEC, Toshiba, Mitsubishi, and Oki, and they were supported by funded projects from the Japanese government to develop the domestic industry. They are also known as "NTT family companies," which have developed communication equipment and infrastructure for some time.

In 1970, minicomputers such as DEC PDP-11 and VAX became popular as office computers in the US, and Japanese companies developed similar models. While imported minicomputers were used with UNIX and BSD as engineering workstations in universities, research institutes, and some software development companies, they gained a small market share because they were too tricky for office users and had relatively poor support service compared to the domestic models.

Personal Computers

Figure 1: PC-8001 (a photo under CC-BY-2.0 taken by Tom West, a user of Flickr)

Personal computers also emerged in this timeframe. Japan obtained microprocessor technology in the 1970s, however, the microprocessor business was not going well except for calculators. People did not know what was good about it. In 1974, the Altair 8800 microcomputer was released in the US and is considered the spark that ignited the microcomputer revolution. Although it arrived in Japan in 1975, it was too expensive due to import fees and thus caused no major impact. In 1976, NEC released a TK-80 training kit, an Intel 8080-based microcomputer board. To be exact, the processor was NEC's uPD8080A, a software-compatible clone of Intel's 8080. It rapidly became quite popular and recorded sales of 17,000 units for a year. This motivated NEC to develop "personal computers" and the Japanese computer industry entered the personal computer era.

In the US, several companies—such as Commodore and Apple—released personal computers in 1977, and the 16-bit IBM PC was released in 1981. In Japan, NEC released PC-8001 in 1979. It was one of the first "made in Japan" personal computers that has a Z80-based processor, 16 kB RAM, and N-BASIC (enhanced version of Microsoft BASIC). PC-8001 recorded sales of 250,000 units for four years with 40% of the market share.

NEC released the PC-8000 series in 1979, the PC-6000 series in 1981, and the PC-9800 series in 1982, and was strongest in the market until 1997. The PC-8000 and PC-6000 are Z80-based, and the PC-9800 is Intel x86 from 8086 to Pentium II. These three series were independently developed in separate divisions of NEC. To avoid competition, PC-6000 was terminated, and PC-8000 and PC-9800 were refocused for hobbyists and business users respectively.

Japanese elementary schools teach 1,026 characters in 6 years. And glyphs for Kanji require at least 16 x 16 pixels on a display or paper. This means that a single character requires 16-bit encoding and displaying one requires 16 x 16 bitmap data.

Technical Differences: Japanese Language Support

A major difference from computers in the US is that Japanese computers must handle Japanese characters. The Japanese language uses three groups of characters: Hiragana, Katakana, and Kanji. The former two groups are phonetic lettering systems with about 100 characters and the latter is logographic. While there are about 100,000 Kanji characters, you need to understand at least 2,200 selected characters to read a newspaper—Japanese elementary schools teach 1,026 characters in 6 years. And glyphs for Kanji require at least 16 x 16 pixels on a display or paper. This means that a single character requires 16-bit encoding and displaying one requires 16 x 16 bitmap data.

Obviously, computers in the US market needed non-trivial modifications to support the Japanese language. However, having all Kanji characters challenging even for Japanese models at that time because it required a lot of memory. The first models supported only Katakana using a modified ASCII code as the single-byte encoding.

PC-8001 has 160 x 100 resolution and no Kanji support. PC-9801, which is designed for office use, has 640 x 400 and Kanji support. Kanji is mandatory for serious business machines. To support it, we had to develop an encoding for Kanji characters. A standard of character set JIS C 6226 was issued in 1978, four years before the release of PC-9800. C 6226 has 6,802 Kanji and non-Kanji characters and is split into Level-1 and Level-2 based on the frequency of use. Level 1 has 3,418 characters—the minimum requirement to support Kanji. And to encode them, encoding standards were developed; Microsoft, Digital Research, ASCII (a Japanese corporation), and Mitsubishi collaboratively developed "Shift JIS" encoding in 1982. The maximum number of characters this encoding scheme can hold is 11,438, and it was sufficient to support Kanji in the JIS C 6226 character set.

Another problem was Kanji font glyphs. If all Kanji in C 6226 Level 1 are supported, it consumes 14 kB of memory to hold the 16 x 16 bitmap font. The PC-9800 series has text VRAM with a Kanji character generator. VRAM is a dual-ported DRAM used as the frame buffer for graphics display. On the IBM-PC, displaying Japanese relied on software. Reading the glyph data and rendering them into the frame buffer takes much longer than hardware-assisted VRAM on the PC-9800 series.

These language-specific factors are part of the reason why the PC-9800 series was strong in the market before Windows 95. In 1987, the PC-9800 series had over 90% of the market sales of domestic, 16-bit, personal computers. They can be divided into the PC-9801 and PC-9821 series. The former series was manufactured from 1982 to 1995, and the latter was from 1992 to 2003. All models are basically compatible. ROM BASIC or disk BASIC was popular in the early phase, then MS-DOS, and eventually Microsoft Windows arrived. This sequence of events was like the IBM-PC in the US. In short, Japanese people also enjoyed Intel x86 machines with games and business software such as Lotus 1-2-3 as well as computer geek culture such as BBS using phone lines.

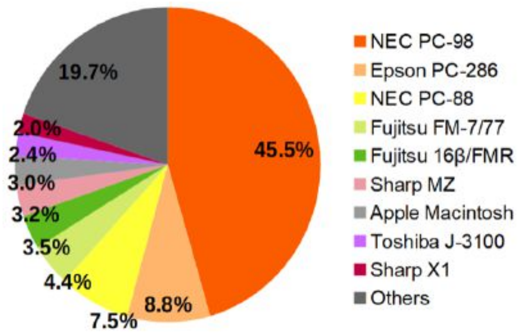
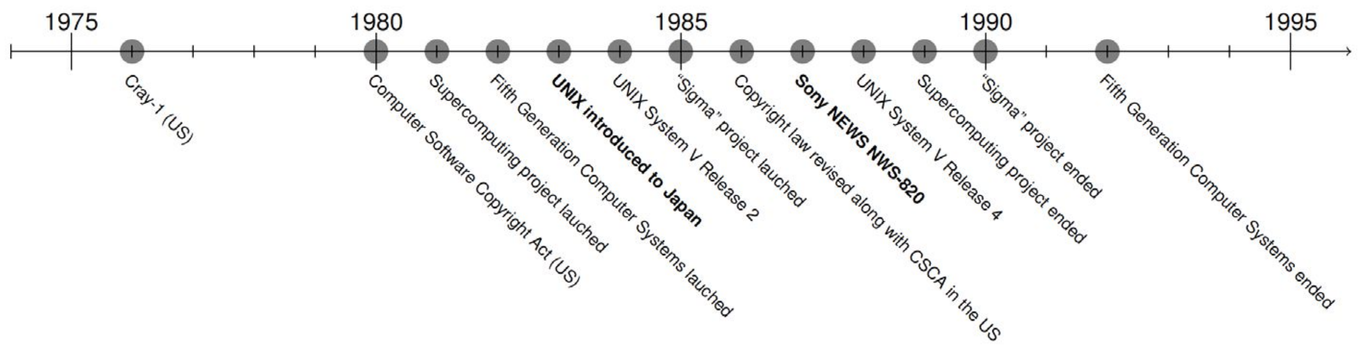


Figure 2: Usage share of personal computers at home as of 1989. 509 Japanese businessmen responded. Source: Nikkei Personal Computing Journal 1989-4-10 issue (a public domain chart from Wikipedia).

There were also exciting machines by other companies. Notable were MSX in 1983 and Sharp X68000 in 1987. They focused on hobbyists and quickly disappeared from the market. At BSD conferences, the author often asks, "what was your first computer?" Amiga 500? Sinclair ZX80? If you ever see me, please share your story.

The IBM-PC and PC-9800 are incompatible regarding the I/O port and memory mapping, but porting software is possible and straightforward except for handling the Japanese language. So, when 386BSD was released, some people were interested in porting to PC-9800.

1980-1990: National Software Projects


1975
1980
1985
1990
1995
Cray-1 (US)
Computer Software Copyright Act (US)
Supercomputing project lauched
Fifth Generation Computer Systems lauched
UNIX introduced to Japan
UNIX System V Release 2
"Sigma" project lauched
Copyright law revised along with CSCA in the US
Sony NEWS NWS-820
UNIX System V Release 4
Supercomputing project ended
"Sigma" project ended
Fifth Generation Computer Systems ended

Before moving on to the 386BSD porting topic, let's see hardware and software technology in the Japanese industry at that time. In the 1980s, mainframes or minicomputers were still popular as larger-scale computers in Japan. Many applications written in FORTRAN or COBOL were used in government facilities, offices, banks, etc. and major electric companies partnered with US companies to learn the technology, including hardware and software. Until 1975, foreign companies were not able to enter the Japanese market. Fostering the industry was done under the protection of the Japanese government. Even after that, in 1979, Fujitsu surpassed IBM in sales in the domestic computer market and the companies gained technology comparable to the US.

Until 1980, all commercial computers in Japan were clones or slightly modified versions of computers from the US. Japan managed to get technology, but there needed to be more innovation and better originality. To overcome this situation, the Japanese government started software projects in the same manner as hardware and the major electric companies joined them. The motivation came from the fact that software businesses emerged in the US in the same timeframe. In 1980, US President Jimmy Carter signed the Computer Software Copyright Act into law and IBM started to dissolve the bundling of hardware and software and to sell software independently. The same law in Japan took effect in 1986, after lengthy discussion. Until that time, IBM's software was the source of software technology, and the code was available in the public domain. Japanese companies could learn everything from MVS for System/360, which is an operating system released in 1974 for the IBM mainframe System/360 series. They needed to pay software license fees if they were to continue to depend on IBM's software.

In 1982, a project called Fifth Generation Computer Systems (FGCS) launched. FGCS aimed at hardware and software for artificial intelligence. This was one of Japan's first attempts at developing original hardware and software technology. They stuck to Prolog, though LISP was popular in the field of AI outside Japan. Parallel computing and concurrent logic programming were the target technologies. This project ended in 1992 with processors, operating systems, and application software that were dedicated to logic programming. Then an article was published in International Herald Tribune, entitled *The Japanese Give Up on New Wave of Computers—Vaunted Threat to US Of a New Superiority Fails to Meet Its Goals*. FGCS produced no practical commercial impact, although the research results provided academic contributions. This decade-long project had yet to be able to displace the US leadership in super-computing.

In 1985, the Sigma Project launched. The government thought Japan needed a standard development platform to foster more software developers. The lack of programmers was believed to be a big problem in the 1990s as the number of computers and software business was rapidly growing. The Sigma platform was designed by companies involved in the mainframe business and transferring the accumulated technology to domestic office computers was the primary target.

Until 1980, all commercial computers in Japan were clones or slightly modified versions of computers from the US.

Sigma and UNIX Workstation Business

From the end-users' perspective, the derivations of this project were Sigma Workstation and Sigma OS. The goal was 32-bit 1 MIPS MPU, 4 MB RAM, 80 MB HDD, and IEEE802.3 (Ethernet). AT&T's UNIX System V 2.0 (SVR2) and some functionality of 4.2BSD for the OS. Sigma Workstation is a SysV UNIX workstation, but the OS is supposed to be a heavily modified version to fit the specific needs of Sigma Tools, which were supposed to be reusable, useful software distributed through the Sigma Project's network infrastructure.

The workstation and OS were being developed at companies independently—there were Sigma-compliant workstations and Sigma-compliant OSes. While the hardware of the Sigma workstation was a relatively easy target, the development of Sigma OS was chaotic. UNIX rapidly evolved in the same timeframe. SVR2 was released in 1984, SVR3 in 1986, and SVR4 in 1988. The project could not catch up with this release speed and their SVR2-based implementations became obsolete. Since Sigma OSes were developed independently, compatibility was also a big issue.

All the big companies already had UNIX workstations in their product lineups, so motivation to develop the Sigma OS was low. The market size of computers was 1.3 million units as of 1988. The workstation held only about 25,000 units. Sun, Apollo, and HP had already appeared and very few big companies considered Sigma very seriously.

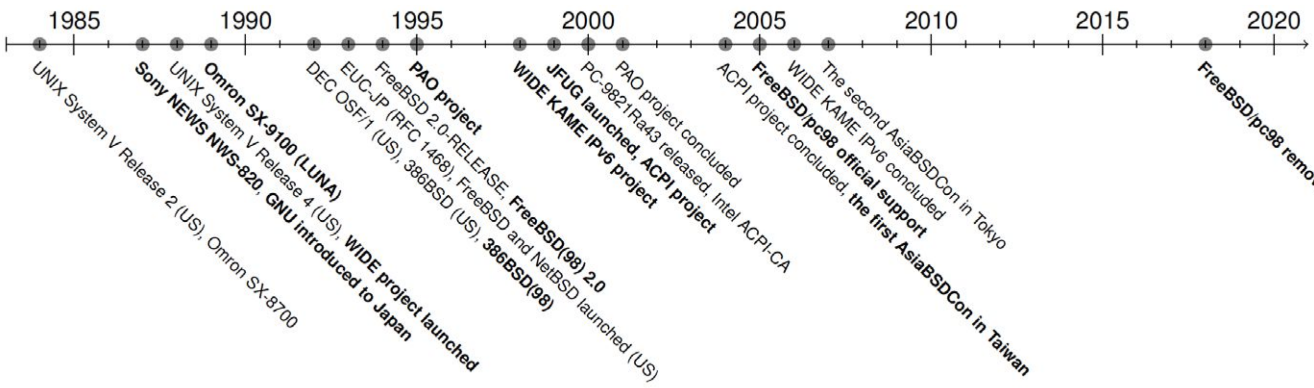
The project had only two years to implement the first version. Eventually, 199 companies were involved. In 1990, *Nikkei Computer*, a magazine covering the computer industry, summarized *"five years and one hundred million US dollars did not produce anything."* Developing the non-hardware part was too ambitious.

On the other hand, UNIX was introduced to Japan and Kouichi Kishida formed the Japan UNIX society. In 1983, two years before the Sigma project launched. Mr. Kishida was the founder of SRA (Software Research Associates, Inc., a private software company in Japan since 1967) and the first person to use UNIX in a commercial company in 1980. While the mainframers joined the Sigma project, some companies decided not to join and SRA and Sony were some of them. They started to develop a UNIX workstation named Sony NEWS, which has nothing to do with NeWS, Network extensible Window System.

Meanwhile, Sun Microsystems was founded in 1982 and they were selling Sun-3, 68k-based workstations with BSD when the Sigma project began. Sony wanted to develop their own workstations and thought making them a part of Sigma was a good idea. Mr. Kishida was disappointed with the plan of Sigma at the very early stage, and he and Sony agreed to go their own ways. The goal was to develop something better than Sun. The advent of Sony NEWS greatly impacted the Sigma project because NEWS was affordable and performant compared to Sigma workstations available at that time. The Japanese government reportedly requested a delay in the product release. UNIX users in Japan loved Sony NEWS.

At the end of the Sigma project, only four companies, NEC, Hitachi, Fujitsu, Omron, were selling Sigma workstations and they were just modified versions of their UNIX workstations. They work as Sigma if running Sigma OS and they also support SysV, BSD, or traditional OSes which they had for business purposes. It is believed that no innovative result was obtained, however, software engineers in Japan learned UNIX through Sigma.

1990-2000: BSD and FreeBSD in Japan


1985
1990
1995
2000
2005
2010
2015
2020
UNIX System V Release 2 (US), Omron SX-8700
Sony NEWS NWS-820, GNU introduced to Japan
UNIX System V Release 4 (US), WIDE project launched
Omron SX-9100 (LUNA)
DEC OSF/1 (US), 386BSD (US), 386BSD(98)
EUC-JP (RFC 1468), FreeBSD and NetBSD launched (US)
FreeBSD 2.0-RELEASE, FreeBSD(98) 2.0
PAO project
WIDE KAME IPv6 project
JFUG launched, ACPI project
PC-9821Ra43 released, Intel ACPI-CA
PAO project concluded
ACPI project concluded, the first AsiaBSDCon in Taiwan
FreeBSD/pc98 official support
WIDE KAME IPv6 concluded
The second AsiaBSDCon in Tokyo
FreeBSD/pc98 removed

Sony NEWS

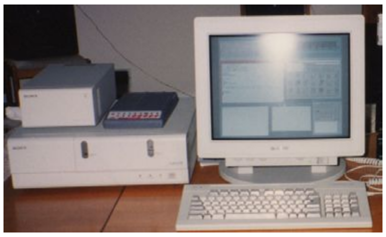


Figure 3: Sony NEWS workstation (a public domain photo from Wikipedia). As mentioned, many Japanese companies were involved in the workstation business. UNIX was the promising OS, but SysV vs BSD had cast a shadow over the market in a different way from the US because of Sigma. On the other hand, Sun's workstation gradually gained market share.

From a historical point of view, Sony NEWS and Omron LUNA were "made in Japan" machines that seriously considered porting BSD and were officially supported by 4.4BSD released in 1993. Other than those, the HP-9000 300 series, DECstation 3100/5000, and SPARCstation 1/2 were supported. CSRG at UC Berkeley was using the HP-9000 300 series as the reference machine then.

The first model of the Sony NEWS series was released in 1987. While the Sigma project stuck to System V because AT&T was driving it, developers at Sony and Mr. Kishida believed BSD was better. It was 68k-based until 1990, and the later models adopted MIPS R3000, R4000, and R10000. NEWS-OS 1.0 was a ported version of 4.2BSD and supported Shift JIS instead EUC-JP. Versions 2.0 to 4.0 were 4.3BSD-based. The final version was 6.1.2 in 1996, but it was switched to SVR4-based after 1993. WIDE project members used many Sony NEWS workstations as a research platform and had a collaborative project with Sony to port 4.3BSD and

4.4BSD. LUNA was 68k-based, like the HP-9000 300 series, and NEWS was MIPS-based similar to the DECstation 3100. Some of the source files in 4.4BSD were shared because of it.

The famous UNIX wars between OSF and UI were from 1988 to 1994. The above models were discontinued around 1993. There were few Japan-specific innovative developments during the time Japanese software engineers were involved in the porting efforts and the vendors caught up with the evolution in SysV and BSD through them.

Anyway, Japanese software engineers learned UNIX and BSD and there were engineering workstations that ran them. What's next?

FreeBSD(98) and FreeBSD/pc98

In 1992, 386BSD version 0.0 was released. Of course, it was for the IBM-PC/AT, so it did not run on PC-9800. Students at KMC (Kyoto University Micro Computer Club) with Internet access and Sun workstations ported it and released 386BSD(98) in 1992. The updated versions of 386BSD were distributed as patch kits and the KMC team released 386BSD(98)-0.1 based on the patch-kit 0.1.

The FreeBSD and NetBSD project started in 1993. FreeBSD 1.0 and NetBSD 0.8 were released in 1993 and based on 386BSD-0.1 and patch-kit 0.2.2. There was no porting effort for a while after 386BSD(98)-0.1. In 1994, four students were independently trying to port FreeBSD. They communicated via Netnews, and the result was eventually released as FreeBSD(98) 1.1.5.1.

The FreeBSD project was working on 4.4BSD-Lite-based FreeBSD as 2.0-RELEASE. It was released in November 1994. The FreeBSD(98) development team tried to catch up with it and released FreeBSD(98) 2.0-ALPHA in December 1994. After that, it was maintained until FreeBSD 5.4R-Rev01. From the 5.5-RELEASE and 6.0-RELEASE, the FreeBSD for PC-9800 series started to be released as an officially supported distribution named FreeBSD/pc98.

The differences between PC-9800 and IBM-PC/AT are bootstrap stages, memory layouts, I/O port mapping, and devices. It was easy to maintain the ported versions once routines to handle these hardware-dependent parts were written. Although it was actively maintained until 2010, it became Tier-2 at the point when FreeBSD dropped floppy disk support in the boot loader. Most of the models do not support bootstrap from a CD-ROM drive. Before the 12.0-RELEASE in 2018, the support was unfortunately removed.

Projects and User Groups of FreeBSD

As FreeBSD(98) was actively developed, there were BSD user and developer communities in Japan. UNIX users had BSD on workstations such as Sony NEWS and the SPARCstation with SunOS4. There were many developers in companies who knew BSD well. In 1988, Jun Murai, a professor at Keio University who is recognized as the father of Japan's Internet infrastructure, established WIDE, a research project having to do with the Internet and related technology. Many software engineers in companies and professors at universities have been involved, and BSD was the popular research platform. One of the notable projects was the IPv6 network stack started in 1998. It was called KAME, which means a turtle in Japanese. IIJ—Internet Initiative Japan, the first commercial ISP in Japan, NEC, Toshiba, Hitachi, Fujitsu, and Yokogawa Electric joined the project. Not only IPv6, but emerging technology such as mobile IP and IPsec have also been implemented and evaluated using BSD for eight years. They were also active on standardizing bodies, such as IETF, and took leadership in academic research and commercial network design in Japan. The results have been merged into FreeBSD, NetBSD, and OpenBSD.

In 1999, several Japanese FreeBSD committers formally established JFUG, Japan FreeBSD Users Group. They tried to organize the ongoing activities by Japanese users whom a company or university had not supported. The translated version of www.FreeBSD.org was hosted as www.jp.FreeBSD.org, and several companies donated the necessary infrastructure. The author was was one of the people who started to get involved in FreeBSD via JFUG around 2000, when he was an undergraduate student. Another group of people formed JNUG, Japan NetBSD Users Group. While both user groups have become inactive these days, the members are largely overlapped and still have. still have smaller-scale meetings to discuss BSD.

As FreeBSD(98) was actively developed, there were BSD user and developer communities in Japan.

One of the notable JFUG projects was PAO. It started by Tatsumi Hosokawa at Keio University (hosokawa@FreeBSD.org) around 1995 before JFUG was formed, and the goal was to improve mobile device support. At that time, IBM ThinkPad 535 and DEC HiNote Ultra were popular, and there were a lot of small gadgets in the Japanese market. Laptops (which Japanese people call "notebook" PCs) needed device drivers and frameworks to support power management, suspend/resume, battery, hot-plugging of PCMCIA cards, etc.

PAO was the distribution name which, interestingly, means a portable, rounded tent used by nomadic groups in Central Asia. Laptop users are nomads, and the PAO project provides them a safe dwelling!

Hosokawa began to work on 2.1.0 and led the PAO project. The results have been merged into the main tree, including various drivers and changes for dynamic resource allocation for kernel subsystems to support hot-plugging. 3.5.1-RELEASE was the last PAO snapshot release, and the project was concluded in 2001, as all the changes were merged to the 5.x and 4.x branches. The previous version of FreeBSD USB stack was also from a member of this project. The current version has been rewritten by Hans-Petter Selasky (hps@FreeBSD.org).

Another related project was ACPI. Takanori Watanabe (takawata@FreeBSD.org) and Mitsuru Iwasaki (iwasaki@FreeBSD.org) started this project in 1999 to implement the ACPI driver. While PAO was based on APM BIOS, newer models needed ACPI support because APM BIOS was being replaced with ACPI. The announcement to the FreeBSD project was made in October 1999. In 2000, Intel released ACPI CA (Component Architecture) as a reference implementation under a permissive license. It partially replaced their efforts and eventually merged into the main tree. Around 2004, most ACPI-related discussions were held on freebsd-acpi in English. Because of that, this JFUG project was concluded in 2005.

Linux eventually dominated the market in Japan.

The activities were more than just technical ones. Providing information about FreeBSD in Japanese was one of the critical roles of JFUG. The members actively worked on translating manual pages, the official website, and book-like documents, such as FreeBSD Handbook. Kazuo Horikawa (horikawa@FreeBSD.org) led the manual page translation, and Ryusuke Suzuki (ryusuke@FreeBSD.org) worked on the others.

However, this kind of contribution did not always go well. In the late 1990s, some Japanese BSD developers were working on the dynamic configuration of the kernel to support hot-plugging as described. A developer tried to extend BSD config(8) framework to support it. His implementation was named newconfig, which extended the bus framework to handle dynamic configurations in addition to the traditional static ones. It was merged into PAO, and PAO depended on it for a long time. The development group wanted to merge newconfig to the main tree. Meanwhile, new-bus was discussed in the FreeBSD project. The new-bus, what we have now in the main tree, was another implementation that can handle dynamic resource allocation. Technically speaking, the capabilities are similar. The challenge for the team was how to convince the FreeBSD project members to accept their patch.

Supporters of newconfig believed it was a less intrusive extension to the existing framework and also possible to maintain device driver compatibility with NetBSD. The new-bus was discussed in a closed mailing list, and they were frustrated to the unclear decision process. The team rewrote a lot of drivers and released snapshots to prove the usefulness and practicality. In the end, new-bus was committed and the team gave up newconfig. It was a sad thing that a substantial number of project members disappointed and went to NetBSD. PAO was rewritten as a new-bus based implementation before the merge happened.

From the author's perspective, JFUG had fulfilled the duty well for about five years. Over 50 people were active in social and technical contributions to JFUG activities. And at the same time, they recognized that Japanese-only groups did not work well for development. The members who were still interested joined the FreeBSD project directly. While JFUG was no longer active in and after 2005, the author was still one of the JFUG members and felt the user community was rapidly shrinking. To overcome this situation, he started to host AsiaBSDCon in Tokyo in 2007—with great help from George Neville-Neil (gnn@FreeBSD.org), one of the authors of *Design and Implementation of FreeBSD* and a columnist for *ACM Queue* magazine. The author had attended BSDCan and EuroBSDcon several times to learn from people outside Japan. BSD conferences are beyond the topics covered by this article, but AsiaBSDCon is one of the activities that has remained in Japan.

BSD in Business and Industry—Wide Adoption of Linux

So, what was it like after 2005, and how is it today?

The number of FreeBSD users around 2000 was the largest in the PC-UNIX market, partially because only FreeBSD had descent PC-9800 series support. Linux and NetBSD were also used but had no stable ported version. FreeBSD's user base was strong, even after IBM-PC/AT became popular. On the commercial front, FreeBSD and NetBSD were used for Internet servers and used to implement embedded systems such as a router box and an Ethernet board for printers. This happened because many software engineers were familiar with UNIX and BSD in the 1990s. Both hobbyist and business user communities were quite active and there were a lot of magazines, books, and conferences relating to FreeBSD and Linux.

On the other hand, Linux eventually dominated the market in Japan. During 2000 to 2010, a significant change occurred in the Japanese UNIX market. As mentioned, several big computer manufacturers were selling UNIX workstations. Whether SysV or BSD-based, the sales were descending because imported machines such as Sun Microsystems were strong and they were looking for a way to offload the development costs. On the other hand, in the 1990s, Internet companies such as Yahoo! started to use PC-UNIX and commodity hardware for their businesses. The future of expensive UNIX workstations became questionable, especially for vendors who were just following the US market.

In 2000, Toshiba, Hitachi, Fujitsu, and IBM Japan independently announced that they would support Linux as their business foundation. IBM Japan's move in May 2000 made a significant impact. Hitachi did this in September 2000, and Toshiba did it in October. By the end of the 1990s, their UNIX workstation business was based on commercial UNIX such as Solaris and HP-UX and not on domestic versions of SVR4 or BSD. After the announcements, each company started to build a business structure to adopt PC-UNIX and went for Linux-centric businesses. They have established Linux support companies and, more importantly, have collaboratively established a Linux education infrastructure.

Japan has a unique hiring practice when big Japanese companies hire new university graduates. It is called "simultaneous recruiting of new graduates." Most students hunt for jobs before graduation from university, seeking "formal letter of employment." The government controls this process, and companies are allowed to begin the selection process, usually in April. Attaining a position as a regular employee at any other time of year, or any later in life, is generally difficult in Japan. This means that most new employees have no business experience at the time of hiring and the companies are responsible for their education. Thus, the above big companies formed LPI-Japan (Linux Professional Institute Japan) as a non-profit offering the education service in a vendor-neutral manner. LPI (Linux Professional Institute, Inc.) is a Canadian non-profit organization founded in 1999 for Linux certifications. LPI-Japan was established as a Japan branch.

A substantial amount of investment by big companies was made to LPI to develop the Japanese version of the educational materials and exams. As a result, most new employees at big computer companies learn Linux as the reference platform. The number is more than 1,000 per year. Official adoption of Linux and this education system increased the user base, and young people had no chance to learn UNIX or BSD. This was one of the reasons why Japanese BSD user groups lost popularity. Older people still enjoy BSD, but there is no such motivation for younger generations. While FreeBSD was one of the most popular PC-UNIX in 1999, Linux became standard around 2005. All business sectors that used BSD, such as embedded system development, were also shifted toward Linux. Many old developers who used to work on BSD at companies have also left.

Sadly, this situation warrants no optimism about the future of FreeBSD in Japan. The FreeBSD Foundation continuously approaches enterprise FreeBSD users in Japan to bridge them with the FreeBSD project. Even with the nationwide movement toward Linux, several companies still use BSD. IIJ has been using NetBSD to build their router products, and Sony's famous gaming consoles, PlayStation 4 and 5, are using the FreeBSD kernel as the core component. The business use of FreeBSD has been changed from a complete OS to a component-level adoption. For example, FreeBSD's network stack is often used to implement TCP/IP functionality in various products. The author suggests that the FreeBSD project should recognize the demands for these use cases, and investment in them would be worth doing to reinforce its strength even after Linux became the standard choice.

Conclusion

This article has traced some aspects of a 50-year history of the domestic industry in Japan. Although the author tried to make this as accurate as possible by using various references and his own experiences, please let him know if you find anything in error.

The BSD community in Japan, including all flavors of BSD-derived operating systems, has been active for quite a long time. Since communication in English was always challenging, the activities were often invisible from places where the official BSD projects run. Regarding FreeBSD, many people outside Japan have helped communicate by visiting Japan. I would like to thank Jordan Hubbard (jkh@FreeBSD.org), one of the founders of the FreeBSD project, Warner Losh (imp@FreeBSD.org) an ex-FreeBSD Core Team member and long-term BSD contributor, Murray Stokely (murray@FreeBSD.org), a release engineer of 4.X-RELEASE, and George Nevile-Neil for their great support and, of course, AsiaBSDCon attendees over 15 years.

HIROKI SATO is an assistant professor at Tokyo Institute of Technology. His research topics include transistor-level integrated circuit design, analog signal processing, embedded systems, computer network, and software technology in general. He was one of the FreeBSD core team members from 2006 to 2022, has been a FreeBSD Foundation board member since 2008, and has hosted AsiaBSDCon, an international conference on BSD-derived operating systems in Asia, since 2007.

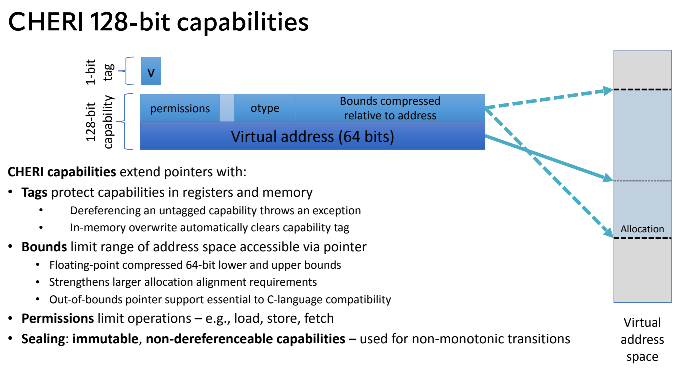
A Dozen Years of CheriBSD

BY BROOKS DAVIS

Since late 2010, the CHERI research project at the University of Cambridge and SRI International has striven to develop, demonstrate, and transition to real-world products architectural extensions providing memory safety and efficient compartmentalization. CheriBSD, our CHERI-enhanced fork of FreeBSD, is one of the most important products of our work. Adapting FreeBSD to support CHERI has informed our architectural changes while demonstrating that our ideas can work at the scale of a large modern operating system.

A Brief Introduction to CHERI

CHERI extends existing architectures (Armv8-A, MIPS64 (retired), RISC-V, and x86_64 (in development)) with a new hardware type, the CHERI capability. In CHERI systems, all access to memory is via CHERI capabilities either explicitly via new instructions or implicitly via a Default Data Capability (DDC) and Program Counter Capability (PCC) used by instructions with integer arguments. Capabilities grant access to specific ranges of (virtual, or occasionally, physical) memory via a base and length, and can further restrict access with permissions, which are compressed into a 128-bit representation (64-bits for the address and 64-bits for the metadata). In memory and in registers, capabilities are protected by tags that are cleared when the capability data is modified by a non-capability instruction or if a capability instruction would increase the access the capability grants. Tags are stored separately from data and cannot be manipulated directly.



CHERI 128-bit capabilities

CHERI capabilities extend pointers with:

- **Tags** protect capabilities in registers and memory
    - Dereferencing an untagged capability throws an exception
    - In-memory overwrite automatically clears capability tag
- **Bounds** limit range of address space accessible via pointer
    - Floating-point compressed 64-bit lower and upper bounds
    - Strengthens larger allocation alignment requirements
    - Out-of-bounds pointer support essential to C-language compatibility
- **Permissions** limit operations – e.g., load, store, fetch
- **Sealing**: **immutable**, **non-dereferenceable capabilities** – used for non-monotonic transitions

Our initial work on CHERI extended the MIPS64 architecture as part of the DARPA CRASH program. In 2014 we began collaboration with Arm, exploring the possibility of adapting CHERI to the Armv8-A architecture. In 2017 we began a port of CHERI to RISC-V informed by both our MIPS work and our collaboration with Arm. This port was performed as part of the DARPA MTO SSITH program. Our collaboration with Arm became public in 2019, with the announcement of the £190m Digital Security by Design program, which has resulted in the Morello architecture prototype, a SoC based on the Neoverse N1 core used in cloud platforms such as Amazon Web Services' Graviton nodes.

We have designed CHERI capabilities to be suitable for use as C and C++ language pointers and have modified the Clang compiler to support them in two modes. In hybrid mode, pointers annotated with `__capability` are capabilities, while other pointers remain integers. In pure-capability mode, all pointers are capabilities, including implied pointers such as return addresses on the stack. Coupled with kernel support and modest changes to the C startup code, run-time linker, and standard library, we have produced a memory safe C/C++ runtime environment called CheriABI[1]. The refinement of this environment is a key thrust of our work on CheriBSD alongside creation of a pure-capability kernel environment and explorations of temporal memory safety and compartmentalization.

In addition to memory safety, CHERI enables fine-grained compartmentalization. Because all memory accesses are via capabilities, the portion of an address space a given thread can reach is defined by its register set and the memory that can be (transitively) reached from there. With appropriate mechanisms to transition between register sets, we can switch rapidly among compartments. Various CHERI implementations implement different mechanisms for this; which one(s) are most appropriate to a commercial implementation remains the subject of active research.

What is CheriBSD?

CheriBSD is FreeBSD modified to support CHERI. But what does that actually mean?

When the kernel is compiled for CHERI, the default ABI is a pure-capability ABI (CheriABI) where all pointers including system-call arguments are capabilities. We also support both hybrid binaries and standard FreeBSD binaries via the freebsd64 ABI compatibility layer derived from the freebsd32 32-bit compatibility layer. Likewise, we build libraries, programs, and the run-time linker for CheriABI by default and build libraries for hybrid binaries that are installed in `/usr/lib64` just like `/usr/lib32` for freebsd32. All of this means that by default users are presented with a memory-safe Unix userland which retains the ability to run unmodified FreeBSD binaries.

The kernel can be compiled as either a hybrid or a pure-capability program. This adds some complexity to the changes we need to make (every pointer to userspace requires an annotation (`__capability`) for hybrid), but we started out with hybrid in the early days of the project when we didn't have strong C compiler support, and pure-capability kernels do have somewhat higher inherent overhead due to the increased pointer size. All internal kernel development is done with pure-capability support in mind. This work includes ensuring that all access to userspace is via a capability[2], changes to the VM system to create capabilities when allocating memory and altering device drivers including the DRM GPU framework to use capabilities.

Historically, CheriBSD has mostly been a compile-from-source proposition. This is familiar to FreeBSD developers, and has many benefits; however, for people who just want to port a custom codebase to CHERI, that's a big hurdle. With the release of Arm's Morello prototype, we've started producing full releases with an installer and packages. We use a lightly customized version of the FreeBSD installer that adds support for installing a GUI desktop environment based on KDE and removes some dialog boxes we deemed confusing. The GUI environment is comprised of packages built from our fork[3] of the FreeBSD ports collection. Because not all software has been ported to CHERI, we build two sets of packages and build and install two versions of the `pkg` command with `pkg` being a script that redirects callers to the other names. There is a CheriABI set which is managed by the `pkg64c` command and installed under `/usr/local` and a hybrid set managed by the `pkg64` command and installed under `/usr/local64`. Most of the desktop environment is CheriABI binaries, with the big exception being web browsers (a CheriABI port of Chromium is in progress). Post-install, hybrid packages are also useful for installing not-yet ported software such as emacs and Morello LLVM.

Beyond memory safety, CheriBSD plays host to much of our research on software compartmentalization. In the MIPS era, we implemented a compartmentalization framework (libcheri) that we applied to the integrated version of tcpdump. While we did not port this work forward to RISC-V and Morello, it informed our early thinking on the use of compartmentalization for increased availability. Our latest release contains a library compartmentalization model where the dynamically linked library runs in its own sandbox. The current implementation is experimental but shows considerable promise at compartmentalizing programs with little or no modification. Additionally, in a stack of development branches, we have a co-process compartmentalization model in which multiple processes share the same virtual address space, relying on CHERI to provide memory isolation. Coupled with a trusted switcher component, this enables extremely fast transition of execution from a thread in one process to a thread in another process. We expect a signification portion of future work on CheriBSD will be motivated by compartmentalization, as we refine our models in the face of an increasing corpus of compartmentalized software.

CheriBSD is both a research artifact under active development and a product servicing dozens or hundreds of users doing their own R&D. Even for users targeting other domains (embedded systems, Linux, Windows, etc.) CheriBSD is currently the easiest place to test CHERI technologies.

Why CheriBSD?

Historically hardware research has focused on bare metal benchmarks or embedded operating systems. They have a lower memory footprint and usually execute fewer instructions (important for simulation) as well as simply having less code to understand and change. Unfortunately, results don't always scale to real world operating systems and it's too easy to hand wave at things like dynamic linking as "a small matter of programming." Adapting FreeBSD was undeniably more work, but doing so has given us the ability to evaluate CHERI with an unmatched level of realism. Some of our ability to use a real, multi-user operating system stems from timing. In 2010, FPGAs big enough to run simple cores supporting full instruction set architectures at decent speeds (100MHz) were finally available for reasonable prices ($5-10k vs $100k or much more). Likewise, desktop computers were big enough and fast enough to support full system emulators like QEMU with relative ease.

People do ask: "why not Linux?" FreeBSD offers a number of advantages for a research project like CHERI. On the technical front, FreeBSD's integrated build system and early adoption of LLVM has made it relatively easy to build large corpuses of software with experimental compilers (C/C++ compiler research is mostly done in LLVM today) both in the base system and via the ports tree. The clean ABI (Application Binary Interface) abstractions to support Linux binaries and the freebsd32 32-bit compatibility layer greatly simply ABI experimentation. (By contrast, Linux supports a single alternative ABI that must be 32-bit, and Windows does all the translation within userspace via a DLL.) While not part of our initial decision, it later emerged that choosing FreeBSD over Linux was fortuitous due to extensive use of `long` in the Linux kernel for both integers and pointers, which cause capabilities to be invalidated. While people are working on Linux ports at Arm and elsewhere, the use of `long` is a major stumbling block.

Historically hardware research has focused on bare metal benchmarks or embedded operating systems.

On less technical fronts, BSD and FreeBSD have a long history of successful research and transition to real-world products. From the Fast File System (FFS) and sockets APIs for TCP/IP in 4.2BSD to Capsicum and pluggable TCP/IP stacks in FreeBSD, many ideas in daily use by billions of people have been incubated in BSD. One factor in this success is FreeBSD's permissive license. Publishing our work under the two-clause BSD license means potential adopters can easily evaluate our work even within companies with proprietary operating systems and strict controls around GPL-licensed software. This has enabled successes like a very positive evaluation[4] of past Windows security vulnerabilities by the Microsoft Security Response Center.

Ultimately, the success of CHERI depends on adoption by multiple operating systems. Today, CheriBSD leads the pack with the latest features and most active research.

A CheriBSD Timeline

- **October 2010**—The first CHERI Project begins
- **May 2012**—CheriBSD running on CHERI-MIPS CPU.
- **November 2012**—Sandboxed custom application demo on CheriBSD.
- **October 2013**—Migrated development to git.
- **January 2014**—CheriBSD compiled with CHERI LLVM.
- **November 2014**—Sandboxed tcpdump (sandbox per-decoder).
- **June 2015**—CheriBSD with compressed capabilities (128-bit vs 256-bit).
- **September 2015**—CheriABI pure-capability process environment up and running.
- **January 2016**—Began merging RISC-V support from FreeBSD.

There are over 1800 commits to the FreeBSD source tree with "Sponsored by:" lines indicating they were likely funded by work on CHERI.

- **April 2019**—CheriABI paper wins Best Paper award at ASPLOS 2019.
- **September 2019**—Morello CPU, SoC, and board announced.
- **August 2020**—CheriBSD ported to CHERI-RISC-V.
- **June 2021**—Pure-capability kernel (RISC-V)
- **January 2022**—First official Morello boards ship. CheriBSD aided in validation.
- **May 2022**—CheriBSD 22.05 release targets Morello board users. This is an initial support release focusing on the installer and basic package infrastructure. The package set included a basic set of tools including the Morello LLVM compiler.
- **December 2022**—CheriBSD 22.12 release includes library-based compartmentalization, ZFS support, DRM support for the on-die GPU, and a basic GUI environment where everything except the web browsers is a pure-capability program.

Benefits to FreeBSD

Research projects like CHERI can provide significant benefits to FreeBSD. We have contributed changes ranging from typo fixes to a port to the RISC-V architecture. We've also given talks, added new committers, and introduced many organizations to FreeBSD.

There are over 1800 commits to the FreeBSD source tree with "Sponsored by:" lines indicating they were likely funded by work on CHERI[5]. This amounts to over 1.5% of commits outside `contrib` and `sys/contrib` since January 2011. These contributions have been made possible by funding over a dozen committers so far including two new ones.

Notable contributions:

- **External toolchain support**—I contributed initial support, later enhanced by Baptiste Daroussin to add the `CROSS_TOOLCHAIN` variable used today. This functionality was added to support compiling with the CHERI Clang compiler as well as custom compilers developed for two other projects: TESLA and SOAAP. TESLA enabled construction and dynamic enforcement of temporal logic assertions, and SOAAP allowed exploration of compartmentalization hypotheses for large applications.
- **Unprivileged installs and images**—I ported the ability to store the owner and permission metadata of installed files in a `METALOG` file from NetBSD in January 2012. This allows the `intallworld` command to be run without root privileges. Coupled with support in `makefs` it was then possible to build UFS filesystems of either endianness. Followed by my complaints that there wasn't a way to embed a filesystem in a partition table without mounting it, Marcel Moolenaar contributed the `mkimg` command in March 2014 to complete the required toolking.
- **MIPS64 maintenance**—While FreeBSD had a MIPS port (essential for our use), it didn't have a lot of users, and didn't get much maintenance. We did quite a bit to keep it running, and improved things that hit our pain points. It served us well, but we breathed a sigh of relief when we'd transitioned our last work to RISC-V and MIPS was removed from the main branch.
- **RISC-V port**—While MIPS had served us well, and we were trying to build a community around our base BERI MIPS FPGA implementation, it become clear that the research community was moving to RISC-V. As a result, we tasked Ruslan Bukin with porting FreeBSD to RISC-V; he landed it in the tree in January 2016.
- **Arm N1SDP platform support**—The Morello platform is based on Arm's N1SDP development board. Ruslan worked with Andrew Turner to support the attached peripherals, including the PCI root complex and IOMMU in 2020.
- **Cross build from macOS and Linux**—In September 2020, Alex Richardson contributed a make wrapper (`tools/build/make.py`) that allows `bmake` and other build tools to be bootstrapped on a non-FreeBSD system. This allows builds on users' non-FreeBSD desktops and laptops, and in CI environments that don't support FreeBSD. Alex and Jessica Clarke maintain this support on an ongoing basis.
- **Consolidated compatibility system call stubs**—Historically, system calls have been declared in `sys/kern/syscalls.master` with compatibility versions declared in `sys/compat/freebsd32/syscalls.master`. Developers would fail to keep them in sync or misunderstand if they needed a compatibility wrapper. As part of adding two ABIs to CheriBSD, I extended the `syscalls.master` file format and stub generation code with enough understanding of ABIs for the script to know what is required. Now there is only one list of system calls and freebsd32 has a `syscalls.conf` that specifies ABI details. I upstreamed this work in early 2022.
- **Unprivileged, cross release builds**—As part of supporting hundreds of users of Morello hardware we needed to start producing releases. Most of our CI and build infrastructure does unprivileged builds on Linux hosts so Jessica closed the last gaps in unprivileged builds and cross build support allowing us to build release images in February 2022.

In addition to these changes, we've made many smaller improvements along the way. With over 1,800 commit messages, I'd use up all my word count use listing a fraction of them.

Beyond technical contributions, the CHERI project has contributed to the community. We've added two new committers: Alexander Richardson and Jessica Clarke. We've also had contributions from graduate students including Alfredo Mazzinghi and Dapeng Gao. From short-term contracts to full-time employment, at one time or another we've supported committers including: Jonathan Anderson, John Baldwin, Ruslan Bukin, David Chisnall, Jessica Clarke, Brooks Davis, Mark Johnston, Ed Maste, Edward Napierala, George Neville-Neil, Philip Paepes, Alexander Richardson, Hans Petter Selasky, Stacey Son, Andrew Turner, Robert Watson, Konrad Witaszczyk, and Bjoern Zeeb

Further, we've exposed many people to FreeBSD as a research platform. We've been part of three DARPA programs (CRASH and MRC from the I2O program office and SSITH from MTO) where people gained FreeBSD experience as part of supporting and evaluating our work. With the UK Digital Security by Design program, dozens of organizations are now using CheriBSD in demonstration projects funded by Digital Catapult and the Defence Science and Technology Laboratory (DSTL).

Conclusions

As research projects go, CHERI has been enormously successful, and FreeBSD has played a major role in that success. Having a well-integrated base OS and monolithic build system, coupled with the ports collection's massive scale, has allowed us to demonstrate CHERI's potential to a wide audience—leading to real-world implementations ranging from Arm's server-class Morello design to Microsoft's CHERIoT microcontroller. In turn, CheriBSD development has led to significant improvements in FreeBSD from the RISC-V port to build system improvements.

Footnotes

1. https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201904-asplos-cheriabi.pdf
2. A few subsystems access userspace via the direct map, and those are validated rather than using capabilities directly.
3. https://github.com/CTSRD-CHERI/cheribsd-ports
4. https://msrc-blog.microsoft.com/2020/10/14/security-analysis-of-cheri-isa/
5. A portion of lines matching "Sponsored by:.*DARPA" are from the CADETS project which focused on Dtrace work, but the vast majority are CHERI related.

BROOKS DAVIS is a Principal Computer Scientist in the Computer Science Laboratory at SRI International and a Visiting Research Fellow at the University of Cambridge Department of Computer Science and Technology (Computer Laboratory). Leads development of CheriBSD, a fork of FreeBSD supporting CHERI ISA extensions. He has been a FreeBSD user since 1994, a FreeBSD committer since 2001, and has served 4 terms on the core team.

Thank you!

The FreesBSD Foundation would like to acknowledge the following companies for their continued support of the Project.

Because of generous donations such as these we are able to continue moving the Project forward.

FreeBSD FOUNDATION

Are you a fan of FreeBSD? Help us give back to the Project and donate today! freebsdfoundation.org/donate/

Please check out the full list of generous community investors at **freebsdfoundation.org/donors/**

Platinum

arm NETFLIX

Meta

Gold

BECKHOFF

JUNIPER NETWORKS

Silver

Tarsnap STORMSHIELD

modirum vmware

How ZFS Made Its Way into FreeBSD

BY PAWEL DAWIDEK

The story of how the ZFS file system made its way into the FreeBSD operating system is a tale of passion for programming, love for technology, and a journey that led to my most valuable contribution to the FreeBSD project. It was the summer of 2005. Although I'm not great with dates, I do remember the circumstances surrounding my first encounter with ZFS. I was with my friends in the Masurian region of Poland, near one of its 2,000 beautiful lakes. One of my friends worked at a Polish telecommunications company at the time, and they used a lot of Sun Microsystems hardware and Solaris. He brought a printed copy of the announcement they had received from Sun, describing a new file system that had been in development for some time and was about to be released as part of OpenSolaris. But before I continue, let's take a step back in time for a bit of context…

I fell in love with programming at first sight. I was 12 years old and my cousin introduced me to Basic on a C-64..

Love at First Sight!

I fell in love with programming at first sight. I was 12 years old, and my cousin—Tomek—introduced me to Basic on a C-64. I was hooked. I felt like a young god: you take this soulless piece of hardware, create a program, and watch it come to life! This was the coolest thing; I couldn't think of anything better. As a result, I was never into video games. Back then, growing up in a small Polish town, it wasn't easy to find people interested in programming, so I was pretty much on my own (well, except when I was testing the limits of Tomek's patience).

When I switched to the Amiga 500, I finally found some friends from the demo scene with whom I could exchange my work using 3.5-inch floppy disks through the postal service. Latency was not the best, but I didn't complain. When my next computer—the Amiga 1200—started to show its age, it was clear it was time to move on. I knew Microsoft Windows was not for me. I'd tried Linux briefly, but I still wasn't convinced. Finally, a friend pointed me to FreeBSD. The installation was a breeze, I couldn't ask for a better experience. Ha! If you didn't cringe at that last sentence, then you clearly haven't had the "pleasure" of using sysinstall. No, the installation wasn't a breeze—it took multiple attempts for me to finally enjoy my first FreeBSD system. My understanding was that sysinstall must be like Navy Seals Hell Week, where the strong are separated from the weak, where real hackers are forged! And I made it! I set my next goal and dream to not only be a hacker who can install FreeBSD but to be a kernel hacker and a FreeBSD committer.

I accomplished my goal in 2003 when I officially joined the FreeBSD project as an src committer. Yes, I threw a party to celebrate that. Since I joined, I have worked in many areas of the system, but mainly with the GEOM framework. The GEOM framework in FreeBSD sits between disk drivers and file systems, allowing plug-ins of various transformations, like mirroring, RAID, block-level encryption, etc. I really liked the GEOM design and loved working with it, so I had decent experience with at least part of the storage stack. As for file systems, I knew enough to stay away from VFS, which is one of the most complicated parts of the kernel.

The curse. UFS has been the default file system in FreeBSD since the very beginning. In fact, UFS is much older than FreeBSD itself. UFS2, which was introduced in FreeBSD 5.0, addressed some shortcomings of UFS1, but some important ones were still not addressed. The main issue was the fsck time after a system crash or power outage. With disks getting bigger and bigger, fsck could take many hours to complete. The solution to this problem was obvious—we either needed to add journaling to UFS or port some other journaled file system to FreeBSD. Easier said than done. In Linux, there were plenty of file systems to choose from, and many people tried to port them to FreeBSD, but for some weird reason, those ports were never finished, so we ended up with extfs without journaling, read-only ReiserFS, and read-only XFS. There was even a read-write HFS+ port from Mac OS X, but, of course, no journaling, and I remember at least one failed attempt to add journaling to UFS. What was this mystery? Had the UNIX gods turned their backs on us?

Let's come back to my vacation in Masuria. My friend starts to read the ZFS announcement while my eyes and my mouth open wider and wider: Pooled storage—you can create as many file systems as you want, and they will all share available space. Unlimited snapshots that take no time to create. Unlimited clones. Built-in compression. End-to-end data verification. Self-healing of corrupted data. Transactional copy-on-write model—always consistent—no need for fsck. EVER. How? How is that even possible? This is not an evolution, but a clear revolution in file systems. I remember dreaming about this perfect marriage: the best file system running on the best operating system… Wouldn't that be amazing?

A few months later, ZFS was released, and it took not only the open-source community but the entire storage industry by storm. Some people hated it, most loved it, others feared it, but nobody was ignoring it. It was called the last word in file systems. It was called a rampant layering violation. However, it was never called just another file system. Almost every operating system wanted

ZFS: Linux userland port started under FUSE, DragonFlyBSD announced ZFS would be ported soon, and Apple started to port ZFS to Mac OS X. ZFS will be everywhere soon, just not in our beloved FreeBSD...

To write this article, I had to analyze a lot of IRC logs from that era. What struck me the most was how much skepticism there was about ZFS itself: too complicated, too many layers, just demoware, design flaws are going to be found soon, just wait for the first disaster story, it will never be ported to a community-developed OS, it's just hype, it's hilarious. I guess people are used to the fact that if something looks too good to be true, it often is. Fortunately, love is blind, and I didn't notice this at that time.

After waiting 10 months after the ZFS release and seeing nobody starting the work, I thought I might as well give it a shot. With almost zero knowledge about the VFS layer, I'd likely fail quickly, but who could stop me from trying? At the very least, I'd learn something new. My porting work started on August 12, 2006. To not raise people's hopes too high, the perforce branch I created had "This is not a ZFS port!" as its description. My initial estimates were six months to have a read-only prototype.

I must admit that even though ZFS was not my creation, working on ZFS was the most engaging project in my career. I love to work hard, and I love to work late. I like to hyperfocus on projects, and I have been fortunate to work on many amazing projects. For many years, I was one of the most productive FreeBSD committers while still developing my own business. But no other project kept me awake for 48 hours straight with almost no breaks and only short naps between those 48 hour periods. What I'm about to tell you sounds impossible, even for me today, but it did happen, I can assure you :)

When I work on big projects, I still like to have something that I can run as quickly as possible and then incrementally implement missing bits. The first step was to port userland components like libzpool, ztest, and zdb. This went mostly ok. The next challenge was to compile and load the ZFS kernel module. When you try to load a kernel module with missing symbols, the FreeBSD kernel linker reports the first missing symbol and returns an error. I had so many missing symbols that I had to hack the linker to report them all at once. It was taking too much time to fix them one by one. After five days, I loaded zfs.ko for the first time. In theory, there were four main meeting points between the FreeBSD kernel and the ZFS code:

1. On the bottom of the stack, we have to teach ZFS how to talk to block devices in FreeBSD, so this means connecting ZFS to GEOM, which in GEOM terms is creating a consumer-only GEOM class. Because of my GEOM experience, it was trivial.
2. On the top of the stack, we need to connect ZFS to FreeBSD's VFS, so port the ZPL layer.
3. Also, on the top of the stack, ZFS storage can be accessed through ZVOLs, and because ZVOLs are block devices, this is again GEOM, but this time provider-only GEOM class.
4. The last component is the /dev/zfs device that is used by userland ZFS tools (zfs(8) and zpool(8)) to communicate with the ZFS kernel module.

Porting the ZPL layer and attaching ZFS to FreeBSD's VFS was, of course, the hardest part. The first kernel mount on FreeBSD happened on August 19, 2006, so, one week in. After exactly ten days (and nights) of work, I had a read-write prototype ready. I could create pools, create file systems, and mount them, create ZVOLs whose behavior was really stable, create files and directories, list them, and change permissions and ownership. My initial estimates of six months for a read-only prototype turned out to be "a little" off. There was still a huge amount of work to do, but the encouragement from the community gave me the needed motivation to continue and finish the project. In 2007, ZFS was officially released with an experimental status in FreeBSD 7.0, and in FreeBSD 8.0 (2009), it was declared as production-ready.

No other port that was announced before my work came to fruition, so I guess to reverse a curse you just need to work hard enough :)

Just to be fair, a working read-write prototype in ten days wouldn't have been possible without one very important decision that was made by the ZFS creators at the early stage of ZFS development. They wanted most of the code to compile in userland, so it could be easily tested and debugged. This was an immense help in my porting efforts because most of the code was already highly portable.

When I work on big projects, I still like to have something that I can run as quickly as possible and then incrementally implement missing bits.

This was an amazing journey, and I wish every software developer a similar experience. While writing this, I'd like to recognize some people. First, I'd like to thank Jeff Bonwick, Matt Ahrens, and the whole ZFS team at Sun for creating this revolutionary technology and always supporting my work. Alexander Kabaev, for all his patience and help with VFS. Robert Watson, for all his encouragement and for being a role model I always looked up to. Kris Kennaway, for being a ruthless early tester—we didn't call him BugMagnet for nothing. Martin Matuska, for stepping up and taking over ZFS maintenance when the time came. And last, but not least, I'd like to thank the entire FreeBSD community—there is nothing that brings more satisfaction than feeling that your work is appreciated and provides real value.

A lot has happened in the 20 years since ZFS was initially released: NetApp started a legal battle against Sun. Apple discontinued the ZFS port. Licensing issues prevented ZFS from being a native component of the Linux kernel. Sun Microsystems no longer exists, and the new owner closed ZFS development. And yet, this great technology prevailed, and the project lives on under the OpenZFS flag. Long live OpenZFS! Long live FreeBSD!

PAWEL DAWIDEK is Co-Founder and CTO at Fudo Security, a security vendor building products for secure remote access. He is also involved in the FreeBSD operating system where he works on security- and storage-related projects, like GELI disk encryption, Capsicum capability and sandboxing framework, jail containers, ZFS and various GEOM classes. Pawel's passion outside of technology is training Brazilian Jiu Jitsu.

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

FreeBSD FOUNDATION™

AArch64: Bringing a New FreeBSD Architecture to Tier-1

BY ED MASTE

FreeBSD descended from 386BSD and began with support for a single CPU architecture, the Intel 80386. Support for a second architecture, DEC Alpha, arrived in FreeBSD 3.2 and support for 64 bit x86 (amd64) came next. The concept of support tiers wasn't yet fully cemented, but amd64 was promoted to Tier-1 status in 2003. The 64-bit Arm architecture AArch64, also known as arm64, obtained Tier-1 status in 2021. We'll explore what that means and how we got here.

Bringing up a new Tier-1 architecture in FreeBSD is a challenging task, and it requires significant effort to ensure that the architecture is fully supported, stable, performant, and compatible with the existing FreeBSD ecosystem.

The FreeBSD Foundation saw potential in 64-bit Arm, and also learned of other entities with an interest in a FreeBSD port to the platform.

Tier-1 Status

The FreeBSD project website [https://docs.freebsd.org/en/articles/committers-guide/#archs] documents the three support tiers. Tier-1 references fully-supported architectures, Tier-2 is developmental or niche architectures, and Tier-3 are experimental architectures.

The documented Tier-1 status refers primarily to the guarantees the FreeBSD project makes with respect to the architecture, including generating release artifacts, providing prebuilt packages, support by the security team, and backwards compatibility goals across updates. Tier-1 also implies that the platform is actively maintained, regularly tested, and receives timely bug fixes and security updates. A Tier-1 platform is expected to be fully integrated into the FreeBSD build system so that all components of the tool chain are functional and developers can easily build, install and maintain the operating system on that platform.

Tier-1 status also covers some implicit characteristics, such as hardware availability. FreeBSD doesn't explicitly require Tier-1 platforms to be widely available or popular, but in practice, Tier-1 status requires that a variety of hardware platforms exist and are available with a reasonable cost. This is because FreeBSD relies on a combination of community support and vendor contributions to maintain and improve its support for different hardware platforms and to build and test third-party software for the architecture.

Tier-1 platforms are also expected to be self-hosting—that is, it is possible to build a new version of the kernel, C runtime, userland tools, and the rest of the base system on FreeBSD on that platform.

Platform Genesis

Like several other platforms, FreeBSD/arm64 began with the interest of a motivated developer. Andrew Turner is a longtime FreeBSD/arm developer who started looking at Arm's AArch64 architecture shortly after it was announced. The FreeBSD Foundation saw potential in 64-bit Arm, and also learned of other entities with an interest in a FreeBSD port to the platform. The Foundation formed a project to coordinate and sponsor both Andrew Turner and engineering firm Semihalf, with the support of Arm, and CPU vendor Cavium.

The earliest commit in the FreeBSD that references arm64 added build infrastructure for the kernel-toolchain build target. As the name suggests, this target builds the tool chain (the compiler, linker, and so on) that is then used to compile, link and convert a kernel. Clang was part of the FreeBSD base system at the time this work was done, so compiler support was fairly straightforward. However, at the time, FreeBSD still included an older version of the GNU ld linker, which predated support for AArch64. Thus, the early build support relied on having the aarch64-binutils port or package installed, and it used the provided linker automatically.

First kernel change for arm64:

```
commit 412042e2aeb666395b3996808aff3a8e2273438f
Author: Andrew Turner <andrew@FreeBSD.org>
Date:   Mon Mar 23 11:54:56 2015 +0000

        Add the start of the arm64 machine headers. This is the subset
        needed to start getting userland libraries building.

        Reviewed by:   imp
        Sponsored by:     The FreeBSD Foundation
```

After several years of development, FreeBSD had a basic, but functional, self-hosted FreeBSD/arm64 port, with some ports and packages available. A lot of development effort, debugging, performance tuning, documentation, and other work remained, but FreeBSD was on the path to adding another architecture to the supported list. FreeBSD 11.0 became the first release to include arm64 support and installable artifacts, as a Tier-2 platform.

Tool Chain

Beginning of arm64/AArch64 support:

```
commit 8daa81674ed800f568b87f5e4b8881d028c92aea
Author: Andrew Turner <andrew@FreeBSD.org>
Date:   Thu Mar 19 13:53:47 2015 +0000

        Start to import support for the AArch64 architecture from ARM. This
        change only adds support for kernel-toolchain, however it is expected
        further changes to add kernel and userland support will be committed
        as they are reviewed.

        As our copy of binutils is too old the devel/aarch64-binutils port needs
        to be installed to pull in a linker.

        To build either TARGET needs to be set to arm64, or TARGET_ARCH
        set to aarch64. The latter is set so uname -p will return aarch64 as
        existing third party software expects this.

        Differential Revision:  https://reviews.freebsd.org/D2005
        Relnotes:         Yes
        Sponsored by:     The FreeBSD Foundation
```

One of the first requirements for a Tier-1 platform is to have a fully-supported, integrated tool chain. Clang is the primary compiler used in building FreeBSD and it has received significant and ongoing AArch64 development effort from several large companies. Thus, compiler support was quite good for the entire duration of platform bring-up.

Other tool chain components, like the linker, debugger, and miscellaneous binary utilities needed more work. As initial FreeBSD/arm64 porting effort was nearing completion, FreeBSD still used the GNU binutils linker ("BFD linker") and had not updated the version of the linker for some time due to licensing concerns. As a result, the linker included in the base system did not support AArch64 and initial support depended on having a binutils port or package installed. We made this as convenient as possible for end users, but it did not meet the requirements for a Tier-1 architecture.

Fortunately, rapid progress was also being made on LLVM's LLD linker within the LLVM community. LLD offered the potential for much faster link times, facilitated optimizations not available with the BFD linker, and wider architecture support. At the end of 2016, we were able to switch to using LLD as the system linker for FreeBSD/arm64. In fact, it was the first FreeBSD architecture to do so.

FreeBSD uses the ELF Tool Chain project for miscellaneous binary utilities such as `strings` or `strip`. These have some machine-dependent functionality (such as lists of relocation types), but the effort required to add arm64 was relatively small.

The final tool chain component that required significant development effort was LLDB, the LLVM family debugger. Fortunately development work was being done to support other operating systems, and it required only incremental effort to add FreeBSD.

We were able to merge much of this tool chain work into the FreeBSD 11 stable branch, and FreeBSD 11.1 was the first release to avoid workarounds and include a functional linker.

Ports and Packages Collection

FreeBSD provides over 30,000 third-party software packages in its ports collection, and many of these have architecture-dependent characteristics. Machine-dependent infrastructure (e.g., controls for a given port to opt-in or opt-out of building on a given architecture) are fundamental parts of the ports tree. FreeBSD/arm64 was available as a Tier-2 architecture and FreeBSD community members experimented and discovered ports that failed to build. These were either fixed, or excluded from building on aarch64 if appropriate. Mark Linimon was one of the key developers who took on this work.

The goal of bringing FreeBSD/arm64 to Tier-1 brought with it some additional requirements of the ports tree. The ports collection does not have an official hierarchy or tier categorization of ports, but there are some ports that are critical. This includes ports that provide tool chain components or other dependencies required to build large ports of the full collection. We had to make sure that these were available and consistently built for FreeBSD/arm64.

Bringing a new architecture to Tier-1 status requires the support and agreement of several teams within the FreeBSD project.

We also need to build package sets in a timely manner for Tier-1 architectures, which requires capable server hardware. The FreeBSD Foundation purchased servers from Ampere Computing, and the project received additional servers donated by Ampere. This hardware allows the arm64 package sets to be built on the same weekly cadence as x86

Support from FreeBSD Teams

Bringing a new architecture to Tier-1 status requires the support and agreement of several teams within the FreeBSD project. This includes the ports management and package management teams as mentioned above, along with the security team, release engineering team, and the core team.

The Release Engineering team is responsible for building and testing release artifacts including ISO and USB memory stick images, as well as cloud computing targets. These artifacts can be cross-built from other architectures, so arm64 build hosts are not absolutely required by the Release Engineering team, but test and QA hardware is needed.

To be Tier-1, an architecture requires the security team to provide source updates for security issues and errata as well as binary updates via freebsd-update.

Finally, the core team's support is necessary to coordinate with the other teams, with the community, and make the official declaration that the platform is officially Tier-1

Hardware Ecosystem

An implicit requirement for a platform to be Tier-1 is the availability of suitable hardware, as alluded to earlier. Hardware is needed at many different price/performance combinations:

- high-end servers to build packages,
- mid-range, sever-class hardware for developer workstations, remote access for porting and testing, and so on,
- low-end embedded style platforms for ubiquitous testing and developer use,
- cloud resources at various levels for development, testing, and production.

AArch64 started with some notable gaps in the available hardware, in particular, related to mid-range (and mid-price) platforms for developer and porting efforts. There were very few options in the mid-late 2010s. The SoftIron OverDrive 1000 was a well-priced, capable system in a convenient developer form factor based on the AMD A1100 processor. Unfortunately, both the A1100 and the OverDrive 1000 were discontinued not long after being introduced.

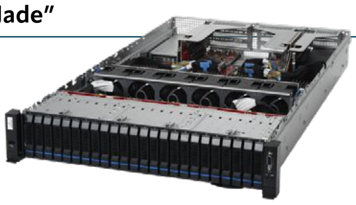
Hardware availability continues to improve with platforms like the Raspberry Pi 4 and Pine A64-LTS at the lower end, Apple devices and the Microsoft AArch64 developer platform in the middle, and high-end Ampere Altra-based server systems. AArch64 virtual machines are also offered by major cloud vendors, using either Ampere platforms or a bespoke CPU design (AWS' Graviton).

Bringing FreeBSD/arm64 to Tier-1 status required a significant investment of time and resources. The 64-bit Arm ecosystem has captured a significant portion of the server market, with no signs of slowing down. FreeBSD will benefit from tapping into this market with this Tier 1 platform.

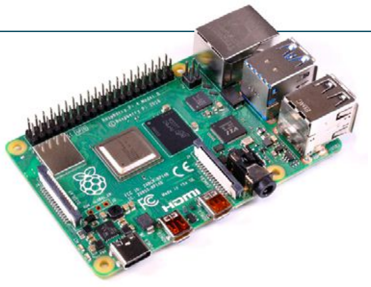
ED MASTE is the Senior Director of Technology for the FreeBSD Foundation, managing the Foundation's technology roadmap, development team and sponsored projects. He is also a member of the elected FreeBSD Core Team for the current term. Aside from FreeBSD, he has contributed to a number of other open-source projects, including LLVM, ELF Tool Chain, QEMU, and Open vSwitch. He lives in Kitchener-Waterloo, Canada, with his with his wife Anna and children.

AWS Graviton

Ampere "Mount Jade"

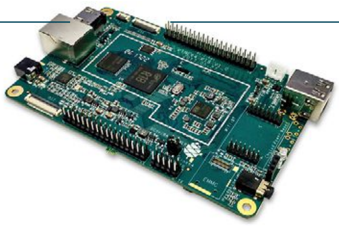


Raspberry Pi 4



Microsoft Arm Developer Kit

Pine A64 LTS



A Brief History of *FreeBSD Journal*

BY GEORGE V. NEVILLE-NEIL

It was May of 2013, and a couple hundred developers had descended on the University of Ottawa for our annual North American conference. Tutorials, talks, and wandering around ByWard Market in Ottawa while looking for things to eat were part of the deal. Dinners on the first two nights were held in residence, in a big room on the ground floor with a lot of us milling around and discussing the finer points of whatever sub-systems we happened to be working on.

BSDCan 2013

I was talking with a few folks from the project and remember Robert Watson standing just to my left, as he said, "What we could use is a magazine about FreeBSD, something with good technical content, written by developers, for the community." He had a gleam in his eye that I've seen when he's about to ask someone to do something they probably want to do but should think about more carefully before actually agreeing. And yet, they say "Yes."

The FreeBSD Journal got its start due to a lot of luck, some great people, and some skill.

Robert knew I'd been working on a magazine called *Queue* for over a decade, so I guess he thought I was the kind of person who might want to start a magazine, which, of course, I was. Robert knows me well and that isn't always an advantage. I answered with, "Huh....," and then looked off into the distance to consider it. "We could...," I continued, and at that point, I knew I would at least give it a try.

Most important things we achieve in life happen with a mixture of luck, skill, and knowing the right people. The *FreeBSD Journal* got its start due to a lot of luck, some great people, and some skill. The *Journal* is the intellectual grandchild of a venerated trade journal from the 1980s, *UNIX Review*. The *Review* was started and run by Mark Compton, who would later be brought in to help create ACM's practitioner-oriented publication called *Queue*, where I would serve on the editorial board, and by so doing, learn enough about publishing to help start the *FreeBSD Journal*.

It was as a member of the *Queue* editorial board that I came to understand how a magazine is produced. With software, we're used to listening to the sound that deadlines make as they go whooshing by, and then recovering by pushing the latest code to the servers, or whatever. But when you publish in print, there is a drop-dead date, and if you miss it, you don't get a magazine that month, full stop. That kind of pressure runs right back into the editorial process, meaning you must not only select authors who write well, but they also need to do it on a deadline. And if too many authors miss the deadline, you get a magazine that's very slim, or nonexistent. You also want to establish a board that is respected, collegial, and gets along well. Imagine, if you will, a room full of luminaries arguing a about a topic, but instead of anyone grandstanding and taking up all the air in the room, they build upon each other's ideas in a sort of techno-nerd version of improv. The best editorial boards and board meetings are like that, and that's what we set out to do with *FreeBSD Journal*.

I thought I knew where to start, and the first person I talked with was the executive editor of *Queue*, Jim Maurer, who had also been running a boutique magazine publishing business for a number of years. *Queue* had published a few pieces by folks in the FreeBSD community and Jim had mentioned that he had enjoyed working with them. It's always cheered me to know that it was the quality of people in the FreeBSD community that had convinced Jim to work on the *Journal*. With Jim on board, I was slightly less panicked, because someone I trusted and who knew the industry had said, "Yes," rather than, "You're crazy!"

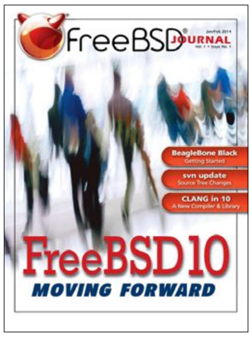
Working in open source usually means working with limited budgets, and I knew I wasn't going to get the FreeBSD Foundation to fund monthly, in-person, editorial board meetings at fine restaurants—per the *ACM Queue* model, but I still needed to convince a group of smart people from the community to form an editorial board. All I could offer was some recognition and the chance to work with other smart people on an interesting, shared project, which, effectively, is why many of us started working in open source in the first place. I began emailing people I thought might be interested and made a direct pitch to each one, "Can I convince you to..." —is my usual opener when I want someone to write or join a board or contribute to the project. It's not subtle, but it does get the job done!

I convinced eight people to give it a try, and the first *FreeBSD Journal* editorial board was: John Baldwin, Daichi Goto, Joseph Kong, Dru Lavigne, Michael W. Lucas, Kirk McKusick, Hiroki Sato, Robert Watson, and me. I was shocked by my good luck.

For a magazine like the *Journal*, the editorial board is its beating heart. The board members understand technology and also writing about technology, and they are also the Rolodex (an ancient, rotating paper product that has been replaced by the contacts app in your phone) which is to say, they also have to be connected and reach out to people who also can understand and write about technology. An essential quality for an editorial board member is the ability to ask for the moon and not take "No" for an answer. In the academic world, people must "publish or perish," but not in the practitioner world. How the board consistently comes up with great authors who produce great articles is shrouded in mystery, but I suspect it has something to do with being very friendly and buying people strong drinks at conferences.

With the board in place, we set out to find funding. We were planning a print magazine, and while the authors were kind enough to write for free, we had to pay for editorial and production services and printing costs. Two early backers were the FreeBSD Foundation and iX Systems. Later, we would solicit advertising, and several companies who were using FreeBSD took out ads to help support the effort.

With the basics in place, the editorial board began bi-weekly, video calls to hash out issue topics. Before we approached even a single author, we had to work out what topics we wanted to cover, issue by issue, for the first year of the magazine. Each topic had to be broad enough to include several articles, and we were also looking to establish regularly appearing columns to give the *Journal* continuity. Our goal was to publish the first issue at the start of 2014, the Jan/Feb issue, and I'm happy to say we made our deadline and the magazine got off to a great start. The main articles in the first issue covered "The New Toolchain" in FreeBSD 10 (Clang/LLVM) as well as booting FreeBSD on the Beaglebone Black, an article all about ZFS, and a piece on FreeBSD for commercial systems. We had columnists in place with Dru Lavigne taking up both the Events Calendar and "This Day in BSD," Thomas Abthorpe on the Ports Report and Glen Barber on SVN Update, which tracked new and interesting features entering the code base. We were off and running.



Over the almost 10 years of its publication, the *Journal* has evolved—as it should. What began as a magazine readers paid for, is now free and fully supported by the Foundation. The publishing technology has changed as well, moving over time from print to app-based to PDF-based delivery. The editorial board continues to change when people move on to other projects as they bring in new people and new voices.

My proudest moment with the *Journal*, second only to getting the first issue published, was when I handed over the editorial reins to a new Editor in Chief, John Baldwin. I have always felt that the best projects should outlive us, and that long term continuity and quality are the hallmarks of great systems, be they software, magazines, or any other endeavor you put your mind to. I could think of no better person to guide the *FreeBSD Journal* in its 9th year, and I'm very happy that the *Journal* is still here to help FreeBSD celebrate it's 30th.

GEORGE V. NEVILLE-NEIL works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are computer security, operating systems, networking, time protocols, and the care and feeding of large code bases. He is the author of The *Kollected Kode Vicious* and co-author with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System*. Since 2014 he has been an Industrial Visitor at the University of Cambridge where he is involved in several projects relating to computer security. His software not only runs on Earth but has been deployed, as part of VxWorks in NASA's missions to Mars. He is an avid bicyclist and traveler who currently lives in New York City.

Write For Us!

Contact Jim Maurer with your article ideas.

(jmaurer@freebsdjournal.com)

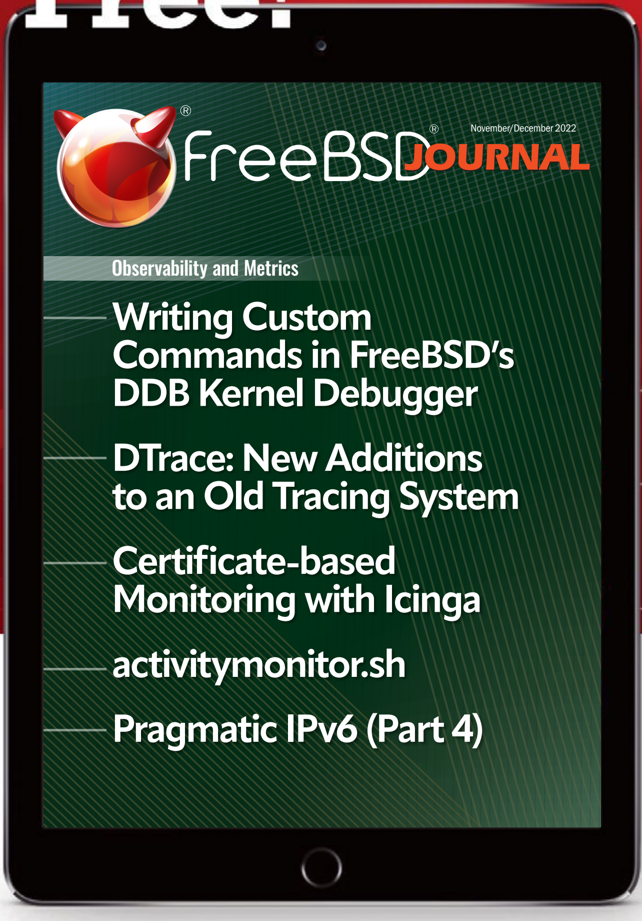
FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!



FreeBSD® JOURNAL

November/December 2022

Observability and Metrics

Writing Custom Commands in FreeBSD's DDB Kernel Debugger

DTrace: New Additions to an Old Tracing System

Certificate-based Monitoring with Icinga

activitymonitor.sh

Pragmatic IPv6 (Part 4)

2023 Editorial Calendar

- Building a FreeBSD Web Server (January-February)
- Embedded (March-April)
- FreeBSD at 30 (May-June)
- Containers and Cloud (Virtualization) (July-August)
- FreeBSD 14 (September-October)
- To be decided (November-December)

Find out more at: freebsdfoundation/journal

Installing FreeBSD 1.0: 30 Years On

BY TOM JONES

Getting FreeBSD 1.0 running on real hardware is an opportunity for me to explore computing from before my time. My earliest memories of using computers pop up with Windows 98—I'm sure there were other computers in my life. The machine on which my friends and I played Castle Wolfenstein was probably a hand me down or a surplus office machine. It wasn't until I had the first computer that was really mine (a G4 iBook in the mid 2000s) that I started to look inside things and explore operating systems.

I jumped at the chance to cover the subject of this article. I am sure the editorial board was envisioning this would cover getting FreeBSD 1.0 to run in one of the many x86 emulators, but I am not one to let an opportunity to overdo something go by.

Retro computing is a growing interest. For a long time in the UK, Charity Shops (think Goodwill) were unable to take electronics. That created a situation where throwing away computers was the easy option for those unwilling to try to sell them on a cold Sunday morning at a car boot sale.

This key factor created an absolutely perfect situation for our computing history to be disposed of and destroyed. For the last decade, I have tried to save the machines I could as they came by. This is why I have an inventory of 50 machines waiting for space in the Center for Computing History, a vt320 on my desk, and a DEC-PRO350 as a conversation piece in the hackerspace.

A collection is one thing, but these machines don't want to be in a museum, they want to run!

In the last decade, there has been a growing movement of retro computing enthusiasts. There are channels like Action Retro, Lazy Game Reviews, and This Does Not Compute which go to obsessive extremes to push old machines to their absolute limits. This can be done by installing accelerator cards from when the machines were still supported and through adapters, modern storages, and peripherals that get machines to scream-along at speeds the original owners could only have imagined.

In this world of renewed interest in classic computers, we get to learn from the large enthusiast community and use the excellent tools and adapters they have created. But we also have to pay the price of the popularity, and with the extra interest and demand, prices for old hardware have gotten silly.

How Do I Find a Machine to Run FreeBSD?

If this was a question from a normal person (not someone deranged trying to relive 1993) I think most of us that have used FreeBSD for a while would offer similar advice:

- research your hardware and figure out what is supported,
- if you are getting a new machine, make sure it has support.

I wanted to buy a machine that managed to meet three criteria—era accurate—supported by FreeBSD 1.0—(ideally) supported by 386BSD.

Why era accurate? Whatever machine I got hold of, I wanted to be able to give a long life. Its first role is to run FreeBSD 1.0, but I also wanted to be able to photograph it, take it to conferences, and use it as a platform for learning more about computing in the foundational times of free BSD UNIX. "What is the newest hardware that FreeBSD 1.0 will run on" is a different article, and not the article I wanted to cover. Era accurate seemed to be the path of least resistance.

Supported by FreeBSD 1.0 should be self-evident, but why 386BSD? I want more from the machine than just 1 article. I want to be able to show people where things came from—386BSD as a project holds a lot of mystique.

From 30 years in the future, it is very difficult to understand what sort of machines were actually around in 1993. This is a form of the set dressing problem that period TV shows and movies hit all the time. You can't really look at a magazine from the 1990s and see the typical home from the 1990s. Instead, people build up their possessions over a long period. A home from the 1990s might have a piece or two that is fashionable, but it won't look like that years' Ikea catalog.

We are saved a little bit from this by computing in the early 1990s moving so quickly—machines had shorter life times. I would set out to get hardware that would fit into the era well.

To find hardware that will actually run our OS we need to find the release notes and installation instructions. Today, figuring out if FreeBSD will run on a machine without the machine can be quite hard, but, hopefully, the original team did a better job at spelling out the supported hardware.

There is a collected CD of FreeBSD 1.0 available from the project and the Internet archive. It contains the release floppy disks, source code, and ports and packages for FreeBSD. In the CD are also the release announcement and installation instructions for various methods.

The FreeBSD 1.0 release announcement gives us some information about the minimum requirements to run. It gives us an idea about which supported processors, memory requirements and additional hardware which may work.

Finding a Machine

With a rough outline of the supported hardware, the next step is to track down a machine. First, I thought I might be able to borrow a 386 from 'someone'. After a couple of weeks of trying IRC channels and striking out, it seemed that anyone with a working 386 was quite precious of the machine and not willing to lend it to a stranger to write an article about installing FreeBSD.

My next bet was to buy a machine.

Here I encounter a problem unique to the island I am currently stuck on. We do not have old computers. Okay, that isn't true, I can get all the Amigas I want and probably a container load of Research Machines PCs from the early 2000s, but an actual 386 is a different matter.

Building from components didn't seem to be a good call. Maybe if I had some history with machines of this vintage, but finding compatible components to build a computer today can be difficult to figure out. How was I supposed to get compatibility information about 30-year-old components which were made by hundreds of different vendors.

Complete systems seemed to be a bust too. The few I saw on eBay in the UK were going for 1000GBP+, a little bit outside of my normal 386 budget.

Eventually, after complaining about my uniquely me situation on a phone call with some friends, one of them started digging around on eBay and found a Czech seller. My friend, more familiar with this period of computing was able to confirm that these smelled good and were the right vintage to fit into this article.

Some hemming and hawing and then I settled on the one with a large CPU frequency display on the front--ever one to ignore practicalities and land on aesthetics. (They were to me, functionally the same).

The seller's specifications for the machine are:

```
CPU AMD 368DX 40MHz
4MB RAM + 128kb Cache
MB Shuttle Hot-327
VGA ISA Trident TVGA 9000c 512kb
HDD ST 3243A - 214 MB
HDD Controller UMC PIO-001 RP 20070R 12
Soundcard ESS 688F Pine Technology
CD-ROM Wearnes 622 - K.O.
FDD TEAC FD-235HF
```

It was clear to me that I wanted to get this machine onto the Internet in the course of this article and so I picked up the only Ethernet interface I could find that was compatible—an Etherlink II 3c509 with BNC and AUI connections.

While the INSTALL files says that FreeBSD will run in 4MB of memory, this seemed like an easy fix, and I picked up quite cheaply an additional 32MB of RAM to give the system a little more breathing room.

Installing FreeBSD 1.0 (first attempt)

With the machine delivered, I was excited to test the hardware and get to installing FreeBSD. The system shipped with MS-DOS 6.22, an operating system that is entirely alien to me. I made sure the hardware was functional by grabbing the shareware of wolf3d and playing through a couple of levels.

The CD-ROM archive offers a few options for installing FreeBSD 1.0, all of which start with creating a set of boot floppies that build a minimal system from which the full install can be finalized.

The minimal install requires 3 floppy disks. To avoid having to rewrite a single disk, I bought a box of 'new' floppies from amazon (10 for 10GBP). I also acquired from the local hackerspace a selection of USB floppy drives to help me bootstrap into the past.

The three disks for installing are a kernel floppy, the file system floppy, and the cpio floppy. The kernel floppy comes in several different flavors depending on your SCSI controller. I don't have a SCSI controller at all, and so either would do.

Creating the disks from a modern operating system was incredibly painful. The USB floppy drives just don't want to work, but when the moon was right, I was able to create the three required disks from FreeBSD-14 with this dd command:

```
$ sudo dd if=kcopy-ah-floppy of=/dev/da0 bs=30b
```

The installation process requires you to boot with the kernel floppy in the floppy drive:

Error! Filename not specified.

Once the kernel is loaded you are presented with a prompt to insert the file system disk and continue.

Then finally you get to the installer.

I was excited when I got here, I had acquired a modern piece of hardware, a SD IDE adapter to allow me to preserve the MS-DOS 6.22 so I could continue verifying the hardware.

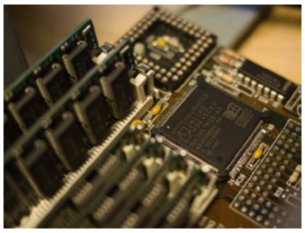
This device did not work at all for me.

Crestfallen and impatient, I decided to hell with it and wiped the MS-DOS disk entirely (who needs dos anyway, I'm getting UNIX!). Setting up FreeBSD to take the entire 200MB of spinning rust for itself.

With the disk formatted a minimal system is installed. You reboot from the kernel floppy again, this time going to a `kc>` prompt where you type 'copy' for the kernel to copy itself to the disk.

Next you reboot, and when prompted, install the cpio floppy. It will walk you through what it needs you to do given a minimal installed system. Next, you have to grab the install sets, installing at a minimum the `bin.tar.gz.xx` file set.

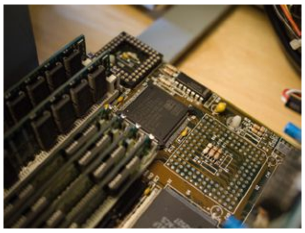
The full install media is made available across a set of dozens of disks, arranged so they can be loaded one floppy at a time onto the system to finish the install. If you were lucky enough to have tape, serial, a MS-DOS partition, or network access you could pull them all as a single archive.



By not reading ahead in the instructions and being inpatient, I made a mistake that made this much more difficult for me. My DOS partition was gone, and I didn't have a network. Tape wasn't an option at all, I've never even seen a supported tape.

Maybe I could pull the files from the CD drive? I hit the eject button on the drive for the first time (weirdly it wasn't part of my shooter-based stress test) and while the motor whined, nothing happened. The CD drive never came to life.

In the list of options is the ability to install via serial. I can do



serial. I hooked up a USB serial adapter to the serial port on the PC and began trying to get Kermit running, which does not ship on the install floppies at all.

The install instructions talk about a 'dos' floppy, but even with several checks, I couldn't find it inside the CD archive. Eventually, I caught the note early on that the 'dos' floppy is an optional disk of utilities which can be pulled from the CD archive. I put Kermit onto a floppy booted into my primordial system and copied across. Hours of fighting later, I managed to figure out the correct modern Kermit commands to speak to 1993 Kermit and started transferring files. At 9600bps, restart higher rate, I tried pushing it up and eventually managed a reliable 56k.

Three days later (or maybe an hour or two, if I am more realistic) I had the base images bin copied onto my host and could finish the install. Each set (binary, src, X) is installed with an extract shell script that comes from the preinstall environment.

Once you have installed the set you want, you need to run the 'configure' script that does the finalization of the preinstall environment and gives you a full working FreeBSD system.

Here I got to pick the hostname and IP address for interfaces.

With the install done, I booted into my new FreeBSD system and immediately posted a victory message both on social media and to the editorial team—"6 months away and the system works! That means the article is pretty much done right?"

Reinstalling FreeBSD 1.0 (second attempt)

My first install was fun, but I had installed onto very old spinning rust, and this worried me. I wanted to do some modernizations. The popularity of vintage computing like this has led to a bunch of devices to replace the finicky and annoying parts of retro computing.

Compact flash (also a rather old technology) is compatible with IDE but has the benefit of being much easier to purchase and not as likely to die randomly with a scream of horror.

I managed to get a CF adapter that lives in a PCI bracket making swapping disks for the OS really easy. Of the CF cards I bought, only 1, a 250MB one, will boot on my system. I suspect the bios freaks out seeing a 1GB (let a lone a 16GB) hard drive.

The system I had installed had some oddities and I thought they might be related to the fractured install process from above (it took several weeks of occasional effort to get the first install together).

First, when the system comes up, it always prints a message saying '**Automatic reboot in progress**' even when it doesn't reboot. While there was an adduser man page there was no command to add a new user. I have new media now, so I thought I would try again.

For the second install, the floppy phases were the same, the disks survived three months without randomly dying.

In the interim, I picked up a Farallon Ether10-T Starlet 9 base10-T Ethernet hub. This device with the correct terminators and 50 OHM BNC cables allowed the 3c509 network card I bought to pull my 386 box onto the Internet (and into the future!).

My second install was much, much smoother. I did the base install to get to the minimal system. Once booted from it, I could then use the network to grab the images. I considered ftp to move them to the 386 but spotted the NFS option in the install document and tried that. FreeBSD 1.0 will happily mount a NFS share on a FreeBSD 14 box :D.

FreeBSD 1.0 at home

Don't fancy trawling eBay to find old, expensive, noisy and unreliable hardware, but want to try out FreeBSD 1.0? I guess I can see why you might not want to reproduce my steps, be warned that you are missing out on a certain amount of joy by cheating and using QEMU.

I have created a backup of the CF card install I did, and I have placed it here: https://people.freebsd.org/~thj/freebsd-ninety-three.img.xz

Grab that image, uncompress it. With a standard install of QEMU (I used QEMU 7.0 on FreeBSD 14-CURRENT) you can launch the image with the following command:

```
$ qemu-system-i386 -hda freebsd-ninetythree.img
```

You will see from the system log that the machine doesn't find any interesting hardware. Left as an exercise to the reader is figuring out networking (if ISA emulation proves too hard, I would look into slip).

The only CF card that worked in my system was still a little small at 250MB, but there is enough of a system there for you to play with. If you want to try on a larger (or smaller disk) then you can figure out all the QEMU bits required to get it going.

With the install done, I rechecked adduser, but the command was still not there. Reading the man page this time, I discovered that the man page was just the process for adding a new user to the system! No automation for you, it is 1993.

Installing Software

The CD-ROM archive comes with quite a lot of additional third party software for FreeBSD. There are both the ports collection, software that has been modified to work on FreeBSD, and prebuilt packages of ports, and some additional software.

From ports, I built top so I could have the 386 idle in interesting ways.

I also grabbed vim and zsh from the prepackaged software to make living on the machine a little nicer.

The next thing I wanted to do was try and serve web content from the 386. I grabbed both ncsa-httpd (all releases available as a git repo) cern-httpd. Source-based distribution of software has changed a lot in 30 years. We have no idea how good we have things now.

ncsa-httpd ships with a Makefile to build which was much friendlier than the cern-httpd script, so I started with ncsa-httpd. ncsa-httpd doesn't include FreeBSD in its list of targets, but it does have NetBSD, so I tried that. Sadly, the era correct NetBSD is different enough that this didn't work out of the box. I ended up modifying the Makefile to pick up the FreeBSD build support that is in the c files, but not the build script.

With ncsa-httpd built, I went through a process of running the generated binary and creating the required directories and moving the right files it complained about being missing until I could get it to start. An install target for make would have been too much to ask for.

I whipped up an index.html file and started serving up some pages.

Not yet being willing to put my 386 on the internet, I, instead, settled for seeing how much work my 386 could actually do as a web server, by running the wrk web server benchmark.

In case you need to know a 40 MHz 386DX with 36MB of RAM can serve about 22 requests a second without any tuning.

Running X

I have managed to install X from the sets on the CD. I have not managed to get X to display anything and, instead, each attempt requires me to reboot to regain access to the system console. Trying X highlighted that I was missing virtual consoles. Grepping the documentation didn't reveal much.

I stumbled onto the `syscons` command (not driver as we have today, command). When trying to use syscons to change the virtual tty, all I got was a helpful message about needing to rebuild the kernel with syscons enabled.

Rebuilding the Kernel

There is a Makefile with a familiar `world` phase in /usr/src, but the kernel part of building the system isn't documented there at all. I stumbled into the `config` man page, which hinted enough that I could build a new kernel.

Building a new kernel is roughly:

- cd to the conf sub directory (usually /sys/ARCH/conf), for us /sys/i386/conf
- run config with a system configuration file (I copied the SYSCONS config as NINETYTHREE)
- cd to `../../compile/SYSTEM_NAME` (NINETYTHREE in this case)
- run `make depend`
- run `make all`
- copy the `386bsd` kernel binary to /
- reboot

The kernel build took 33 minutes (5 for make depend, 28 for make all).

FreeBSD 1.0 now and forever

I keep getting lost trying to get different parts of the system going. My next steps are figuring out how to get running on my extra virtual consoles, and once I find a serial mouse, running X. Rather than a straightforward install story, this project has ended up being an absolute ton of fun for me. I really didn't need a web server for this article, but I couldn't resist getting one up and running once I had the idea.

I picked up enthusiast magazines from the period to learn the interesting goings on in computing in the early 1990s. I got the Dr.Dobbs articles on 386BSD to help stage some of these pictures. Sure, I had to fight around on ebay to buy over-priced parts, but that has its own draw, there is a high when you finally find a listing for something obscure you were looking for.

FreeBSD 1.0 is like looking at your now adult little brother and seeing a small child there. All the wonder and magic of the world is back. The bugs I encountered feel entirely fixable. I am sure it is a rough system, but the edges are asking to be smoothed. I entirely understand why the early contributors got involved, it is hard to resist finding something to hack on.

This all said, I don't think you, or anyone else, should go lightly into a project like this. The hardware is difficult to find 30 years on and it won't get easier. Prices for hardware are going to go up and reliability of components will go down.

The options are to experience the joys of the platform without having to go through the suffering of drives and disks dying, finding termination for the BNC cables, or teaching software that FreeBSD is an operating system. There are many choices to emulate this hardware and much of the release information is available with source tarballs to build applications.

I think your 386 FreeBSD should be virtual, but you will have to fight me to get this 386 FreeBSD.

TOM JONES is the FreeBSD Engineering Manager at Klara Inc, he only gets to play with fun old computers in his spare time.

FreeBSD 1.0 dmesg

```
FreeBSD 1.0.2 (GENERICAH) #0: Sun Nov 14 18:22:24 PST 1993
  root@gndrsh.cdrom.com:/usr/src/sys/compile/GENERICAH
CPU: i386DX (386-class CPU)
real mem  = 38006784
avail mem = 35909632
using 819 buffers containing 6709248 bytes of memory
Probing for devices on the ISA bus:
pc0 at 0x60-0x6f irq 1 on motherboard
pc0: type color
sio0 at 0x3f8-0x3ff irq 4 on isa
sio0: type <16450>
sio1 at 0x2f8-0x2ff irq 3 on isa
sio1: type <16450>
sio2 not found at 0x3e8
sio3 not found at 0x2e8
lpt0 not found at 0x3bc
lpa0 at 0x378-0x37f on isa
lpa1 not found at 0x278
fd0 at 0x3f0-0x3f7 irq 6 drq 2 on isa
fd0: unit 0 type 1.44MB 3.5in
wd0 at 0x1f0-0x1f7 irq 14 on isa
wd0: unit 0 type SMART CF
ahb0 not found
aha0 not found at 0x330
wt0 not found at 0x300
mcd0 not found at 0x300
ed0 at 0x280-0x28f irq 5 maddr 0xd8000 msize 8192 on isa
ed0: address 02:60:8c:7c:22:04, type 3c503 (16 bit)
ed1 not probed due to maddr conflict with ed0 at 0xd8000
ed1 not probed due to irq conflict with ed0 at 5
ie0 not found at 0x360
is0 not probed due to I/O address conflict with ed0 at 0x280
npx0 on motherboard
npx0: 387 Emulator
biomask 4040 ttymask 3a netmask 3a
```

FreeBSD

Congratulations

on 30 years of valuable contributions to the open source community!

LINUX MAGAZINE

OpenSource JOB HUB

FOSSLIFE

LINUX NEW MEDIA
The Pulse of Open Source

Meta

Is a proud platinum donor of FreeBSD and congratulates them on their 30 year anniversary.

HAPPY

30

ANNIVERSARY