Forking NetBSD

Converting a NetBSD 1.1 system to OpenBSD 2.0

Chris Pinnock

6th October 2022

Abstract

Recently we installed a virtual NetBSD/i386 1.0 system and upgraded it step by step to version 9.3 using only the source code and the compiler [6]. In [7] we examined the change of binary format from a.out to ELF between NetBSD/i386 1.4.3 and 1.5.4. Between releases 1.0 and 1.1, the OpenBSD project was formed, forking from NetBSD. In this paper we convert a NetBSD 1.1 system to OpenBSD 2.0 using just the source code and the compiler. We will touch on the history of the projects but the politics of the fork are well-documented elsewhere.

Disclaimer

Do not attempt to use the images or binaries in any production setting. The software will have security vulnerabilities. Also in this paper, we will use a password-less root account and this is not recommended in production.

Introduction

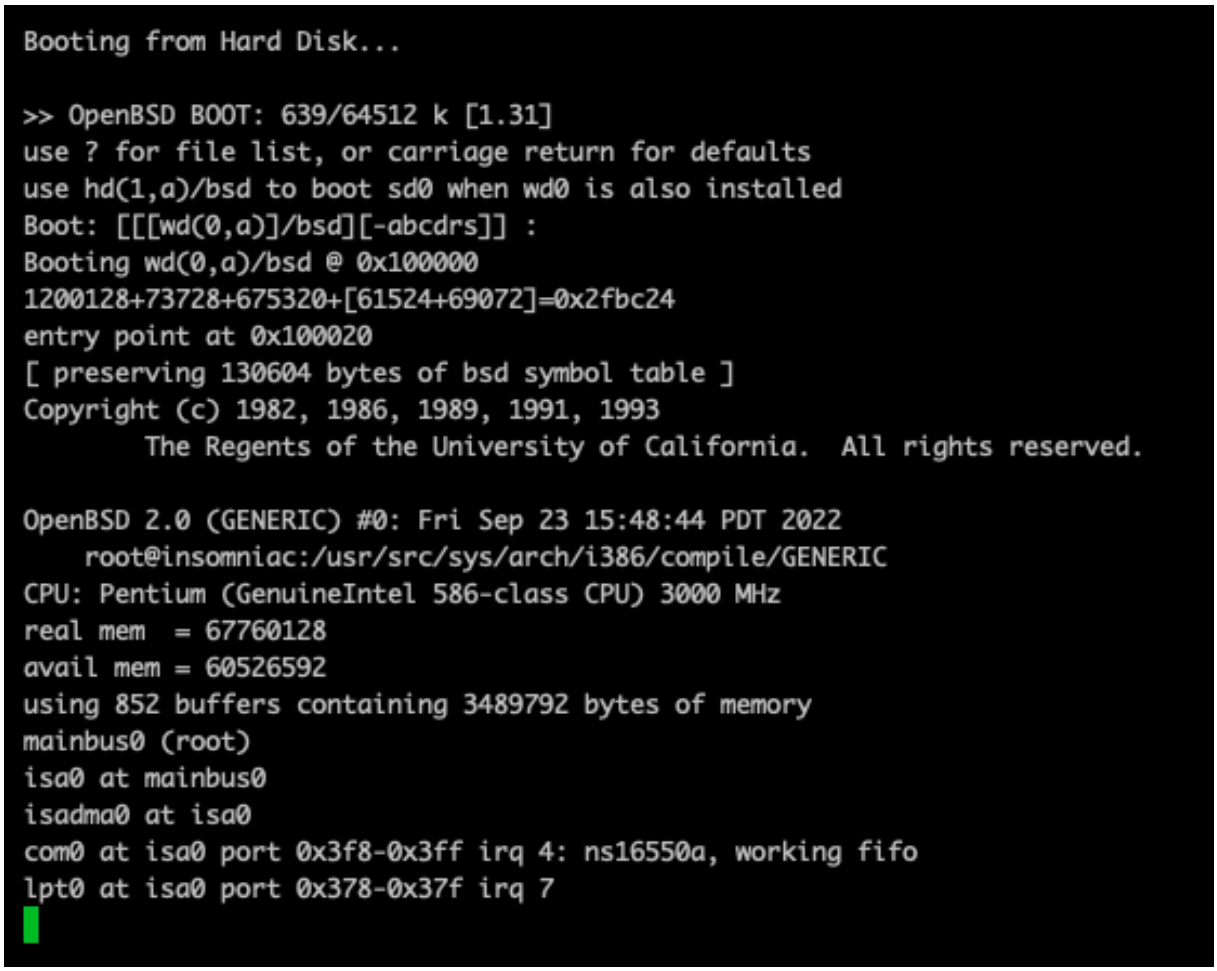

Figure 1: An OpenBSD 2.0 system converted from NetBSD 1.1

Open-source licenses give people the freedom to take source code, create new software and go in new directions. Such licenses allow code to be reused and improved upon, not necessarily by the originators of the original program. Some licenses such as the GNU General Public License (GPL) prevent closing the source code later on. The GPL stipulates that source code must be available if the binaries are distributed [1]. The BSD-style licenses allow a broader reuse including closing the source in a commercial situation [8].

Given the flexibility of the software model, it's no surprise that "forks" occur in the open-source world. There are often differences of opinion in how a project should progress and so the project splits into two groups, with the source trees diverging over time. There have been many software forks over the

years including[1]:

1. NetBSD from 386BSD (1993)

Within the 386BSD development community, there was concern about the speed of the development of the project. An unofficial community patch kit was developed collaboratively, but these patches did not make it back into the 386BSD source tree. The NetBSD project was formed to support a more open development model. It was based on 386BSD, the patch kit and some missing programs from the BSD Net/2 release [15]. Although the initial release was for the PC platform, it quickly supported other architectures.

2. FreeBSD from 386BSD (1993)

Another group of BSD developers waited a little longer than the NetBSD team to see if the 386BSD project would be more collaborative before setting their own project. The FreeBSD project originally targeted just the PC platform [10].

3. egcs from gcc (1997)

With the rationale of stability, the GNU Compiler Collection was developed under close control of the Free Software Foundation. There was difficulty getting patches and improvements accepted by the core development team. Several developers setup the Experimental GNU Compiler System (EGCS)[2] to collect various forks into one. The system supported many new architectures, a Fortran frontend, Pentium optimisations and other improvements. Eventually the projects merged again during the GCC 2.95 release [11].

4. DragonflyBSD from FreeBSD (2003)

DragonflyBSD was forked from FreeBSD 4.8 by Matthew Dillon [9]. The team developed an alternative symmetric multiprocessing and threading model to FreeBSD because they believed the model adopted in FreeBSD 5 would have performance problems.

5. XOrg from XFree86 (2004)

Initially the project forked due to a disagreement with the new XFree86 software license used for version 4.4 which includes a credit clause. Prior to 4.4, the project used the MIT license [17].

6. MariaDB from MySQL (2009)

Although MySQL was released under the GPL, it was also released under proprietary licenses and was owned by MySQL AB. This company was acquired by Sun, who were themselves acquired by Oracle. Due to a concern about the direction of future releases, the founder of MySQL forked the project and setup the MariaDB project [13], [14].

[1] For a more comprehensive list, please see Wikipedia [12].
[2] Also Enhanced GNU Compiler System.

7. OpenBSD from NetBSD (1995)

We are interested in OpenBSD in this paper. In 1995 there was a disagreement amongst the core group of NetBSD which lead to Theo de Raadt's departure from the project. We do not go into the detail here as it is well-documented on the web. It was not the first disagreement by members of the human race and it certainly won't be the last. Opinions differ, tensions rise, tempers fray and arguments ensue. After a period of time, Theo formed the OpenBSD project. The project would focus on computer security, but would continue with the goals of clean code and portability [16].

The initial release of OpenBSD was 1.2 and was based on NetBSD between 1.0 and 1.1. Shortly afterwards, OpenBSD 2.0 was released. The project has taken its own path and has not been binary compatible for some time. For several years after its release, it could boast of no remote security holes and to date only two such problems have been found.

Every 6 months there is a release and to date there have been 52 releases. As I write this paper, version 7.2 is being prepared for an Autumn release in 2022 [4].

The desire to produce portable secure code and also unencumbered BSD-licensed code has led to several bi-products including:

- OpenSSH, a BSD-licensed version of the Secure Shell program replacing the encumbered SSH. The implementation was a fork of *OSSH* which itself was a fork of the *SSH* project.
- OpenBGP, a BGP daemon enabling a Unix system to become a BGP router.
- OpenSMTPD, a modern mail transport agent.
- LibreSSL, a fork of OpenSSL used in the OpenBSD tree.
- Game of Trees, a recent addition which will provide a drop in replacement for *git*.

Having recently played with upgrading early NetBSD versions using just the compiler and source code (see [6] and [7]), I was curious to see if it was easy to "convert" an early NetBSD system to OpenBSD. Obviously the OpenBSD team would have done this incrementally but I was interested to see what was involved.

On examining the OpenBSD CVS tree [5], it seems the basis is NetBSD 1.1_ALPHA (a test release of NetBSD). Some early modifications were made for the OpenBSD 1.2 release but there is no tag in the CVS tree for it. The earliest tag I could find was for OpenBSD 2.0. As a result I chose to convert a NetBSD 1.1 system to OpenBSD 2.0. Actually it was not too difficult to do so. You can follow along with the method using QEMU.

Preparation

We use the QEMU emulator to provide an i386 environment and we have prepared resources that will work with QEMU straight away. However there is no reason that you cannot use VirtualBox or BOCHS -

you will just need to get a NetBSD 1.1 installation into the right format.

To install QEMU on your machine, do one of the following:

```
# NetBSD using pkgin
pkgin install qemu

# OpenBSD
pkg_add qemu

# FreeBSD
pkg install qemu

# Debian/Ubuntu Linux
apt install qemu

# macOS (with Homebrew, https://brew.sh)
brew install qemu
```

Make sure that you have 2GB of disc space free. It is advisable to work in a dedicated directory.

Throughout the text, % usually represents a command-line prompt on the host machine and # usually represents a root command-line prompt on the emulated machine. We use *csh* throughout because it was the default shell for root on these operating systems. However the commands should work fine if you decide to use *sh* or *ksh* instead.

We've assumed paths relative to */usr/src* throughout the text, so "*cd gnu/usr.bin*" means "*cd /usr/src/gnu/usr.bin*". If you use *csh* the *cdpath* variable is set so that the short version works by default.

Within the OpenBSD Source code at the time, the directories contained the following code:

Directory	**Purpose**
bin	Source for statically linked binaries required to bring the system up
distrib	Architecture specific tools to build the OpenBSD release distribution
etc	Default configuration files and tools to configure a system
games	Terminal games including Adventure, Hangman, Tetris and Trek
gnu	GNU GPL and other non-BSD licensed software including the *gas* (Assembler), GNU *awk*, *diff*, *gcc* (Compiler), *gdb* (Debugger), *grep*, *perl*, *texinfo* and supporting libraries
include	The C preprocessor header files containing definitions of functions and system calls
kerberosIV	Programs and libraries for the Kerberos authentication system

Directory	Purpose
lib	The library source code include the standard C library and math library
libexec	Source code for binaries that are not directly run by users (such as Internet daemons)
lkm	Loadable Kernel Modules including the Xfree86 aperture driver
regress	Regression tests for testing changes to software
sbin	As *bin* but typically requiring super-user (root) privileges
share	Files that are shareable between architectures and systems, such as make files, time zone information and manual pages
sys	The source code for the kernel
usr.bin	Source for general system-wide binaries, usually dynamically linked
usr.sbin	As *usr.bin* but typically requiring super-user privileges

Migrating from NetBSD 1.1 to OpenBSD 2.0

1. Obtain the sources for OpenBSD 2.0.

You can check these out from OpenBSD's CVS repository using tag OPENBSD_2_0 or alternatively download my prepared copy [S1]. You will need to copy these onto the VM and we will do this via FTP. If you are not feeling adventurous, you can download a NetBSD 1.1 VM with the sources already in */usr/src* [D2] and skip to step 5.

2. Download and setup the NetBSD 1.1 VM. [D1]

```
% wget http://downloads.chrispinnock.com/netbsdhist/3-vms/amnesiac-netbsd-
    i386-1.1.img.bz2
% bunzip2 amnesiac-netbsd-i386-1.1.img.bz2
% cp amnesiac-netbsd-i386-1.1.img openbsd-i386.img
% chmod u+w openbsd-i386.img
```

3. Boot the VM.

```
% qemu-system-i386 -hda openbsd-i386.img -net user -net nic,model=pcnet
```

Log in as root (no password). Rename the machine *insomniac*.

```
# echo "insomniac" > /etc/myname
```

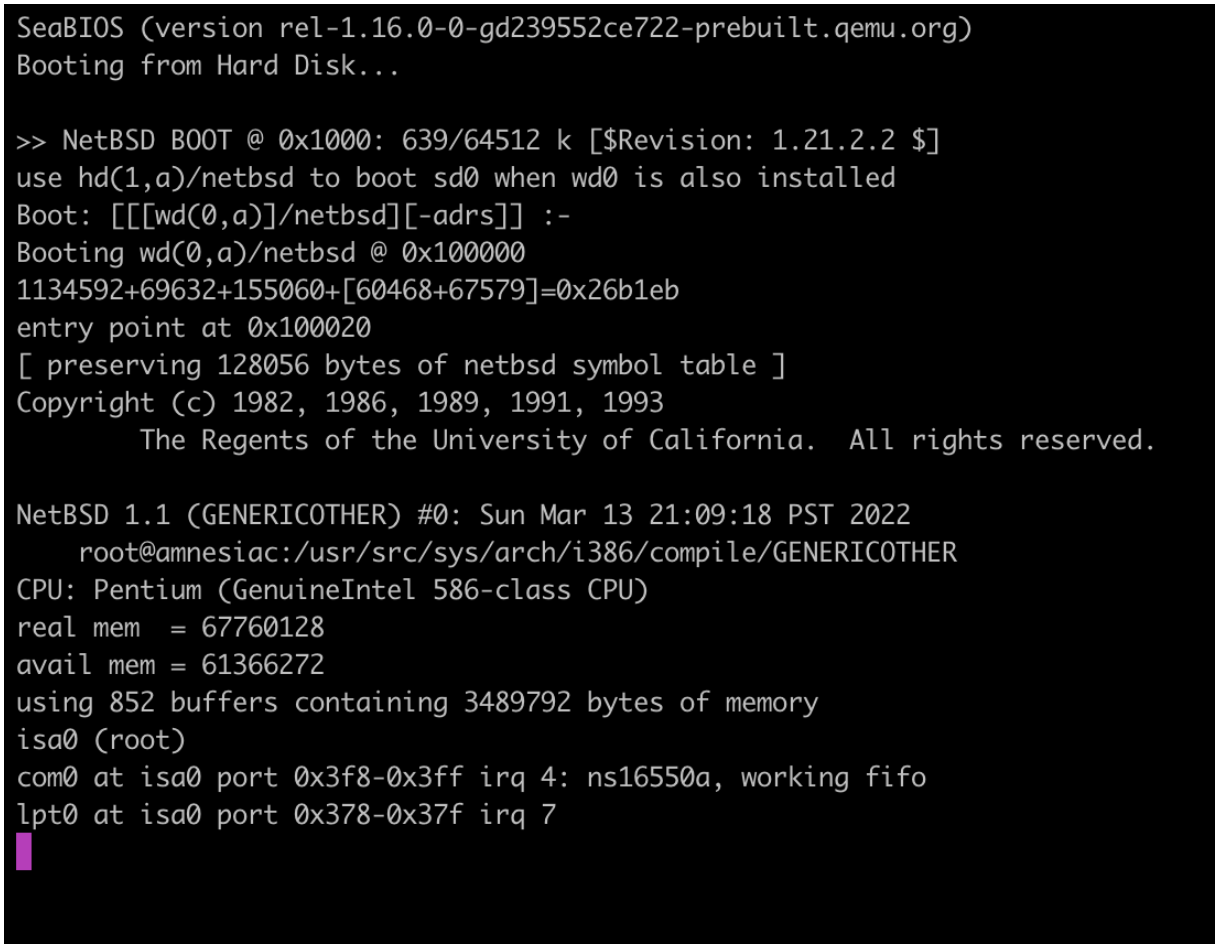

Figure 2: Booting NetBSD/i386 1.1

4. Setup the network on the VM. Get the sources (e.g. FTP) and decompress them. I used an FTP server on my LAN. For QEMU the following works (but replace 192.168.178.44 with the IP address of your FTP server):

```
# ifconfig le0 10.0.2.5 netmask 255.255.255.0
# route add default 10.0.2.2
add net default: gateway 10.0.2.2
# cd /usr/
# ftp 192.168.178.44
Connected to 192.168.178.44.
220 files.terry3.de FTP server ready.
Name (192.168.178.44:root): ftp
331 Password required for ftp.
Password:
230 User ftp logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> pass
Passive mode on.
ftp> prompt
Interactive mode off.
ftp> mget openbsd-2.0.tgz
local: openbsd-2.0.tgz remote: openbsd-2.0.tgz
....
ftp> quit
221 Goodbye.
```

5. Remove the source code and distribution sets.

```
# rm -rf /usr/src /usr/obj /usr/distrib/*
# tar zxf openbsd-2.0.tgz
```

6. Build the OpenBSD kernel.

The usual way to upgrade the kernel between releases is to rebuild the *config* tool and then build the kernel with *make*. However we will need to make two adjustments for this to work.

Firstly the Pentium (586) processor support will not compile because it uses new instructions that the current system's assembler does not understand. Also the *ahc* drivers for the PCI and EISA buses will not built with our current toolchain. We will build a kernel without these for now and come back to them.

Additionally, there is a clause in *sys/vm/vm_extern.h* that will not work with the compiler. We will make a small adjustment to this file.

Rebuild config:

```
# cd usr.sbin/config
# make && make install && make clean
```

Preserve *sys/vm/vm_extern.h* and remove lines 86-88 and 90. Here we use *ed* but you can use *vi* if you like [2].

```
# cd sys/vm
# cp -p vm_extern.h vm_extern.h.orig
# ed vm_extern.h
90d
86,88d
w
q
```

Make a copy of the GENERIC kernel configuration and delete lines 13, 99 and 100 containing the *ahc* driver definitions.

```
# cd sys/arch/i386/conf
# cp GENERIC GENERIC1
# ed GENERIC1
100d
99d
13d
w
q
# config GENERIC1
Don't forget to run "make depend"
# cd ../compile/GENERIC1
# make depend && make
```

Preserve the old kernel and install the new kernel. We will call it *netbsd* because the boot loader will automatically use this filename, but OpenBSD calls it *bsd*. We have not installed the OpenBSD boot loader yet, so we will continue to use the old filename.[3]

```
# cp /netbsd /netbsd.11
# cp bsd /bsd
# cp bsd /netbsd
```

Restore the change to *sys/vm/vm_extern.h* so that we don't forget later.

```
# cd /usr/src
# cp -p sys/vm/vm_extern.h.orig sys/vm/vm_extern.h
```

Now reboot the system.

```
# shutdown -r now
```

7. NetBSD User-land with an OpenBSD kernel

The system should have successfully booted with the OpenBSD kernel. Now it is time to rebuild user-land.

[3] Of course, you could interrupt the boot loader and supply a different filename if you prefer.

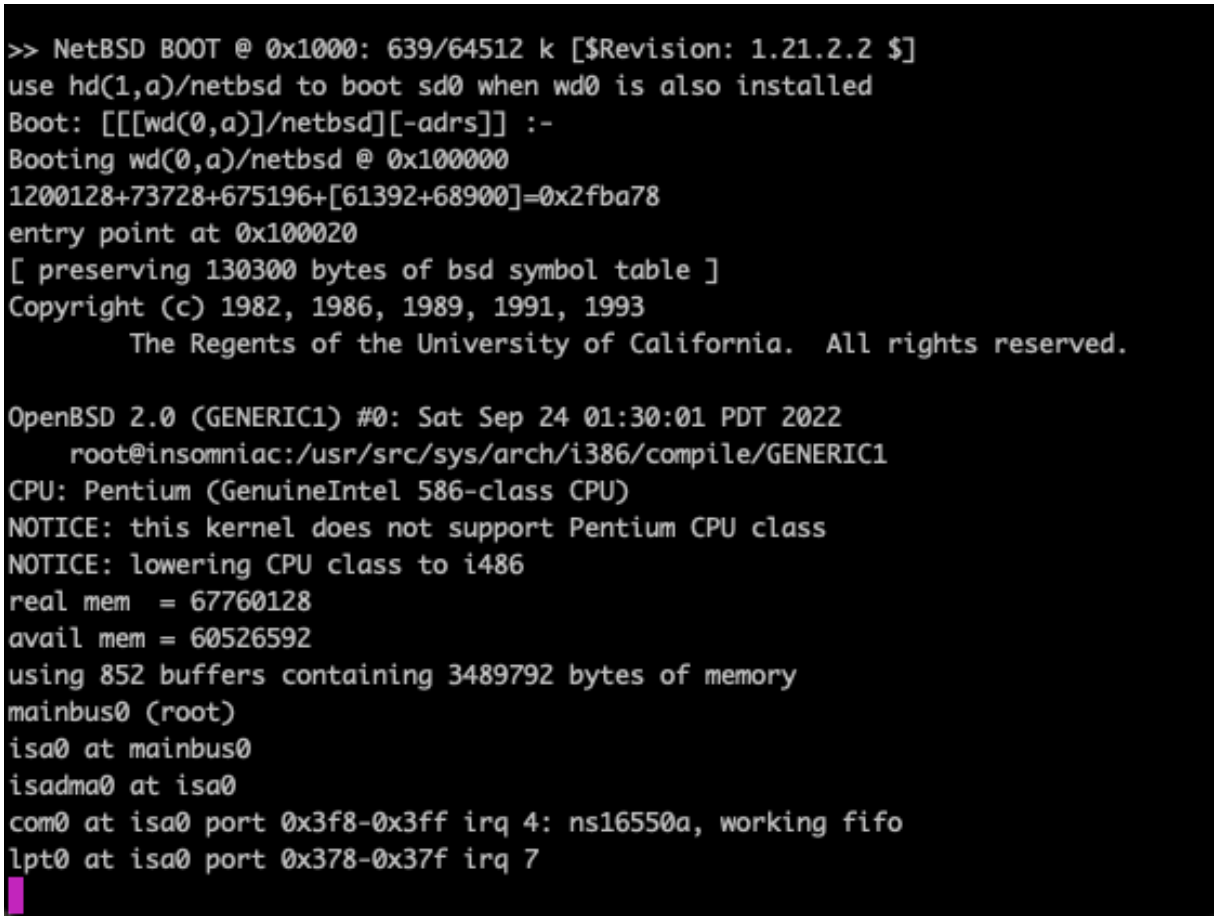

Figure 3: Booting OpenBSD with the NetBSD boot loader and user-land

8. Make, supporting files and directories

First we rebuild *make* and install it, then install the supporting make files so that the OpenBSD source files are compatible. Then we install the target directories on the file system.

```
cd usr.bin/make
make && make install && make clean
cd share/mk
make install
cd etc
make DESTDIR=/ distrib-dirs
```

9. Make the object directories. These directories hold the object files during the build and are useful to segregate the output of the build from the source code.

```
# cd /usr/src
# mkdir -p /usr/obj
# make obj
```

10. Rebuild other prerequisites: *yacc*, *lex*, *tsort*, *lndir* and *sh* [4]. These tools are used during the build later on.

tsort and *lndir* are needed when building the GNU tools. A newer *sh* is needed for shell substitutions in the new Makefiles.

```
cd usr.bin/yacc
make && make install && make clean
cd usr.bin/lex
make && make install && make clean
cd usr.bin/tsort
make && make install && make clean
cd usr.bin/lndir
make && make install && make clean
cd bin/sh
make && make install && make clean
```

11. Next we update the include files and libraries, starting with the compiler runtime support *csu*. This code contains initialisation and exit routines used by all programs. Then we rebuild the C library and then all libraries.

```
cd lib/csu
make && make install
cd include
make includes
cd lib/libc
make && make install
cd /usr/src/lib
```

[4] I rebuild *yacc* and *lex* out of habit. You may be able to get by without rebuilding these.

```
make && make install
```

12. OpenBSD 2.0 included *info* pages with the GNU software. We need to build *texinfo* in advance of the rest of the build. We build using the SUBDIR trick below because the OpenBSD make files do extra work than normal to run configure and make in the source directory.

```
# cd gnu/usr.bin
# make SUBDIR=texinfo
# make SUBDIR=texinfo install
# cd usr.bin/info_mkdb
# make && make install
```

13. Rebuild the assembler and linker:

```
# cd gnu/usr.bin/gas
# make && make install
# cd gnu/usr.bin/ld
# make && make install
```

14. The hardest bit of this exercise is persuading the compiler to build. In [7] we saw a similar problem upgrading from NetBSD 1.1 to 1.2 (cf. [3]). We will partially build the compiler until it fails, install one piece of it and then rebuild it.

```
# cd gnu/usr.bin
# make SUBDIR=gcc
...breaks...
# cp /usr/obj/gnu/usr.bin/gcc/cc1 /usr/libexec
# make SUBDIR=gcc
# make SUBDIR=gcc install
```

15. Now we rebuild the rest of user-land, except for three components: *libg++*, *groff* and *xlint*. *Groff* doesn't work because our C++ installation is not finished.

Comment out `SUBDIR+= libg++` in *gnu/lib/Makefile*, remove *groff* from the `SUBDIR` declaration in *gnu/usr.bin/Makefile* and *xlint* from the `SUBDIR` declaration in *usr.bin/Makefile*.

Then rebuild everything else.

```
cd /usr/src
make build
```

16. Reboot the system with `shutdown -r now`

17. Tidy up *libg++*, *xlint* and *groff*.

The first library is for C++ and once it is made, the C++ compiler will work correctly again. This will allow us to build *groff*.

Reverse the changes made in 14 to *gnu/lib/Makefile*, *gnu/usr.bin/Makefile* and *usr.bin/Makefile*. Then:

```
cd gnu/lib
make && make install
cd gnu/usr.bin
make SUBDIR=groff && make SUBDIR=groff install
cd usr.bin/xlint
make && make install
```

18. Reboot and rebuild everything from scratch. This is probably unnecessary, but it does ensure that the system is built correctly and everything is in the right place.

```
cd /usr/src
make build
```

19. Install the OpenBSD boot loader.

The command to install the boot block is *disklabel*, which is the tool for creating BSD partitions. The boot loader can be found in */usr/mdec*.

```
cd /usr/mdec
disklabel -v -B -b wdboot wd0
```

20. The GENERIC kernel will now build correctly. Build it and install it at */bsd* (and if you want save a copy at */bsd.20*).

21. Reboot and check that you have a working OpenBSD system.

At this point we are almost complete. What remains is to tidy up */etc*. We leave that as an exercise for the reader. A copy of my VM upgraded to OpenBSD 2.0 can be found on my website [D3].

Exercises

1. Tidy up /etc and /dev

1. From */usr/src/etc*, update */etc* with OpenBSD's configuration files.
2. Hint: Update */etc/ttys* before making device nodes
3. Hint: Update *MAKEDEV* from *etc/etc.i386* in */dev* and run it `./MAKEDEV all` in the */dev* directory.

2. Process Improvement

1. Are there any ways that you can improve the process above?
2. Is it necessary to build *yacc* and *lex*?
3. Are there any files in */usr/src* that would help you to determine whether a program should have been installed or not?

4. Try the build again but without the order above by building the kernel, rebooting and then running *make build*. When something fails, look at the errors and try to determine the root cases.

For example, if one does not rebuild */bin/sh* before the full build, some of the manual pages will not install correctly. You can examine *bsd.prog.mk* to figure out why this might be caused by the shell.

3. Upgrade the system to 2.1

Hints:

1. Clean out */usr/obj*.
2. Get the OpenBSD 2.1 source code from CVS or [S2]. Replace */usr/src*.
3. Build and install *make*.
4. Install *share/mk*.
5. Make the object directories with *make obj*.
6. Build and install *mktemp* and *config*.
7. Build *GENERIC* without the *aic* and *ahc* drivers.
8. Install and reboot.
9. Build and install *compile_et* and *mk_cmds*.
10. Install the include files.
11. Build and install *lib*.
12. In this order, build and install in *gnu/usr.bin*: *gas*, *ld*, *texinfo*, then *gcc* (use the `SUBDIR` trick for *texinfo* and *gcc*).
13. *make build*.
14. Update to the new *MAKEDEV* and run it. Also update */etc/ttys*.
15. Before you reboot change *ufs* to *ffs* in */etc/fstab*.
16. Build *GENERIC* with the missing drivers.

Acknowledgements

This is the third paper I've written on early BSD distributions. In addition to the people that helped me with [6] and [7], I would like to thank Anil Madhavapeddy and Richard Mortier at the University of Cambridge for their encouragement. Roland Dowdeswell and Arrigo Triulzi also made some helpful suggestions.

Bibliography

References

[1] A Quick Guide to the GPLv3, Brett Smith, GNU.Org.

[2] *Ed Mastery: **The Standard** Unix Text Editor*, Michael W. Lucas, Tilted Windmill Press, 2018.

[3] Upgrading to gcc 2.7.2, NetBSD Current Users mailing list, December 1995.

[4] The OpenBSD Project, The OpenBSD Project.

[5] OpenBSD CVS Source code tree, The OpenBSD Project.

[6] Transitioning from a.out to ELF on NetBSD, Chris Pinnock, Preprint August 2022

[7] NetBSD/i386 from 1.0 to present, Chris Pinnock, Preprint September 2022

[8] BSD licenses, Wikipedia.

[9] DragonFly BSD, Wikipedia.

[10] FreeBSD, Wikipedia.

[11] GNU Compiler Collection, EGCS, Wikipedia.

[12] List of software forks, Wikipedia.

[13] MariaDB, Wikipedia.

[14] MySQL, Wikipedia.

[15] NetBSD, Wikipedia.

[16] OpenBSD, Wikipedia.

[17] X.Org, Wikipedia.

Media

[D1] NetBSD 1.1 VM

[D2] NetBSD 1.1 VM with OpenBSD 2.0 sources

[D3] OpenBSD 2.0 VM

[S1] OpenBSD 2.0 sources

[S2] OpenBSD 2.1 sources