# NetBSD/i386 from 1.0 to present

Chris Pinnock

23rd September 2022

# Contents

## 1 Abstract

> *Dedicated to William Jolitz, 22/7/1957 - 2/3/2022.*

The NetBSD operating system was born in 1993 and is based on the 386BSD operating system. Originally it only supported i386 PC hardware, but it has grown into a portable system supporting most consumer computing equipment. During the COVID lockdown, the author took a nostalgic trip down memory lane and made virtual machine images of older NetBSD versions. In [12] the author describes the upgrade between NetBSD/i386 1.4.3 and 1.5.4 where the binary object format changed from a.out to ELF. In this paper we tackle all versions between 1.0 to 1.4.3 and then 1.5.4 to present day upgrading using the source code and compiler.

## 2 Disclaimer

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

# 3 Introduction

In this document we recreate the upgrades from NetBSD 1.0 to present using the source code and the compiler. It has always been possible to rebuild the NetBSD operating system [8] using just the tools that are in its source tree. One early exception was the lack of *cvs*, but one could obtain the source code by other means.

We start by setting up a virtual machine called *amnesiac* with NetBSD/i386 1.0 and upgrade it to the latest version step by step, using only the source code and compiler. We shall mostly use QEMU [2] throughout the paper, but to setup the original VM it was easier to use VirtualBox [3] due to its floppy support. We will work on an Apple Mac (Intel chipset) but the paper can be followed using any modern environment capable of running VirtualBox, QEMU or Bochs.

As we will discuss later, NetBSD 0.8 and 0.9 are not publicly available, so we start with version 1.0 which is the earliest version available in the NetBSD Archive [8].

At the end of the process, our system has copieds of the different GENERIC kernels from 1.0 to 9.3. The present version is over 30 times the size of the 1.0 kernel. If we were doing this with production hardware, it's very likely we would have had to change components or upgrade it.

```
amnesiac# ls -l /netbsd*
-rwxr-xr-x  1 root  wheel  19089180 Mar 26 02:47 /netbsd
-rwxr-xr-x  1 root  wheel    659600 Mar 13 22:09 /netbsd.10
-rwxr-xr-x  1 root  wheel   1332275 Mar 13 22:09 /netbsd.11
-rwxr-xr-x  1 root  wheel   1477949 Mar 13 16:13 /netbsd.12
-rwxr-xr-x  1 root  wheel   1530049 Mar 13 17:47 /netbsd.121
-rwxr-xr-x  1 root  wheel   2211878 Mar 14 02:23 /netbsd.133
-rwxr-xr-x  1 root  wheel   3407541 Mar 14 05:12 /netbsd.143
-rwxr-xr-x  1 root  wheel   4941157 Mar 14 05:50 /netbsd.154.aout
-rwxr-xr-x  1 root  wheel   5048054 Mar 14 09:20 /netbsd.154.elf
-rwxr-xr-x  1 root  wheel   6182687 Mar 14 11:53 /netbsd.162
-rwxr-xr-x  1 root  wheel   7447216 Mar 14 14:35 /netbsd.203
-rwxr-xr-x  1 root  wheel   7476202 Mar 15 01:12 /netbsd.21
-rwxr-xr-x  1 root  wheel   8577892 Mar 15 11:50 /netbsd.31
-rwxr-xr-x  1 root  wheel  10343742 Mar 15 23:47 /netbsd.4
-rwxr-xr-x  1 root  wheel  12002769 Mar 16 05:31 /netbsd.52
-rwxr-xr-x  1 root  wheel  13116694 Mar 16 14:53 /netbsd.61
-rwxr-xr-x  1 root  wheel  17143555 Mar 17 04:50 /netbsd.72
-rwxr-xr-x  1 root  wheel  19695304 Mar 20 09:12 /netbsd.82
-rwxr-xr-x  1 root  wheel  20714756 Sep  9 06:32 /netbsd.93
```

If you follow the upgrade process, please be prepared to dedicate some time to it. It's possible to upgrade from 1.0 to 1.3.3 in an afternoon, but between later releases, more time is required. You do not need to be at the keyboard during each build, but you will need to work on a machine that can run the emulator uninterrupted.

# 4 NetBSD Background

We will only scratch the surface of the history here. The interested reader can consult [1], [5] and [7].

NetBSD has its origins in UNIX, which was born at Bell Laboratories in 1969. The sixth edition of UNIX was readily available outside of Bell along with its source code. The availability of the source encouraged others to use the system for research and contribute to it.

The Computer Systems Research Group (CSRG) at University of California, Berkeley (UCB) was one of many organisations that contributed with the blessing of Bell Laboratories and its parent AT&T. In order to use UNIX and its variants, one had to purchase a source code license from AT&T, but other than this, there was relative freedom in what could be done with the system. UCB released their version of UNIX under the name Berkeley Software Distribution (BSD) and anyone could use it provided they had an AT&T source code license.

At UCB, work was undertaken by William Jolitz to port BSD to the PC platform based on the Intel 80386 processor. The work was based on the 4.3BSD Net/2 release. Together with his wife Lynne, he released the port in March 1992 as 386BSD [2] and wrote an accompanying series of articles about the port in Dr Dobbs [1]. A group of enthusiasts started to contribute patches. There were differences of opinion in approach with the 386BSD project founders and so not all of the patches made it back into 386BSD, but they were released as a patch kit.

The NetBSD project was founded in 1993 by Chris Demetriou, Theo de Raadt, Adam Glass and Charles Hannum. The founders wanted to encourage open collaboration between the developers and continue to produce a reusable code base in the spirit of the CSRG. The first release 0.8 was derived from 386BSD, the patch kit and some programs from Net/2 that were missing.

NetBSD 0.8 was available from an FTP site and this was the only way to obtain it. The *Net* in NetBSD reflects the use of the Internet and the cooperation over the network to produce the software. The software had to be downloaded, written onto floppy discs and then loaded onto the machine. The installation process used a boot disc containing the kernel and a second disc containing the root filesystem. By today's standards, the installation is involved and error prone [1]. A few months later, the team released NetBSD 0.9 included loadable kernel modules and numerous bug fixes and enhancements.

Around the same time that NetBSD was born, another group of developers who waited a bit longer for the patches to be accepted into 386BSD, formed the FreeBSD project [6] with the primary goal to produce a BSD-based operating system for the PC platform.

BSD influenced the UNIX world greatly, leading to commercial UNIX versions such as BSDI and SunOS, and of course the community projects we see today such as DragonflyBSD, FreeBSD, NetBSD and OpenBSD. The TCP/IP Internet communication stack we use today is largely based on work at UCB.

[1] In this paper, we have recreated the experience for NetBSD 1.0 (the installation is similar).

We should explain why we have decided to start with version 1.0 and not 0.8. AT&T's licensing subsidiary UNIX System Laboratory (USL) decided to increase the cost of the UNIX license and consequently end users of BSD would incur the higher cost. One of the aims of the Net/2 release was to remove encumbered code from BSD. An effort by Keith Bostic and others, led to the rewrite of many utilities so that they could be distributed without the AT&T license. By the end of the activity, the license only applied to six source files. In fact, when Jolitz did his port he rewrote these six files essentially making the Net/2 release self-sufficient for PC hardware. BSDI also rewrote the files when they started their operating system work.

BSDI begun to sell their UNIX system for $995 in January 1992. Shortly after, BSDI received a cease-and-desist letter from USL, requesting that BSDI cease promoting the product as UNIX. USL alleged that the product contained prioprietary code. As the resulting case developed, BSDI's claim was that they were simply distributing the BSD system with six additional files. The judge agreed with their position and told USL to reorganise its case.

USL was purchased by Novell who settled the case in the summer of 1993. Changes were made to source code and copyright notices were added where necessary. This let to the 4.4BSD-Lite release which was unencumbered by legality. Later CSRG released 4.4BSD-Lite release 2 shortly before disbanding [4].

The knock-on effect of the lawsuit for the NetBSD project as a derivative of Net/2 was that the project had to rebase its code on 4.4BSD/Lite and withdraw 0.8 and 0.9 in case they contained code under dispute. As a result it is very difficult to find the 0.8 and 0.9 releases on the Internet. NetBSD 1.0 was released rebased on 4.4BSD/Lite.

The NetBSD project expanded and gained traction, aligning on the following goals:

- Good design, stability, performance
- No encumbering licenses
- Portable code
- Interoperability with other systems
- Comply to Open Standards as much as possible

The vast number of different architectures that NetBSD supports has made it necessary to write clean and portable machine independent code.

The lawsuit will have influenced the approach on encumbering licenses for software. Omitting some components such as *gcc), NetBSD has always been released using a BSD-style license [3]. This meant that the code could be reused in almost all situations, including a commercial entity wanting to base a closed source product on it. This is a different approach to the GNU Public License, where source code must be distributed with binaries. The debate about which style of license results in the most reusable set of code is beyond the scope of this paper.

Later the project formed a core group to oversee development and the NetBSD Foundation to serve as the legal entity managing the project.

Since version 0.8, there have been many releases. Here we summarise the key features from each major version (see [7] and [9]).

- 0.8, April 1993
  - PC hardware (i386) only release
  - 386BSD 0.1 + patch kit
  - other programmes from Net/2
  - installer and general improvements
- 0.9, August 1993
  - Loadable Kernel Modules, allowing new drivers to be added during runtime
- 1.0, November 1994
  - Multi-platform release for HP300, Amiga, Mac (68k), Sun (sun4c) and PC532 platforms
  - Net/2 derived code replaced by 4.4BSD/Lite code
  - Shared libraries
  - Kerberos 5 authentication system added
- 1.1, November 1995
  - Support for Alpha, Atari and mvme68k platforms
  - New generic audio subsystem
  - Linux and SunOS emulation, for example allowing PC Linux binaries to run on NetBSD/i386
- 1.2, October 1996
  - Network File System v3
  - ARM and Sharp x68k platform support
- 1.3, March 1998
  - New i386 installer vastly simplifying the installation
  - X Windows (XFree86) graphical user interface added to the base system
  - ISA Plug'n'Play, PCMCIA, ATAPI and APM support
  - Linux ext2 and FAT32 filesystem support
  - The NetBSD packages collection, making it easier to install 3rd party software
- 1.4, May 1999
  - UVM, a complete rewrite of the virtual memory system improving performance

  - RAIDframe, software RAID system for managing resilience on hard discs
  - IP Filter, firewall and packet filter
  - USB Support
  - Workstation Independent driver framework *wscons*

- 1.5, December 2000

  - IPv6 and IPSec support
  - OpenSSL and OpenSSH for secure communications with other systems
  - The rc.d system for starting and stopping services
  - Migration to the ELF binary format (see [12])
  - Kernel tracing tool *ktrace*
  - Filesystem soft dependencies for better crash handling
  - Windows NT filesystem support

- 1.6, September 2002

  - Unified Buffer Cache, unifying the filesystem and virtual memory caches and improving system performance
  - The *build.sh* build system is introduced allowing reliable builds of NetBSD (and eventually cross-builds from other systems)
  - Ten new platforms
  - Multibyte locales to support non-Latin character sets

- 2.0, December 2004

  - POSIX threads and multiprocessor support on i386
  - AMD64 support
  - Unix Filesystem v2 and Samba filesystem support

- 3.0, December 2005

  - Xen hypervisor support
  - Filesystems over 2TB
  - Pluggable Authentication Module support
  - PF, packet filter as an alternative to IPFilter

- 4.0, December 2007

  - Bluetooth stack
  - Verified executables, preventing unknown binaries from running in secure environments

- 5.0, April 2009

  - Improvements in Loadable Kernel Modules and threading

  - Replacement of Xfree86 with the X.Org X windows system
  - Import of the Clang compiler with the aim of replacing GCC

- 6.0, October 2012

  - Logical Volume Manager, abstracting physical discs into logical volumes for better management
  - Better flash device support
  - MPLS support in the network stack
  - NetBSD's Packet Filter introduced
  - The 2038 UNIX time epoch fixed by changing *time_t* to 64-bit

- 7.0, October 2015

  - In kernel implementation of the Lua scripting language
  - Dynamic internet protection with the blacklisting daemon
  - Multiprocessor ARM support
  - Raspberry Pi 2 support and other ARM boards

- 8.0, July 2018

  - USB 3 support
  - Hardened memory layout by making only essential pages writable
  - Reproducible builds, where *build.sh* outputs identical binaries on different machines
  - UEFI boot support
  - NVMe support for newer flash drives
  - NVidia GPU support
  - Raspberry Pi 3 support

- 9.0, February 2020

  - 64-bit ARM architecture support
  - Hardware acceleration for emulators (e.g. Qemu) using NetBSD's Virtual Machine Monitor.
  - Improvements to the firewall, ZFS and drivers

We will now install v1.0 on a virtual i386 environment, then step through the releases of NetBSD using only the source code and the compiler. After v1.6, the process becomes simpler because the *build.sh* system has been introduced.

# 5 Setting up and getting ready

## 5.1 Conventions

Throughout, where you see % it is usually a command prompt on the system running the emulator and # is usually a root prompt on the emulated NetBSD system. Historically, NetBSD used the *csh* shell for root and we have assumed the same here.

You can use *sh* or *ksh* if you like, but you will need to make adjustments. For example, where you see the use of *setenv*, for example:

```
% setenv VARIABLE thing
```

you will need to substitute:

```
$ VARIABLE=thing
$ export VARIABLE
```

We do not cover the source code structure here, but the interested reader can refer to the History section of [12].

## 5.2 VirtualBox

We will use Oracle's VirtualBox[2] because it supports the i386 architecture and has working floppy drive support, specifically for 1.2M floppies. NetBSD 1.0 was released in November 1994 when hard discs were small and 5.25" floppy discs were still prevalent.

I would normally use QEMU[3] for a project like this but I could not get the floppy disc support to work. We will start with VirtualBox and swap to QEMU later on. There is no reason that you cannot follow the process using VirtualBox throughout. Also it should be possible to use Bochs[4].

To install VirtualBox, download the relevant package and follow the installation instructions at the web site. The version used in this document was 6.1.32 r149290.

## 5.3 Obtaining the media

The NetBSD 1.0 distribution can be found at the NetBSD archive. To install the system on VirtualBox, you will need:

[2]VirtualBox: https://www.virtualbox.org
[3]QEMU https://www.qemu.org
[4]Bochs https://bochs.sourceforge.io

- A kernel boot floppy kcaha-10.fs
- The installation filesystem floppy disc image inst10.fs
- The distribution sets loaded onto floppy images.
- Patience.

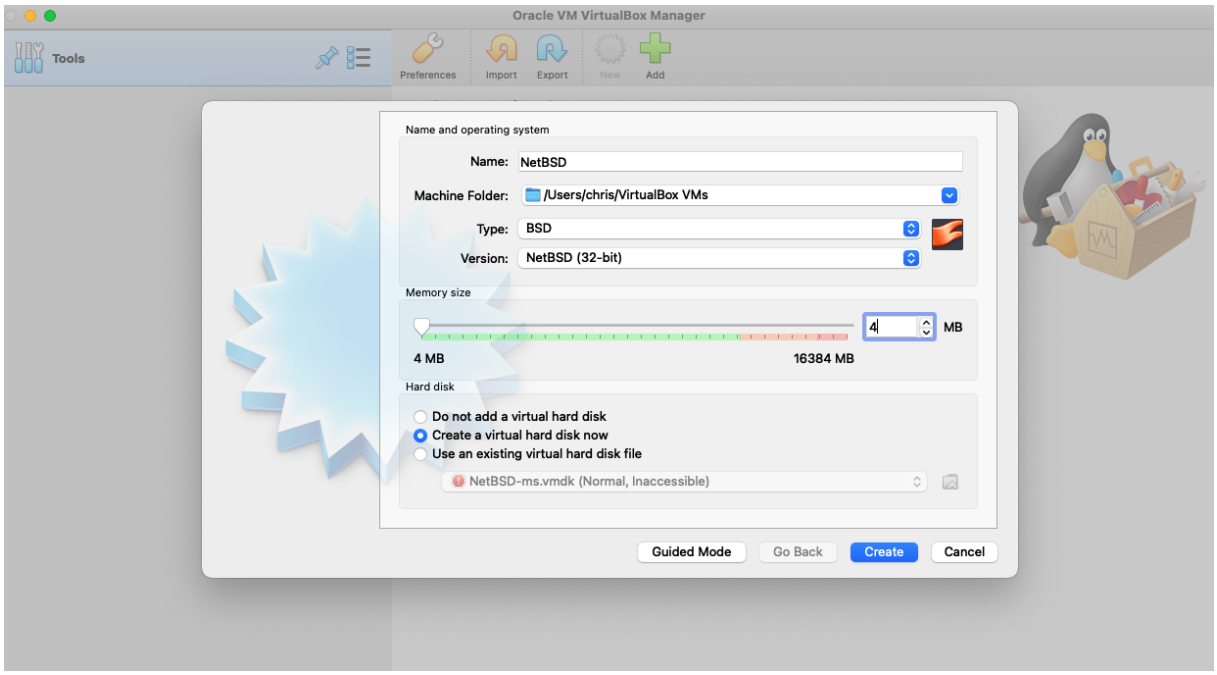
To make this easy, I have prepared a tarball of everything needed [D1] including the kcaha-10, inst-10 floppies and the complete binary installation media on floppy images numbered from 0 to 17.

Additionally I've prepared the source sets for 1.1 [D2] (28 disc images) and 1.2 [D3] (36 disc images) as floppy images so that they can be loaded on the VM.

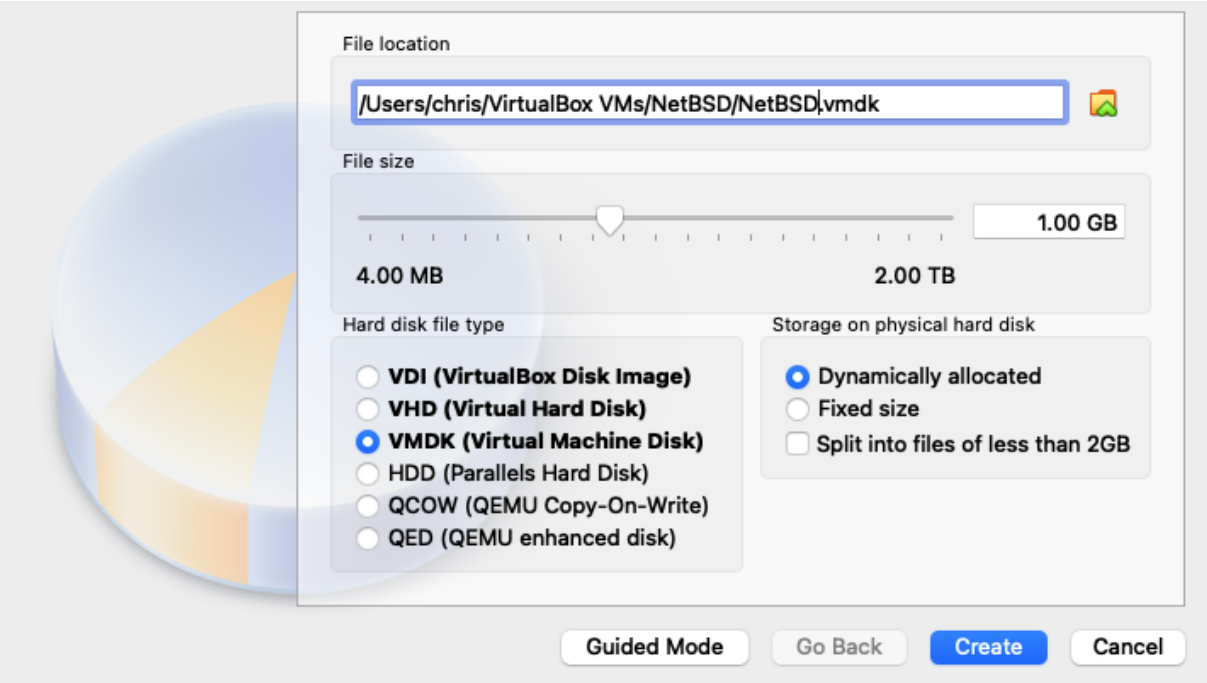
## 5.4 Installing NetBSD 1.0

We follow the original installation procedure [10] to the letter. The only difference is that we are using a virtual environment and we will use the VirtualBox floppy disc emulation to insert and remove floppy disc images.

1. Start VirtualBox, click New and create a new 32-bit NetBSD VM called "NetBSD". Four megabytes of RAM is sufficient for what we are doing here.




2. Create a VMDK-based hard disc image of 1GB:

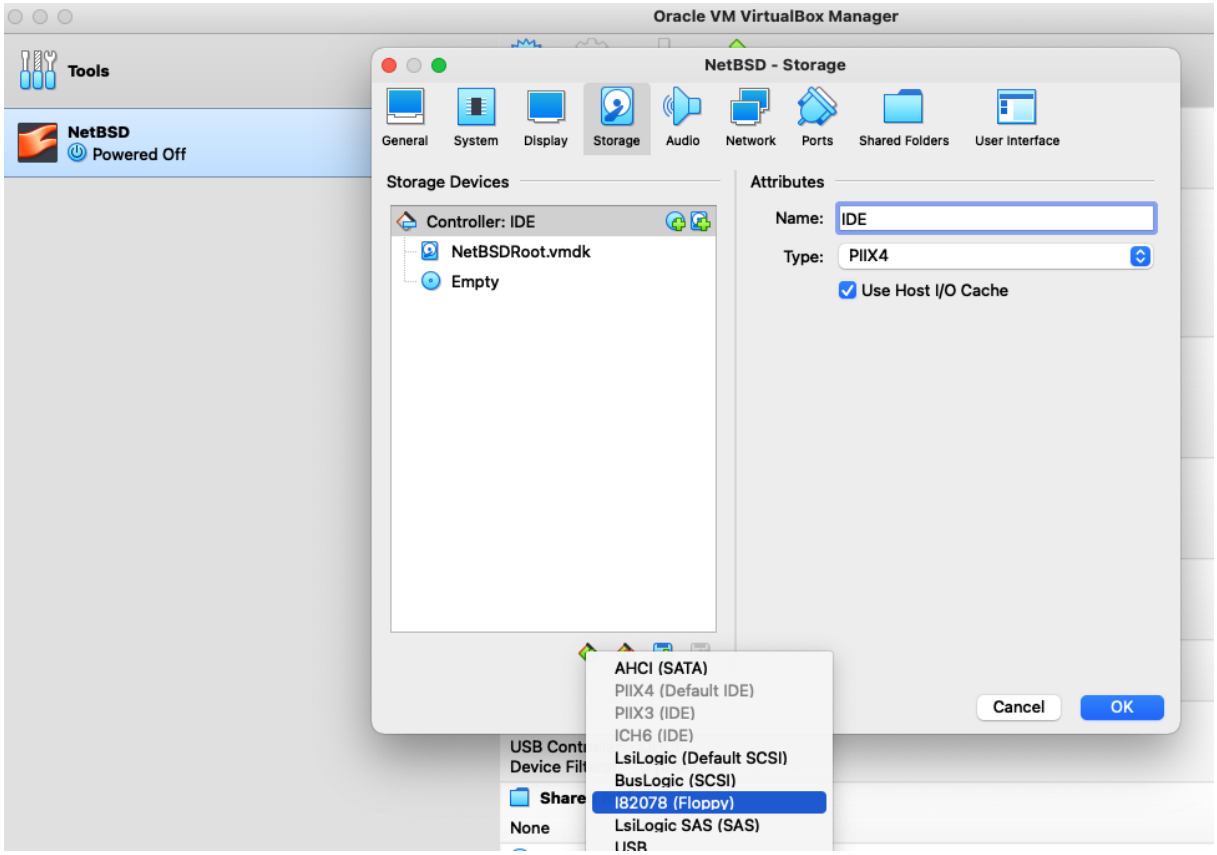



Do not try to use a larger drive here because NetBSD 1.0 will not cope and you are likely to end up with a corrupt filesystem if the 2GB barrier is breached. Remember that this is an operating system from the 90s and a 1GB drive was huge for its time.

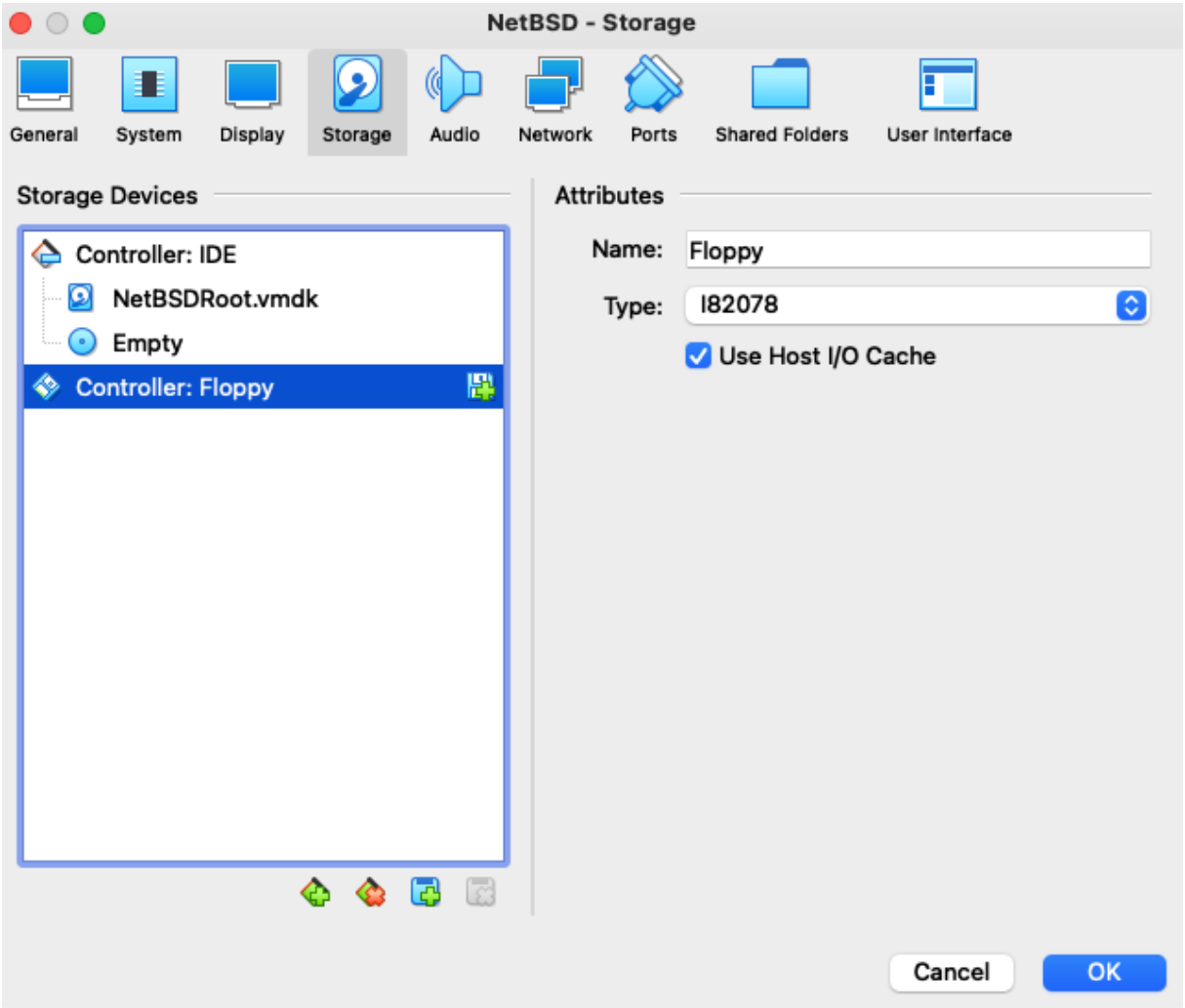
3. Rename the disc files so that VirtualBox will recognise them (it needs a file with .img at the end):

```
% mv inst-10.fs inst-10.img
% mv kcaha-10.fs kcaha-10.img
```

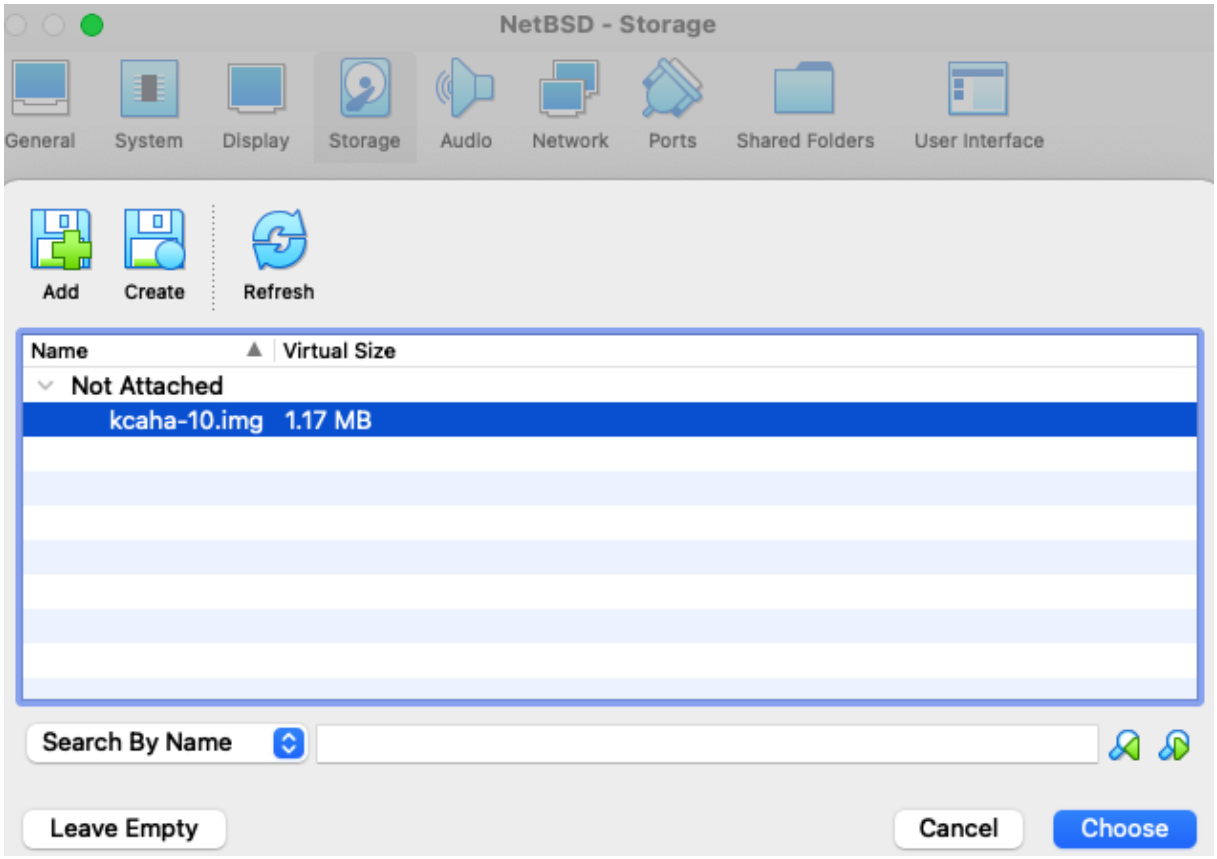
4. Add a floppy drive to the machine. Go to Settings, then Storage and add a floppy drive using the left-most addition button at the bottom of the controller list.




5. Select the Kernel floppy in VirtualBox. Use the disc add button next to the floppy controller:




Then choose the kernel floppy disc image:

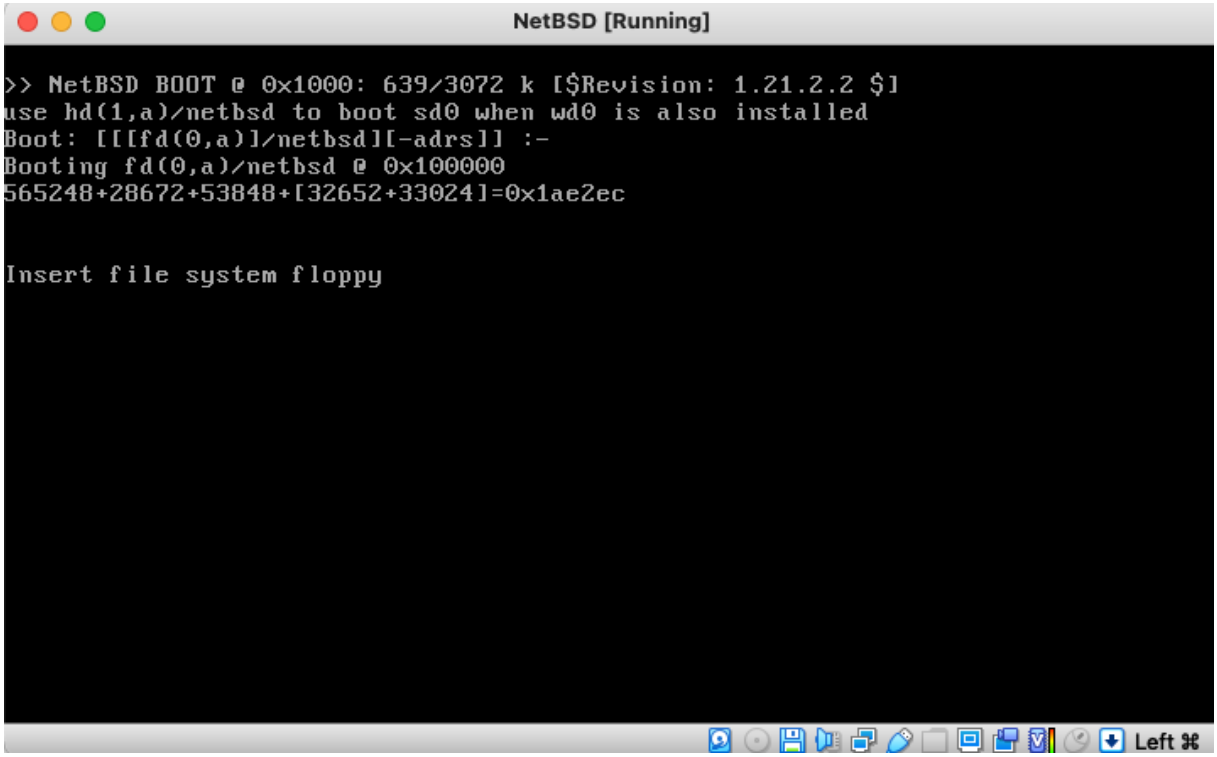



6. After saving the settings, we need to change the density of the floppy drive. This cannot be done in the GUI. In a terminal window, set the density of the Floppy Drive to 1.2MB, otherwise the system will not boot from the drive correctly.[5]

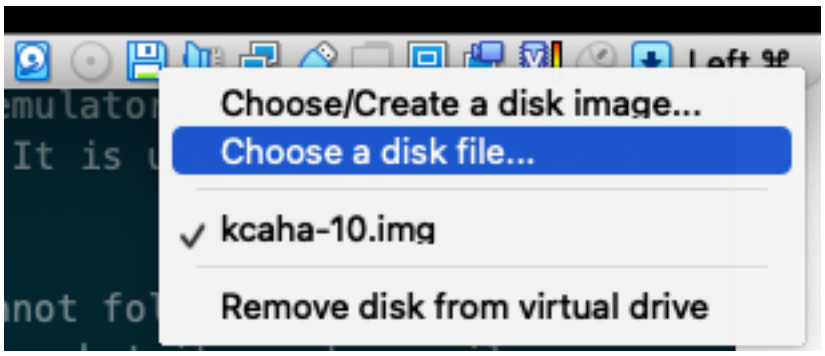
```
% VBoxManage setextradata "NetBSD"
   VBoxInternal/Devices/i82078/0/LUN#0/Config/Type "Floppy 1.20"
```

7. Start the VM with the Start button. The kernel floppy will boot.

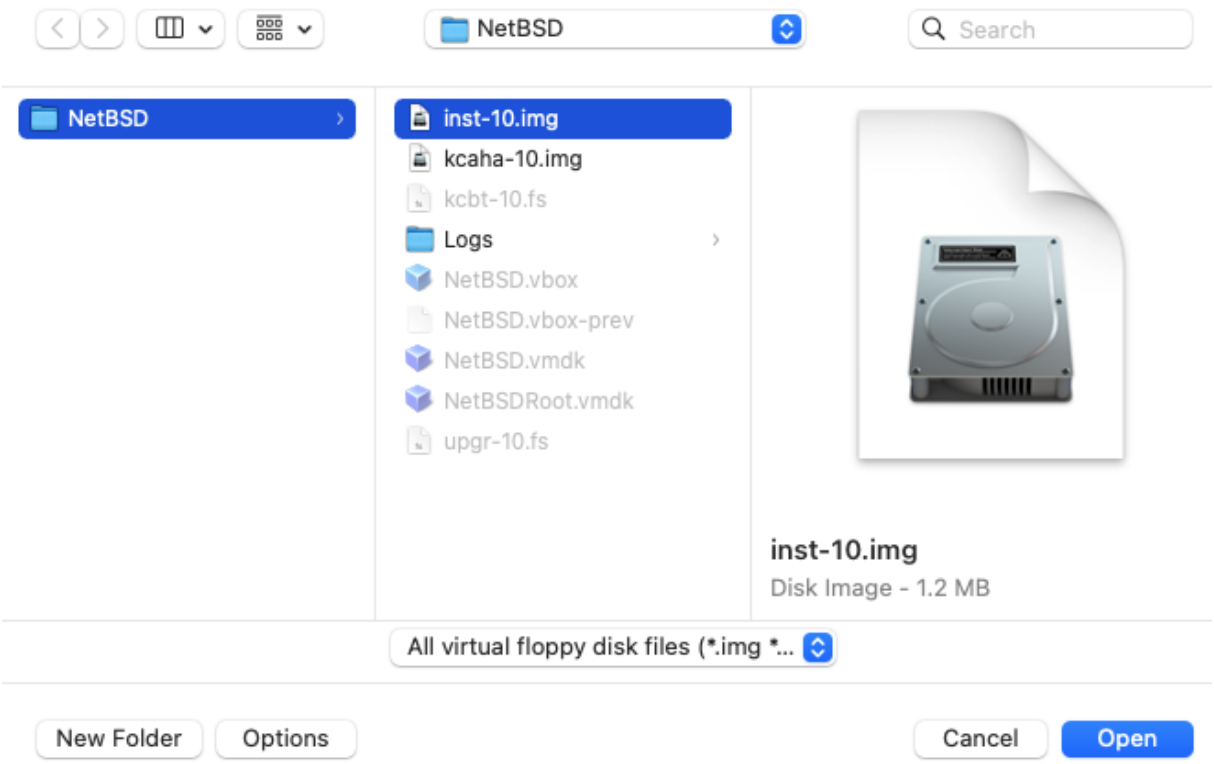
[5]The command is one line. I have excluded the usual backslash to assist with cut and paste.



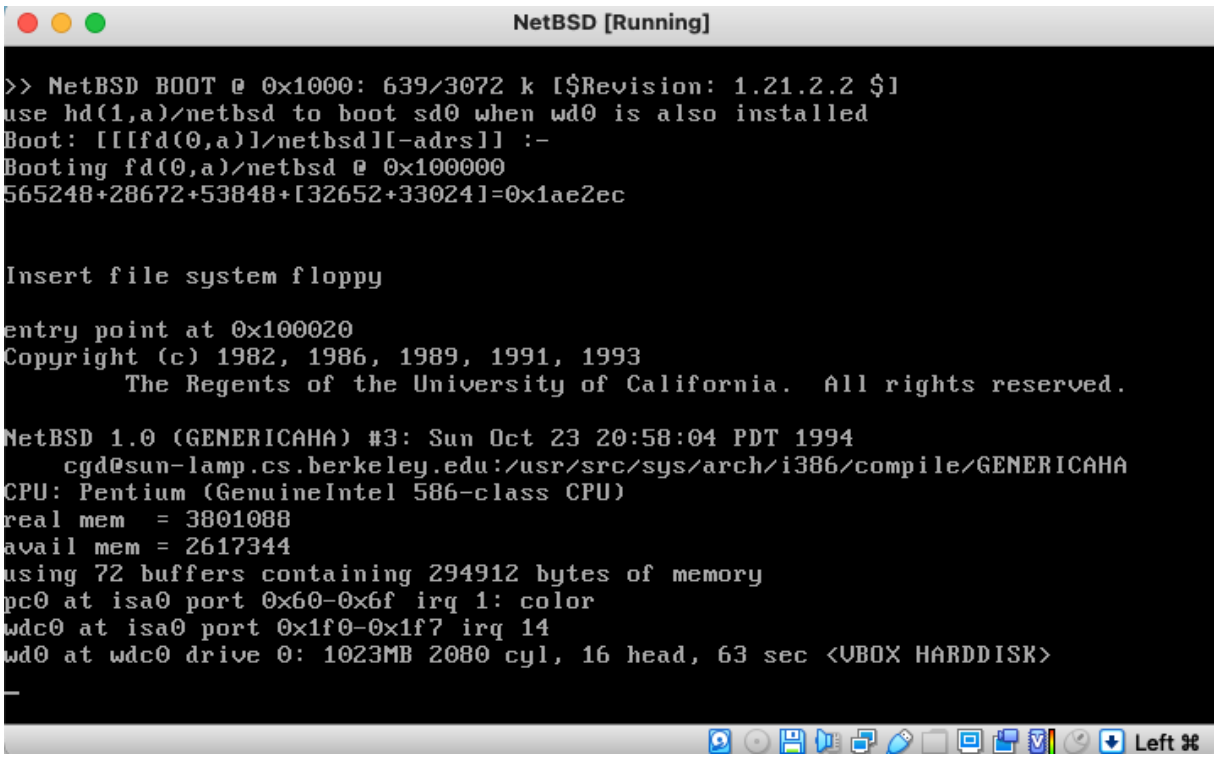

8. Now swap the floppy to the installation disc and then press enter on the window. Click the floppy disc icon in the bottom right of the emulator screen:




And choose the installation floppy image:




9. The system will boot. During the boot, make sure you write down the disc geometry from the kernel messages. For example, 2080 cylinders 16 heads, 63 sectors.



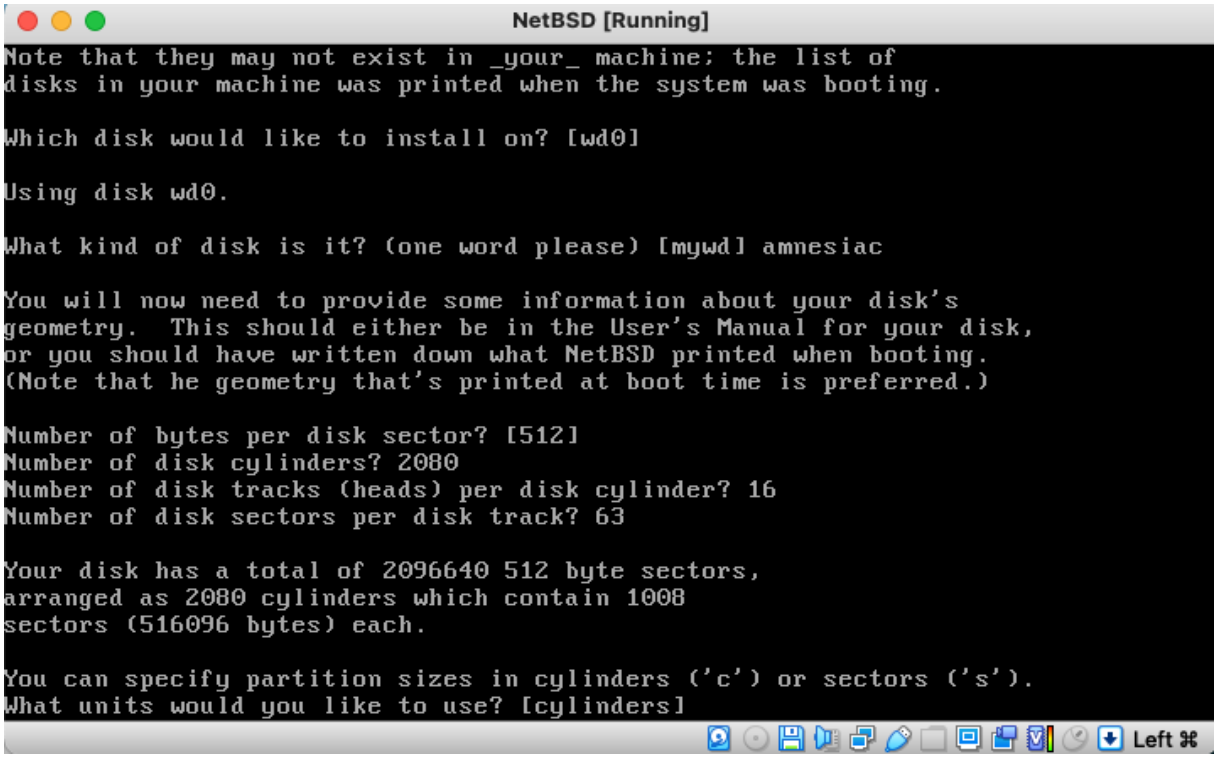

Once the system has booted, press enter at the prompt:

```
fdc0 at isa0 port 0x3f0-0x3f7 irq 6 drq 2
fd0 at fdc0 drive 0: 1.2MB 80 cyl, 2 head, 15 sec
fd1 at fdc0 drive 2: density unknown
pci0 at isa0 port 0x0-0x665: configuration mode 1
pci0 bus 0 device 0: identifier 12378086 class 06000002 not configured
pci0 bus 0 device 1: identifier 70008086 class 06010000 not configured
pci0 bus 0 device 2: identifier 040515ad class 03000000 not configured
pci0 bus 0 device 3: identifier 100e8086 class 02000002 not configured
pci0 bus 0 device 4: identifier cafe80ee class 08800000 not configured
pci0 bus 0 device 5: identifier 24158086 class 04010001 not configured
pci0 bus 0 device 6: identifier 003f106b class 0c031000 not configured
pci0 bus 0 device 7: identifier 71138086 class 06800008 not configured
ie0: unknown AT&T board type code 15
npx0 at isa0 port 0xf0-0xff: using exception 16
biomask 4040 netmask 2 ttymask 2
changing root device to fd0a
WARNING: no swap space found
/etc/rc: Can't open /etc/rc
Enter pathname of shell or RETURN for sh:
```

10. Now follow the installation instructions. We summarise them here, but please read them at [10].

- Proceed with the installation and choose IDE.
- wd0 is the correct drive
- We call the disc *amnesiac*
- Put the disc geometry numbers into the next questions

- Use cylinders for the disc metric
- Use the full disc (e.g. 2080 cylinders) for NetBSD



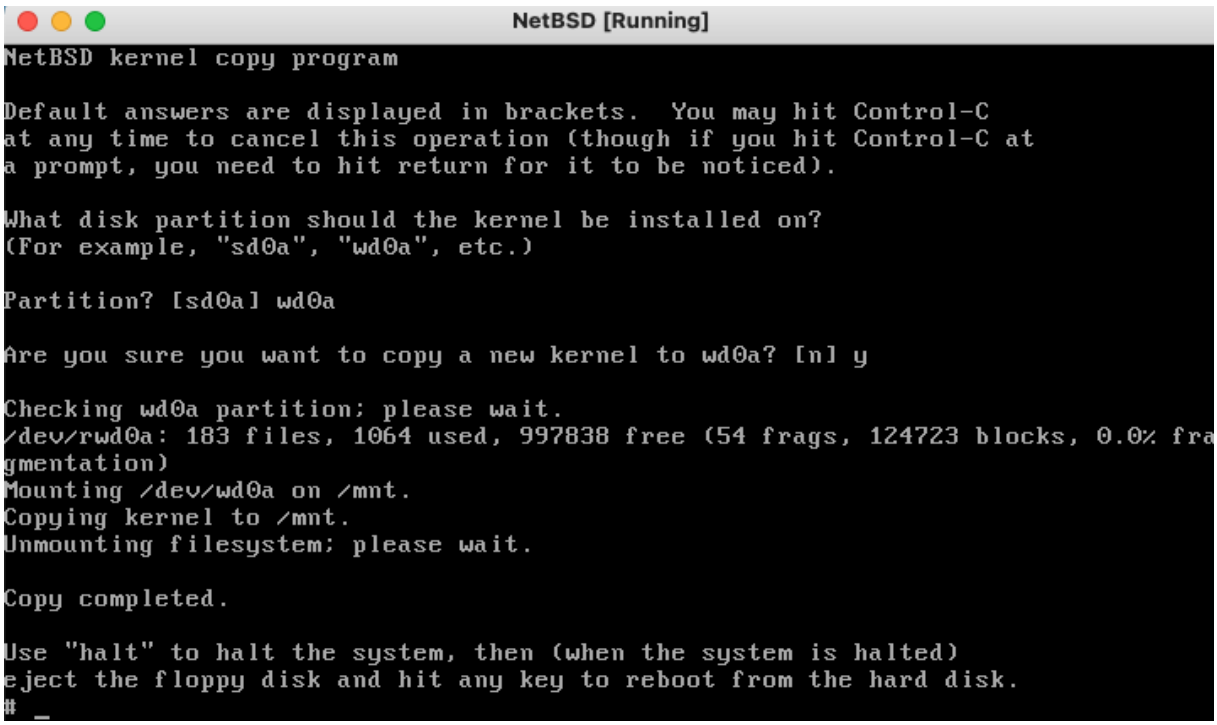

- For the root partition, use 208 cylinders
- For the swap partition use the same
- Use the rest (e.g. 1664 cylinders) for /usr

11. Now shutdown the machine and remove the virtual floppy disc. Replace it with the Kernel boot floppy and reboot.[6]

11. This time, do not replace the disc when asked to "Insert file system floppy" - just press Enter.

12. When booted, press Enter and run *copy_kernel* to put a kernel file onto the hard disc.

[6] Sometimes VirtualBox crashes when rebooting. If this happens, simply power it on again.




13. Shutdown the system again, remove the floppy and reboot the VM. It should boot from the hard drive image.

14. You now have a virtual machine with a very basic NetBSD installation on it. The next steps are to get the rest of the installation media. We will do this via floppy images.

15. Firstly some precautions. We will preserve a copy of */bin/cat*. This is a special "crunched" binary that incorporates several programs into one. Crunched binaries help preserve space on the installation media. In */bin* you will see 18 files which are hard-linked together. Similarly in */usr/bin* there are symbolic links to *cat*. See the *crunchgen(1)* manual page for more details.

Some care is needed as the installation process overwrites these links and if something goes wrong during the process, it will be difficult to recover.

```
# cp /bin/cat /crunched
```

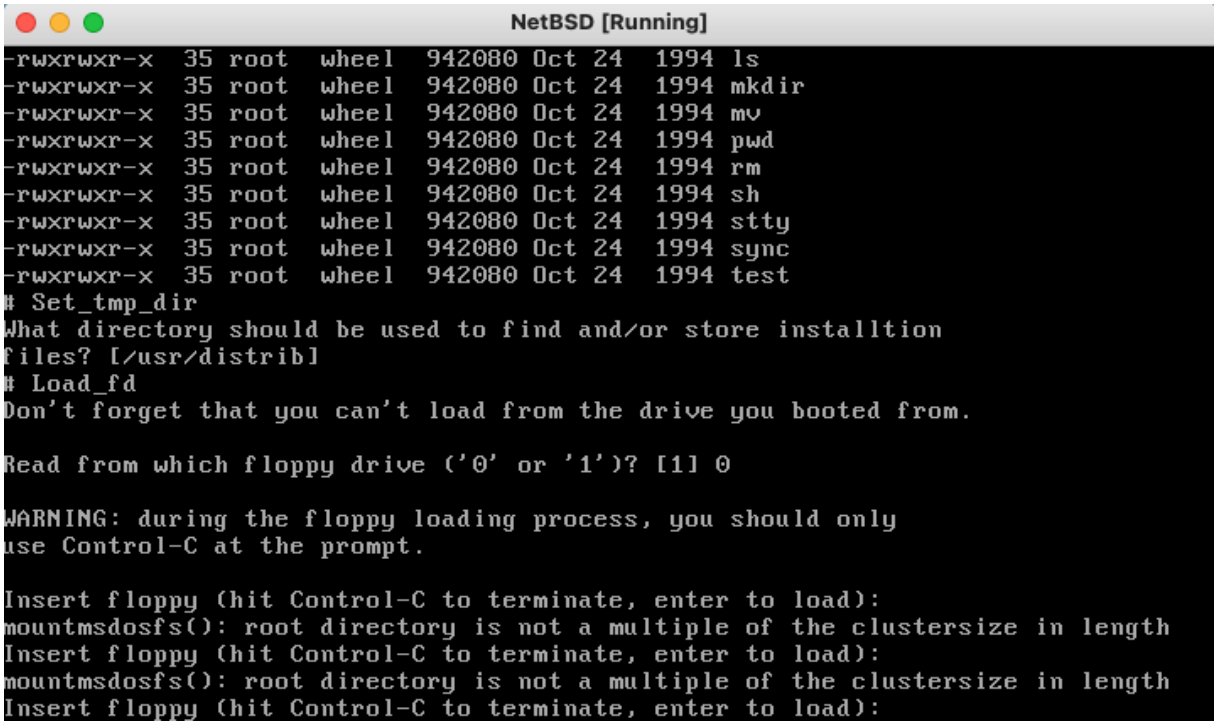
(Another thing you can do now is power down your VM and make a backup of the VMDK file.)

The installation utilities are in a file called */.commonutils*. We will use the shell functions in there to install the system [7].

16. Use *Set_tmp_dir* to setup where you will load the contents of the floppies onto the hard drive (e.g. */usr/distrib*). Use *Load_fd* and choose drive 0. Now, using the floppy icon, insert each installation

[7] Note that the installation routines are using */bin/sh* and not */bin/csh*.

media floppy disc and then hit enter. The contents will be loaded into */usr/distrib*.[8] Repeat for discs 0 through to 17. After copying the last one, don't forget to virtually eject it from the VM.




18. Now install the binaries using the *Extract* shell function

```
# cd /usr/distrib
# Extract base10
# Extract comp10
# Extract etc10
# Extract man10
# Extract text10
# Extract secr10
# Extract games10
# Extract misc10
# Extract text10
```

19. If successful, the files in */usr/distrib* can be safely removed.

20. Tailor the system to your needs by adjusting the files in */etc* and reboot the system. I changed the hostname to *amnesiac* by editing */etc/myname*.

21. Log in as root and have a look around. Navigating around the machine, we see that we are back in 1994. Root's shell is *csh* by default.

```
NetBSD/i386 (amnesiac) (ttyv0)
```

[8] If you are getting frustrated at this point, imagine having to do this with real floppy discs.

```
login: root
Last login: Sun Mar 13 16:08:47 on ttyv0
Mar 13 16:12:48 amnesiac login: ROOT LOGIN (root) ON ttyv0
Mar 13 16:12:48 amnesiac login: ROOT LOGIN (root) ON ttyv0
Copyright (c) 1980,1983,1986,1988,1990,1991 The Regents of the University
of California.  All rights reserved.

NetBSD 1.0 (GENERICAHA) #3: Sun Oct 23 20:58:04 PDT 1994

Welcome to NetBSD!

Terminal type? [pc3]
Don't login as root, use su
amnesiac# cc -v
gcc version 2.4.5
```

22. Now you can install the source sets for 1.1 which will be needed for the next step. To do this, change to */bin/sh* and source the installation routines stored in */.commonutils*. Obtain the source floppy images and then load the discs as before. The sources will appear in */usr/src*:

```
# /bin/sh
# . /.commonutils
# Load_fd
# Extract src11
# Extract ksrc11
# Extract gsrc11
# Extract dsrc11
# Extract ssrc11
```

23. If you will switch to QEMU later and want to follow the full upgrade process, it might be easier to copy the 1.2 source floppies to the image now.

I have provided a VM in VMDK format with the sources for 1.1 extracted in */usr/src* and the sources for 1.2 ready in */usr/distrib* [D4]. Similarly this is available in QEMU format [D5].

## 5.5 Converting to QEMU

*QEMU* is a freely available emulator supporting many architectures including i386. You can obtain it from https://www.qemu.org, or directly from the packaging system on your machine. For example:

```
# NetBSD from source
cd /usr/pkgsrc/emulators/qemu && make install
# or 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
```

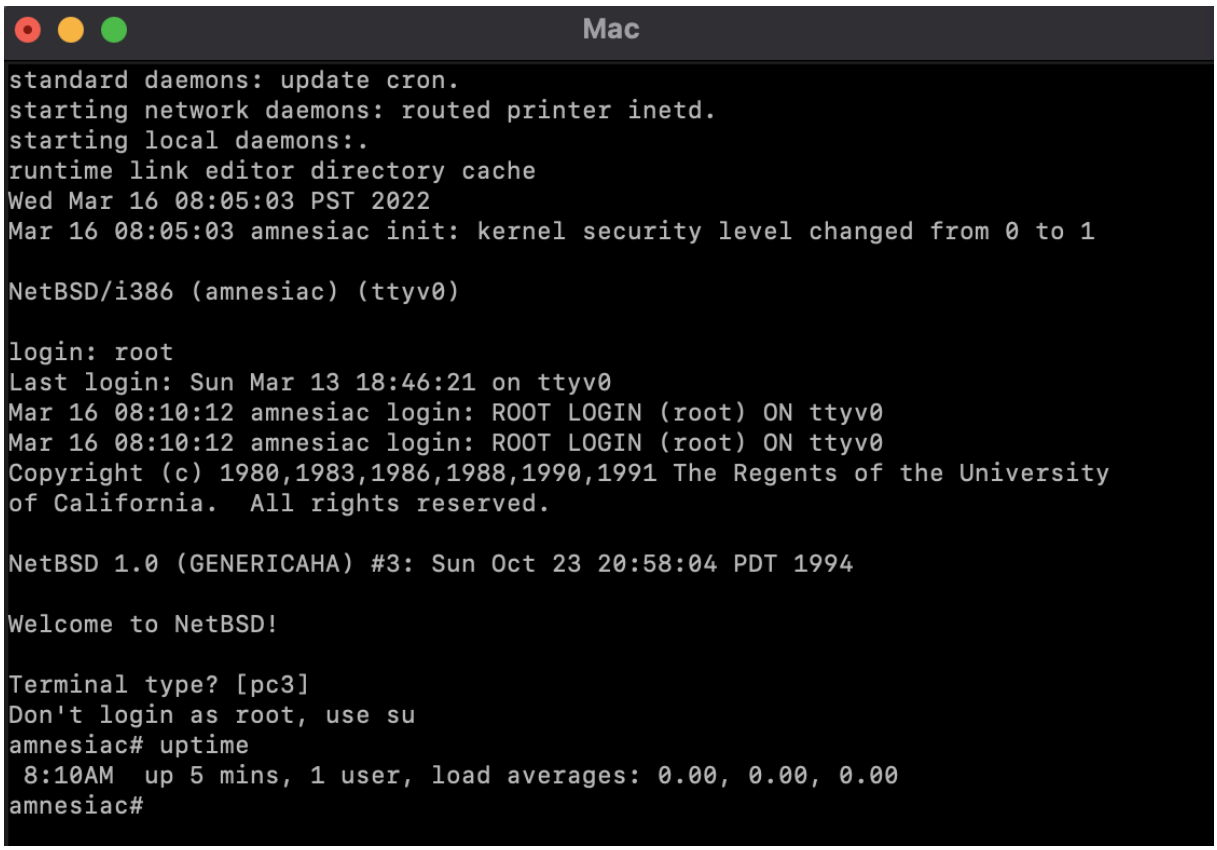
The image we have created can be easily converted to a QEMU machine format such as *qcow2*. First, locate the *VMDK* file in the VirtualBox directory and copy it to a new directory for use with QEMU. Then use *qemu-img* to convert it to *qcow2*, as follows:

```
% mkdir ~/Scratch
% cd ~/Scratch
% cp ~/VirtualBox\ VMs/NetBSD/NetBSD.vmdk .
% qemu-img convert -O qcow2 NetBSD.vmdk netbsd-i386-1.0.img
```

This VM can now be booted from QEMU, like so:

```
% qemu-system-i386 -hda netbsd-i386-1.0.img  -boot c
```

The version of QEMU on your system may support an emulated console in the terminal. You can start the system using `-display curses`. It is easier to cut and paste from such a terminal.




We have provided a VM of NetBSD 1.0 with the sources extracted for convenience [D5].

## 5.6 Approach to upgrading by source

The NetBSD source code is designed to be built with *make* and top-level *Makefile* has several targets that will help us. The *build* target attempts to build everything in the source tree and install it. But unfortunately it will not always work as-is between releases without some tinkering. The reason for this is that between major versions there are interdependencies between the different software components where certain tools need to be upgraded before others.

As a general approach, we will:

- Build the new kernel first and reboot, building any tools we need to achieve this (usually *config* but sometimes others)
- Keep a copy of each kernel we build
- Then build the *userland*, the regular software of the system
- Sometimes it will be necessary to rebuild the compiler, assembler and other tools first
- But we will try to use *make build* as much as we can
- If we come across an awkward binary, we will leave it out until we can build it

- We will not update /etc unless we absolutely need to
- We will preserve VMs between releases

Throughout the process I have preserved each kernel in the root filesystem. For example when upgrading from 1.0 to 1.1, I will do the following:

```
# cp /netbsd /netbsd.10   # make sure I have a copy of 1.0
# cp netbsd /netbsd.11    # archive copy
# cp netbsd /             # new installed copy
```

When you have finished upgrading between releases, shutdown the virtual machine, exit QEMU and make a copy of the virtual machine image. For practical purposes, it is good to be able to go back to a known-good checkpoint. I recommend preserving a VM image after upgrading to a new release and also after rebooting with a new kernel. For example, after finishing the upgrade to 1.1, stop QEMU, store a copy of the VM and then make a fresh copy for the upgrade to 1.2.

```
% mkdir ~/VMstash
% cp amnesiac-netbsd-i386-1.1.img ~/VMstash
% cp amnesiac-netbsd-i386-1.1.img amnesiac-netbsd-i386-1.2.img
```

Throughout the process, I have preserved a copy of each release VM and they can be found here [D8].

## 6 Upgrading NetBSD 1.0 to 1.1

1. Start the VM in QEMU and login as root. In the default installation, we will get a warning that the shell is running as root with the current working directory in the PATH. To fix this edit both *.cshrc* and *.profile* in */* and */root*. Then logout and login again.

2. The first step is to build the tools necessary to upgrade the kernel. The kernel is the operating system binary that is loaded at boot time. First we rebuild *config*. This program takes a kernel configuration file and makes a build directory for it.

(In this paper, when we change directory a source directory it is assuming that the path is relative to */usr/src*. The root *csh* profile includes a hook to find the right directory wherever you are on the filesystem.)

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

3. Upgrade *make* and *mkdep*. *Make* is used to build programs and *mkdep* is used to make dependency files.

```
# cd usr.bin/make
# make && make install
```

```
# cd ../mkdep
# make && make install
```

4. Upgrade *yacc*. *Yacc* (Yet Another Compiler Compiler) produces C code from compiler grammar files. It is used to build compilers.

```
# cd yacc
# make depend && make && make install
```

5. Upgrade the assembler *gas*. The assembler turns compiler output into machine language objects.

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

6. Configure and build the kernel specified in GENERICOTHER.

```
# cd sys/arch/i386/conf
# config GENERICOTHER
# cd ../compile/GENERICOTHER
# make depend && make
# cp /netbsd /netbsd.10
# cp netbsd /netbsd.11
# cp netbsd /
```

7. Reboot and login again.

8. Now we update the supporting files on the system. The first command updates the Make libraries used by the system source code. The second installs the latest C header files and the third makes sure that all directories we need are installed.

```
# cd /usr/src/share/mk && make install
# cd include && make install
# cd etc && make DESTDIR=/ distrib-dirs
```

9. Bootstrap *lex*. *Lex* is a lexical analyser used when building the compiler and other tools. There is a version of *lex* on the system but it is not sufficient to build the new version. There is a contingency for "first time builds" which we use here:

```
# cd lex
# cp initscan.c scan.c
# make && make install && make clean
```

5. The *libcurses* library needs to be updated for *libterm* to build correctly (strictly for a compatible includes file). We do this here so that *libterm* builds later on. Both of these libraries are used for terminal interaction.

```
# cd lib/libcurses && make depend && make && make install
```

6. Bootstrap *lint*. This program is used for checking C program files for mistakes and formatting errors. *Lint* needs to be bootstrapped because it expects a working *lint* to be installed already and it was not included in 1.0.

```
# cd usr.bin/xlint
# cd lint1 && make && make install
# cd ../lint2 && make && make install
# cd ../xlint && make && make install
# cd .. && make && make install
```

7. Update *rpcgen* used in the build later for building the Remote Procedure Call programs.

```
# cd usr.bin/rpcgen && make && make install
```

8. Next we compile the compiler runtime support library and the C library. Then we rebuild the other programs.

The runtime support library contains supporting code to initialise and exit programs. It is included by the compiler in most programs on the system.

There were three minor version bumps to the C library between 1.0 and 1.1, so it's important we update it first.

```
# cd /usr/src/lib/csu
# make depend && make && make install
# cd /usr/src/lib/libc
# make depend && make && make install
```

9. We rebuilt the assembler earlier. Now we update the linker and the compiler and install them, meaning we have updated the key components of the compiler build-chain[9].

```
# cd gnu/usr.bin/ld && make && make install
# cd gnu/usr.bin/gcc2 && make && make install
```

10. Now we rebuild everything else and install it. We will essentially do what the *build* target does in *Makefile* but we will do it by hand to observe what it is doing. First we build and install the new libraries, then we build and install the rest of userland.

```
# cd /usr/src/lib
# make depend && make && make install
# cd gnu/lib
# make depend && make && make install
# cd ../../domestic/libcrypt
# make depend && make && make install
# cd /usr/src
# make depend && make && install
```

[9] During my experiments, I found it necessary to upgrade the linker directly after the C library. Upgrading all the libraries together results in a broken system.

Once this is completed, you can reboot your system to ensure it is running all the up to date software. It's a good idea to halt the VM and make a copy of it before proceeding.

The VM can be booted with a functioning network interface[10] as follows (changing the filename as appropriate):

```
% qemu-system-i386 -hda amnesiac-netbsd-i386-1.1.img -net user -net nic,
    model=pcnet -boot c
```

The network interface can be configured as follows:

```
# ifconfig le0 10.0.2.5 netmask 255.255.255.0
# route add default 10.0.2.2
```

As an exercise, observe the script */etc/netstart* and see what you need to do to make the IP configuration permanent. This usually involves editing the files */etc/hostname.le0* and */etc/mygate*.

## 7 Upgrading NetBSD 1.1 to 1.2

We follow a similar process to the previous section.

1. Remove the 1.1 sources and extract 1.2 sources using the shell installation routines.

```
# /bin/sh
# mv /usr/src /usr/src.1
# cd /usr/distrib
# . /.commonutils
```

My images include the 1.2 sources, but if you have not used these you will need to load them from floppy disc images with *Load_fd*.

```
# Extract src12
# Extract ksrc12
# Extract ssrc12
# Extract gsrc12
# Extract dsrc12
# rm -rf /usr/src.1
```

2. Update *config* and build a GENERICOTHER kernel. Reboot the system.

3. Rebuild *make*, *yacc*, *lex*, *lint* (in *usr.bin/xlint*), *nm* and *tsort*.

5. Install the directories, make files and includes

```
# cd etc && make DESTDIR=/ distrib-dirs
# cd ../share/mk && make install
```

[10] There are only a few things missing, the most notable being a tool to build a CD image bootable on Macs for NetBSD/macppc.

```
# cd /usr/src && make includes
```

6. Rebuild the archiver *ar*, assembler *gas* and linker *ld*

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

7. Rebuild the compiler *gcc*. This requires a bootstrap step. See the instructions at [13]. To quote:

> If you are still running 2.4.5 as your cc1, you will not be able to compile libgcc. To do that, do a make up until it fails with libgcc. It will have made gcc, cc1, cc1plus, cc1obj and cpp. Install at least gcc (cc), cc1, and cpp. Then you can go into libgcc and make libgcc.a. Make sure you then install the new libgcc.a.

```
# cd gnu/usr.bin/gcc
# make
...fails...
# cd cc && make install
# cd ../cc1 && make install
# cd ../cpp && make install
# cd ../libgcc && make && make install
# cd ..
# make clean && make && make install
```

8. Before building everything we put a small workaround in place for *bin/ln*. Edit its Makefile and remove *symlink.7* from MAN=. There appears to be a small bug somewhere preventing installation. The consequence is missing one manual page for now.

9. Now to rebuild everything else and install it. We will essentially do what the build target does without the cleaning step but we will do it by hand:

```
# cd /usr/src
# cd lib
# make depend && make && make install
# cd ../gnu/lib
# make depend && make && make install
# cd ../../domestic/lib
# make depend && make && make install
# cd /usr/src
# make depend && make && install
```

10. Reboot the VM as normal. You can also clean */usr/distrib* of source floppy files.

# 8 Upgrading NetBSD 1.2 to 1.2.1

This is not a difficult upgrade. First obtain the sources [D6] and put them on a nearby FTP server that your VM can reach. Then transfer them to the VM and extract them. We've included the network configuration here.

```
# ifconfig le0 10.0.2.5 netmask 255.255.255.0
# route add default 10.0.2.2
# cd /usr
# rm -rf src
# cd distrib
# ftp 192.168.178.10
  ....
ftp> pass
ftp> prompt
ftp> mget netbsd-1-2-1-source-code.tar.gz
# mv *121/* .
# rmdir src121 ksrc121 ssrc121 gsrc121 dsrc121
# /bin/sh
# . /.commonutils
# Set_tmp_dir
# Extract src121
# Extract ksrc121
# Extract ssrc121
# Extract gsrc121
# Extract dsrc121
```

Then configure and rebuild a kernel in the usual way

```
# cd sys/arch/i386/conf
# config GENERIC
# cd ../compile/GENERIC
# make depend && make
# cp /netbsd /netbsd.12
# cp netbsd /netbsd.121
# cp netbsd /
```

Reboot the system then rebuild userland. The *make build* target will work here without issues as the changes between 1.2 and 1.2.1 are not substantial.

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

# 9 Upgrading NetBSD 1.2.1 to 1.3.3

The 1.3 sources use two new defines `__COPYRIGHT` and `__RCSID` throughout which our system's definitions in */usr/include* are not ready for yet.

One easy way to avoid these is to define `lint` when compiling the first few tools. A less elegant way is to temporarily remove the defines from the sources before we have installed the new includes files.

1. Cleanup the source tree. Obtain the NetBSD 1.3.3 source sets *src.tgz*, *syssrc.tgz*, *sharesrc.tgz* and *gnusrc.tgz* [S13] via a local FTP server and then untar them with *tar zxf*.

NetBSD 1.3 introduced the *pkgsrc* and *xsrc* source modules for the Packages Collection and the X11 Windowing system. Neither of these are required for what we are doing.

2. Rebuild config

```
# cd usr.sbin/config
# make CFLAGS=-Dlint && make install
```

3. Rebuild and install lint

```
# cd usr.bin/xlint && make && make install
```

4. Build a GENERIC kernel and reboot

```
# cd /usr/src/sys/arch/i386/conf
# config GENERIC
# cd ../compile/GENERIC
# make depend && make
# cp /netbsd /netbsd.121
# cp netbsd /netbsd.133
# cp netbsd /
# shutdown -r now
```

5. Rebuild *install*, *make*, *yacc* and *rpcgen*. The new versions are needed in advance of the build.

```
# cd /usr/src/usr.bin/xinstall
# make CFLAGS=-Dlint && make install
# cd ../make
# make CFLAGS=-Dlint && make install
# cd ../yacc
# make CFLAGS=-Dlint && make install
# cd ../rpcgen
# make CFLAGS=-Dlint && make install
```

6. Install the make libraries (this needs the new *install* version). Once this is done, we do not need the lint compiler flag.

```
# cd share/mk && make install
```

7. Make and install the rest of the binaries:

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

8. Reboot. When booting you will see this message. We need to install new boot blocks to bring them up to date. The explanation is in the *installboot(8)* manual page.

```
WARNING: Disk appears to be old-NetBSD or FreeBSD.  See installboot(8).
```

```
# cd /usr/mdec
# ./installboot biosboot.sym /dev/rwd0a
```

8. If you started with an earlier NetBSD before 1.0, you might need to use *fdisk* to change the partition ID from 165 to 169 but this is not necessary in our case.

```
# fdisk
******* Working on device /dev/rwd0d *******
Warning: BIOS sector numbering starts with sector 1
parameters extracted from in-core disklabel are:
cylinders=2080 heads=16 sectors/track=63 (1008 sectors/cylinder)

Figures below won't work with BIOS for partitions not in cylinder 1
parameters to be used for BIOS calculations are:
cylinders=2080 heads=16 sectors/track=63 (1008 sectors/cylinder)

Information from DOS bootblock is:
0: <UNUSED>
1: <UNUSED>
2: <UNUSED>
3: sysid 169 (NetBSD)
   start 0, size 16 (0 MB), flag 0x80
   beg: cylinder    0, head   0, sector  1
   end: cylinder    0, head   0, sector 16
```

## 10 Upgrading NetBSD 1.3.3 to 1.4.3

1. Get the 1.4.3 source sets [S14] and follow the same process as for the upgrade to 1.3.3 to get them onto the machine. Note that on 1.3.3, passive mode is default on *ftp* - issuing *pass* to *ftp* is not necessary.

2. Rebuild and install *config*.

3. The GENERIC kernel will not build without some effort so we will workaround it and come back to it later on.

- Copy the GENERIC configuration file to GENERIC1

- Remove any NFS options (lines 95, 110, 141)
- Remove any references to pciide (lines 373, 400 and 404)
- Remove the wscons section (lines 236-250) and enable pc0 (line 227)
- Build the kernel and install it. Reboot.

Our VM doesn't have a PCI IDE interface, so we are vaguely justified in doing this. Also we can live without the Network File System. We continue to use the pc0 driver because our userland is not ready for the Workstation Independent console drivers *wscons* that were introduced in 1.4.

4. Rebuild *make*, *yacc* and install the *share/mk* files. Then rebuild *lex*.

```
# cd usr.bin/make && make && make install
# cd ../yacc && make && make install
# cd /usr/src/share/mk && make install
# cd ../../usr.bin/lex && make && make install
```

5. Install the new C header files:

```
# cd /usr/src
# make includes
```

6. Build the libraries. The C library build requires us to relax the compiler warnings, otherwise the build will not complete. Set NOGCCERROR to avoid them.

```
# setenv NOGCCERROR
# cd lib && make && make install
```

7. Make sure all the correct directories exist.

```
# cd etc && make DESTDIR=/ distrib-dirs
```

8. The build now includes GNU info pages, so we need *texinfo* in advance of it to build them. To bootstrap we avoid making info pages for *texinfo* by setting MKINFO to no

```
# cd gnu/usr.bin/texinfo
# make MKINFO=no && make MKINFO=no install
```

9. Build a new linker, assembler, compiler and tools;

```
# cd /usr/src/gnu/lib/libbfd && make && make install
# cd /usr/src/gnu/usr.bin/ld && make
# cd ../gas.new && make
# cd ../binutils && make
# cd ../egcs && make

# cd ../ld && make install
# cd ../gas.new && make install
# cd ../binutils && make install
```

```
# cd ../egcs && make install
```

10. Rebuild the system fully and reboot.

```
# unsetenv NOGCCERROR
# cd /usr/src
# make build
# shutdown -r now
```

11. We have a working 1.4 system with a slightly modified GENERIC kernel and the old pc0 driver. Now we should be able to build GENERIC properly and use the new drivers

```
# cd sys/arch/i386/conf
# config GENERIC
# cd ../compile/GENERIC
# make depend && make
# cp /netbsd /onetbsd
# cp netbsd /netbsd.143
# cp netbsd /
```

12. Before rebooting fix *ttys* and devices for the *wscons* driver.

```
# cd etc/etc.i386
# cp ttys /etc
# cp MAKEDEV /dev
# cd /dev && ./MAKEDEV all
```

13. Reboot

## 11 Upgrading NetBSD 1.4.3 to 1.5.3

NetBSD 1.5 introduced the ELF binary format for the i386 platform and the upgrade is a little trickier as a result. I have written a separate paper [12] on this topic in which I start with a 1.4.3 system installation and upgrade to 1.5. Please refer to that paper for the detail.

We have avoided updating */etc* up to now and have got away with it so far. Because we started with an earlier NetBSD and not a clean 1.4.3 installation, we need to make one adjustment before upgrading to 1.5.3. The build needs some additional groups. Update */etc/group* from */usr/src/etc/group* (e.g. by just copying it if you have not made local modifications) before *make build*.

After upgrading to 1.5.3, */etc* needs be tidied up particularly due to the introduction of the *rc.d* system. For now, one can do the following:

- copy */usr/src/etc/defaults/*.conf* to */etc/defaults*
- From */usr/src/etc*, copy *rc*, *rc.lkm*, *rc.local*, *rc.shutdown*, *rc.subr* and *rc.conf* to */etc*.

- Install */usr/src/etc/rc.d* in */etc* (use *make install*)
- edit *rc.conf* and set *rc_configured* to *YES*
- check */etc/defaults/rc.conf* and set variables in */etc/rc.conf* to suit your installation. For example:

```
dhclient=YES
dhclient_flags="le0"
```

- */etc/mygate* is no longer needed
- */etc/fstab* now needs an entry for swap

```
/dev/wd0b none swap  sw 0 0
```

In later releases, the *postinstall* script will help us tidy up */etc*.

## 12 Interlude - Amnesiac needs new (virtual) hardware

By the end of the upgrade to 1.5.3, we are approaching the limits of the system we setup for 1.0. For example, the hard disc is unlikely to be large enough for future upgrades. Here we see the filesystem sizes after the 1.5.3 build, then without the objects and then without the source code:

```
# df -k
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/wd0a      101487    49414    46998    51%    /
/dev/wd0e      812014   672890    98523    87%    /usr
# cd /usr/src && make cleandir
# df -k
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/wd0a      101487    49414    46998    51%    /
/dev/wd0e      812014   456733   314680    59%    /usr
# rm -rf /usr/src
# df -k
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/wd0a      101487    49414    46998    51%    /
/dev/wd0e      812014   124943   646470    16%    /usr
```

Shutdown the VM before making any changes. We will assume the image name is *amnesiac-netbsd-1.5.3.img*.

### 12.1 New network card

Under Qemu emulation, the *ne* device works better on NetBSD. We could have switched to this a few releases ago, but seeing as we are fiddling with hardware, let's do it now. In our virtual world, all we need to do is change the model in the QEMU command line options as follows:

```
 -net nic,model=ne2k_pci
```

When we boot the VM next, we can change the configuration in */etc/rc.conf* to run the DHCP client on boot:

```
dhclient_flags="ne2"
```

## 12.2 Bigger hard disc

We will add a second hard disc, partition it and copy the data across. First, shutdown your VM, then make a 20GB image file and boot your VM with it as the second disc.

```
% qemu-img create -f qcow2 amnesiac-netbsd-i386-1.5.3.newhw.img 20G
% qemu-system-i386 -hda amnesiac-netbsd-i386-1.2.img \
    -hdb amnesiac-netbsd-i386-1.5.3.newhw.img -net user \
    -net nic,model=ne2k_pci -boot c
```

On an x86-based machine, the BIOS use disc partitions to separate operating systems and drives. One of these partitions will contain our BSD installation. Within this BIOS partition, BSD uses a *disklabel* to mark out its own partitioning scheme (also called slices) to identify the root partition, swap partition and others.

1. Partition the new image using *fdisk* by obtaining the geometry from the kernel boot messages:

```
# dmesg | grep wd1
...
wd1: 20480 MB, 16383 cyl, 16 head, 63 sec, 512 bytes/sect x 41943040
   sectors
...
# fdisk -u wd1
...
Do you want to change our idea of what BIOS thinks? [n] n
The data for partition 0 is:
<UNUSED>
Do you want to change it? [n]  y
sysid: [0] 169
start: [0]
size: [0] 41943040
Explicitly specify beg/end address? [n]
sysid 169 (NetBSD)
    start 0, size 41943040 (20480 MB), flag 0x0
       beg: cylinder    0, head   0, sector  1
       end: cylinder 1022, head 254, sector 63
Is this entry okay? [n] y
...
Should we write new partition table? [n] y
```

2. Now we initialise the disklabel and add three BSD partitions for /, swap and /usr using a scheme matching our existing disc.

```
# disklabel -i -I wd1
partition> P
8 partitions:
#  size      offset fstype  [fsize bsize cpg/sgs]
c: 41943040  0      unused   0     0         # (Cyl. 0 - 41610\*)
d: 41943040  0      unused   0     0         # (Cyl. 0 - 41610\*)
e: 41943040  0      unused   0     0         # (Cyl. 0 - 41610*)
partition> e
Filesystem type [?] [unused]: <RET>
Start offset [0c, 0s, 0M]:  0
Partition size ('$' for all remaining) [41610.2c, 41943040s, 20480M]: 0
partition> a
Filesystem type [?] [unused]: 4.2BSD
Start offset [0c, 0s, 0M]:
Partition size ('$' for all remaining) [0c, 0s, 0M]: 2080c
partition> b
Filesystem type [?] [unused]: swap
Start offset [0c, 0s, 0M]: 2080c
Partition size ('$' for all remaining) [0c, 0s, 0M]: 2080c
partition> e
Filesystem type [?] [unused]: 4.2BSD
Start offset [0c, 0s, 0M]: 4160c
Partition size ('$' for all remaining) [0c, 0s, 0M]: $
partition> W
Label disk [n]? y
Label written
partition> Q
```

3. Create file systems on the partitions using *newfs*.

```
# newfs /dev/rwd1a
/dev/rwd1a:     2096640 sectors in 2080 cylinders of 16 tracks, 63 sectors
1023.8MB in 130 cyl groups (16 c/g, 7.88MB/g, 1984 i/g)
   super-block backups (for fsck -b #) at:
         32,   16224,   32416,   48608,   64800,   80992,   97184,
         ...
# newfs /dev/rwd1e
```

4. Shutdown and boot single user by interrupting the boot process and typing `boot -s`

5. Check all filesystems and mount them (press return at the first prompt):

```
Enter pathname of shell or RETURN for sh:
# fsck
** /dev/rwd0a
** File system is clean; not checking
** /dev/rwd0e
** File system is clean; not checking
```

```
# mount -a
```

6. Mount the new partitions

```
# mount /dev/wd1a /mnt
# mkdir -p /mnt/usr
# mount /dev/wd1e /mnt/usr
```

7. Copy the data, excluding the mount point /mnt and the device hierarchy /dev. We will carefully use *tar* preserving the permissions on the files with the p option.

```
# tar cpf - .c* .instutils .profile netbsd* altroot \
            bin boot crunched emul etc home install \
            root sbin stand sys tmp usr var | \
            ( cd /mnt && tar xvpf - )
# mkdir /mnt/mnt
```

8. Install the device nodes on the new drive:

```
# mkdir /mnt/dev
# cp /dev/MAKEDEV /mnt/dev
# cd /mnt/dev
# ./MAKEDEV all
```

7. Install a boot block:

```
# cd /usr/mdec
# ./installboot biosboot.sym /dev/rwd1a
```

8. Halt the machine. In the days of physical hardware, we would swap the discs over or enter the BIOS to boot from the other drive. Here we just boot from the new image on Qemu.

```
% qemu-system-i386 -hda amnesiac-netbsd-i386-1.5.3.newhw.img \
    -net user -net nic,model=ne2k_pci -boot c
```

## 13 Upgrading NetBSD 1.5.3 to 1.6.2

1. Obtain the 1.6.2 sources [S16] and decompress into */usr/src*.

2. Update *config* as usual. The kernel uses some new CPU instructions and the assembler needs to be updated before building it. The assembler needs a newer bfd library.

```
# cd usr.sbin/config && make && make install
# cd gnu/lib/libbfd && make
# cd gnu/usr.bin/gas.new && make && make install
```

3. Build the GENERIC kernel and reboot

4. Add the following groups to */etc/group/*

```
named:*:14:
ntpd:*:15:
sshd:*:16:
```

Add the following users to the password file using *vipw*

```
postfix:*:12:12::0:0:& pseudo-user:/var/spool/postfix:/sbin/nologin
named:*:14:14::0:0:& pseudo-user:/var/chroot/named:/sbin/nologin
ntpd:*:15:15::0:0:& pseudo-user:/var/chroot/ntpd:/sbin/nologin
sshd:*:16:16::0:0:& pseudo-user:/var/chroot/sshd:/sbin/nologin
```

4. Use the power of build.sh.

The rest of the process of very easy thanks to the power of the *build.sh* script which was introduced with the 1.6 release. The script was designed to build the sources in the correct order and also to cross-build for different machines. It is still possible to use *make* but you need to figure out the dependencies between the build tools and the sources as we have been doing up to now. The shell script *build.sh* builds a sandboxed toolchain first and then builds the programs. You can simply run the script and go and get coffee.

```
# mkdir -p /usr/obj
# cd /usr/src
# ./build.sh
```

5. 1.6.2 shipped with the *postinstall* script to check for configuration problems. Running with *check* will explain what is wrong and running with *fix* will attempt to fix the problems. Some problems may need fixing by hand, particularly if local configurations need to be merged.

```
/usr/src/etc/postinstall check
/usr/src/etc/postinstall fix
```

## 14 Upgrading from 1.6.2 to 2.0.3

This is even easier than the upgrade to 1.6 as the *build.sh* script had matured into a powerful tool capable of doing everything needed. The script resembles the *build.sh* script of today. Builds of NetBSD require little intervention with *build.sh* and it is possible to use it on other platforms such as Linux, OpenBSD and FreeBSD. In the latest releases of NetBSD, *build.sh* is capable of cross-building releases of all supported architectures.[11]

1. Retrieve the sources for 2.0.3. The *cvs* source control tool shipped with 1.6.2 so it is possible to

[11] There are only a few things missing, the most notable being a tool to build a CD image bootable on Macs for NetBSD/macppc.

update the sources. You will have to use the *pserver* method as the *ssh* binary on 1.6.2 is not capable of connecting to a modern *sshd*.

```
# cd /usr/src
# cvs -q -d :pserver:anoncvs@anoncvs.netbsd.org:/cvsroot update -r netbsd
   -2-0 -dP
```

It may be as well to run the *cvs* update again to check it has completed successfully without any conflicts or merges. Additionally any files marked with '?' can be safely removed as they are remnants of a previous build.

2. Use *build.sh* to build a kernel and reboot. See also *BUILDING* in the top-level source directory.

```
# rm -rf /usr/obj
# mkdir -p /usr/obj
# cd /usr/src
# ./build.sh -u tools
# ./build.sh -u kernel=GENERIC
# cp /usr/src/sys/arch/i386/compile/obj/GENERIC/netbsd /netbsd.203
# cp /netbsd.203 /netbsd
# shutdown -r now
```

3. Rebuild userland, install and fix the configurations[12]. Some may need fixing by hand.

```
# rm -rf /usr/obj
# mkdir -p /usr/obj
# cd /usr/src
# ./build.sh -u -O /usr/obj distribution
# ./build.sh -u -O /usr/obj install=/
# /usr/src/etc/postinstall check
# /usr/src/etc/postinstall fix
```

## 15 Later releases

It's possible to upgrade through every major version using *build.sh* as above. Here are the branches to use to get the last release with bug fixes for each major version, e.g. netbsd-2 will actually be called 2.1_STABLE because it may have some patches added to 2.1.

| Version | Branch |
|---|---|
| 2.1 | netbsd-2 |
| 3.1 | netbsd-3 |

[12] different variants of build.sh usage will work here depending on the version of build.sh. See *BUILDING*.

| Version | Branch |
|---|---|
| 4.0.1 | netbsd-4 |
| 5.2.3 | netbsd-5 |
| 6.1.5 | netbsd-6 |
| 7.2 | netbsd-7 |
| 8.2 | netbsd-8 |
| 9.3 | netbsd-9 |

Between each release, it is advisable to remove the object files in */usr/obj* and older toolchain directories. Take care to read the UPDATE file as it will usually have information about known problems between releases. Also read the BUILDING file to check on changes for *build.sh*. In v3, *postinstall* is installed in */usr/sbin* and can be run from there.

## 15.1 Upgrading from 2.1 to 3.1

When updating the src tree, I ran into a problem with the source directory *usr.sbin/racoon*, possibly caused by a reorganisation of this directory between the releases.[13]. Deleting the directory and updating using *cvs update -dP* will fix any discrepancy.

The v3 releases introduced Pluggable Authentication Modules, changing the way that authentication is done on the system. You will not be able to login after a reboot if you have not updated /etc properly with *postinstall*.

*dhclient* will not work properly unless the device nodes have been rebuilt.

```
amnesiac# cd /dev
amnesiac# ./MAKEDEV all
```

The *postinstall* script has been installed in */usr/sbin* since version 3.

```
# /usr/sbin/postinstall check
# /usr/sbin/postinstall fix
```

[13] There was a switch from the KAME toolset to the ipsec-tools code.

## 15.2 Upgrading from 4 to 5.2

The v5 release had issues with some ACPI implementations and the 5.2 kernel will not boot properly on the QEMU emulator. In our situation we can turn ACPI off on QEMU. (If we were doing this for "real" on hardware, our machine would likely not have had ACPI.)

```
% qemu-system-i386 -hda amnesiac-netbsd-i386-5.img -net user -net nic,
    model=ne2k_pci -boot c -no-acpi
```

## 15.3 Upgrading from 7.2 to 8.2

The filesystem super-block format changed between 7.2 and 8.2. You will need to tune the filesystems with *fsck* so that they can be mounted correctly at boot. If you boot 8.2 without converting, you will get this error:

```
Supported file systems: mfs lfs ext2fs ffs nfs umap procfs
overlay null kernfs fdesc union tmpfs smbfs puffs ptyfs
ntfs msdos cd9660 coda
no file system for wd0 (dev 0x0)
cannot mount root, error = 79
```

To rectify, interrupt the boot process and boot a 7.2 kernel in single user mode. Then use *fsck_ffs -c 4* to update the super-block format for each filesystem you have.

```
boot /netbsd.72 -s
...
# fsck_ffs -c 4 /dev/rwd0a
** /dev/rwd0a
** File system is already clean

CONVERT TO NEW SUPERBLOCK LAYOUT? [yn] y
```

## 15.4 Miscellaneous problems

- If for some reason the DHCP client does not work, you can always resort to manual configuration (assuming a QEMU setup):

```
ifconfig ne2 10.0.2.5 netmask 255.255.255.0
route add default 10.0.2.2
echo "nameserver 10.0.2.3" > /etc/resolv.conf
```

- Sometimes *dhclient* will not start due to a corrupted lease file. Make sure that *dhclient* is stopped and then remove */var/db/dhclient.leases*.

- On some terminals the delete key does not always do what you would expect. If your delete or backspace key results in ^? instead of deleting, you can try this:

```
# stty erase ^?
```

- Don't forget, because you are working in a virtual machine environment you can make copious backups of the VM images and return to a known good checkpoint at anytime.

### 15.5 Building 9.2 directly from earlier releases

During the process of subsequent upgrades, we tried to build NetBSD 9.2 on each earlier version. The earliest release we were able to build 9.2 on was 6.1. It may be possible on earlier NetBSD releases, but we will leave this as an exercise to the reader. NetBSD 9.3 was released during the preparation of this paper, but it is likely that the same holds true.

## 16 Things I might have done differently

I'm not likely to do anything like this again. It's something one does for interest and historical nostalgia. However there are two things that I would have done differently.

### 16.1 NetBSD 0.8 and 0.9

It would have been great if I could have started with an installation of NetBSD 0.8 and worked from there. Given the withdrawal of the 0.8 and 0.9 releases due to the lawsuit we mentioned earlier, it is difficult to find installation media for these two releases.

It would have been interesting to see the upgrade from 0.9 to 1.0 because 1.0 introduced dynamically linked binaries.

There are archives of NetBSD 0.8 on the Internet if you look hard enough, but it's not necessarily clear if they are the official binaries or not. Even if the official binaries were available, they may still be reluctance to distribute them because of the lawsuit.

A developer has recently recreated an approximate 0.8 installation from 386BSD and the various patch kits. [11]

### 16.2 Root Disc

When I changed the root disc after the 1.5.4 upgrade, I felt as if I had cheated. It would have been nice for a root disc originally created on 1.0, to have been converted and upgraded throughout the process.

The problem is that NetBSD 1.0 did not support very large hard discs. If you install 1.0 on a machine with a disc sized over 2G, it will not be able to cope. This is simply a matter of the sizes of pointers used in the filesystem code.

If I was doing this again, I would have setup a second hard drive image for */usr*, copied the contents to it and retained the original hard drive image for the root filesystem.

## 17 Amnesiac, the living VM

The NetBSD/i386 VM installed as 1.0 and upgraded through every major release using only the sources can be found on the supporting website to this paper [D7]. We will attempt to keep it up to date for as long as i386 is a supported architecture or the author gets bored.

We will end the paper by checking that binary compatibility was retained throughout the releases. Remember the binary we saved during the installation of NetBSD 1.0? We made a copy in the root directory of the binary */crunched* which contains all of the installation binaries. And it still runs - here we run it with its shell personality.

```
amnesiac# mv /crunched /root
amnesiac# cd /root
amnesiac# file crunched
crunched: a.out NetBSD/i386 demand paged executable
amnesiac# ln -s sh crunched
amnesiac# ./sh
#
```

## 18 Acknowledgments

The paper is dedicated to William Jolitz who laid the foundations for a PC version of BSD. I thank Roland Dowdeswell, Perry Metzger and Arrigo Triulzi for their comments and historical perspective, but also to the NetBSD, OpenBSD, FreeBSD and other communities who keep the spirit of the CSRG alive.

# 19 Bibliography

## 19.1 References

[1] 386BSD, Wikipedia.

[2] 386BSD website, William and Lynne Jolitz.

[3] BSD Licenses, Wikipedia.

[4] Computer Systems Research Group at UCB, Wikipedia.

[5] *The Design and Implementation of the FreeBSD Operating System*, McKusick and Neville-Neil, Pearson, 2005.

[6] The FreeBSD Project, FreeBSD Foundation.

[7] NetBSD, Wikipedia.

[8] The NetBSD Project, NetBSD Project.

[9] NetBSD Changelogs, NetBSD Project.

[10] NetBSD 1.0 installation guide, NetBSD Project.

[11] NetBSD 0.8 VM contructed from 386BSD, Gunkies.org.

[12] Transition from a.out to ELF: Memories of upgrading NetBSD 1.4.3 to 1.5, Chris Pinnock, August 2022.

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

## 19.2 Media

All the prepared media for this paper can be found at http://downloads.chrispinnock.com/netbsdhist/. Specifically:

[D1] NetBSD 1.0 Installation Floppy Images

[D2] NetBSD 1.1 source floppies

[D3] NetBSD 1.2 source floppies

[D4] VirtualBox VM of 1.0 installed with the sources for 1.1 and 1.2

[D5] QEMU version of NetBSD 1.0 VM, same as D4 but for QEMU

[D6] Source code for 1.2.1

[D7] VM images of Amnesiac

[D8] All VM images from 1.0 to 9.2. These are compressed and are set read-only. You will need to change the permissions with chmod before running.

The source sets for 1.3, 1.4 and 1.6 can still be obtained directly from the NetBSD Archive:

[S13] NetBSD 1.3.3 source sets, NetBSD Project.

[S14] NetBSD 1.4.3 source sets, NetBSD Project.

[S16] NetBSD 1.6.2 source sets, NetBSD Project.