NanoBSD and ALIX

In the previous issue of BSD Magazine, Bill Harris described how to do a basic installation of FreeBSD on a PC-Engines ALIX board with a Compact Flash card. This is a great way to get started, but there are some risks to this approach.

What you will learn…

- The working of and working with NanoBSD
- The creation of NanoBSD for an embedded system

What you should know…

- Your way around a FreeBSD system
- Basic system administration
- How to compile FreeBSD from source

CF cards can be written to only a limited number of times so putting your `/var` and its logfiles on it, will quickly wear out the card. To adress this issue and others, Poul-Henning Kamp wrote a script called nanobsd.sh. NanoBSD is not a fork from FreeBSD, but an optimized build script for read-only media. This article gives an overview of NanoBSD in general and my setup in more detail.

The working of NanoBSD

When NanoBSD boots, `/` (and its subdirectories like /boot, `/root` and `/usr`) are mounted as a read-only file system, while `/etc` and `/var` are mounted as read-write file systems on memory disks. The content of these memory disks is lost when the power is lost or when the system reboots.

There is a partition on the NanoBSD CF card reserved for the persistent storage of the `/etc` configuration files. This partition is mounted early in the boot process as /cfg and the files are copied to the `/etc` memory disk. During normal operation the `/cfg` partition is not mounted to prevent accidental writes to the configuration files. A number of scripts are used to keep the `/cfg` partition up to date with changed configuration files in `/etc`.

The CF card contains three partitions in total. The p3 partition is used for the persistent storage of configuration files. The p1 and p2 partitions both contain a/file system. This is very convenient for a system upgrade and roll-back as we will see in a later section.

The building of NanoBSD

NanoBSD is built off-line, which means that the preparation, build and installation process has no impact on the build system or the live NanoBSD system. It will produce image files that can be put on the CF card in the NanoBSD system. There are a number of important files and directories.

nanobsd.sh

The script has been located in the FreeBSD source tree since FreeBSD 6.0 and can be found in the `/usr/src/tools/tools/nanobsd` directory. Running the script with no options will produce a disk image with a GENERIC kernel and a complete world. A nanobsd.conf configuration file can be used to tune the build process.

nanobsd.conf

This configuration file overrides defaults that are set in the `nanobsd.sh` script. These defaults include the name, the architecture, the world options and kernel configuration.

It is also possible to add custom script functions in order to tune the system even further.

Adding ports and files

Because the/file system is read-only, ports have to be added during the build process. All port files (distfiles) that reside in the `/usr/src/tools/tools/nanobsd/Pkg` directory are compiled and installed before the image file is created.

The distfiles must be of the same architecture as the target system and do not forget to install all dependencies as well! These dependencies can be found in the port description on the *www.FreeBSD.org/ports* page. Individual files that reside in the `/usr/src/tools/tools/nanobsd/Files` directory are copied before the image file is created.

Build process

With a source tree of the desired FreeBSD version in `/usr/src`, the commands

```
# cd /usr/src/tools/tools/nanobsd
# sh nanobsd.sh
```

will start the build process. The detailed output of the process is written to logfiles and only high-level progress status is written to the screen. All files are located under `/usr/obj/nanobsd.NANOBSD`. The most important files are

- `_.bw` (build world logfile),
- `_.bk` (build kernel logfile),
- `_.iw` (install world logfile),
- `_.ik` (install kernel logfile),
- `_.disk.full` (full disk image for the entire CF card)
- `_.disk.image` (partition image for one partition on an existing NanoBSD CF card)

If something did go wrong, the only indication, is the termination of the script before an image file is created. Reading the logfile of the last reported step will give more info on the exact reason for failing.

If parts of the build process have already been completed before the process failed, these parts can be reused in the new build by specifying command line options to nanobsd.sh:

`-n` – do not clean directories before building
`-k` – do not build kernel
`-w` – do not build world
`-b` – do not build anything

After the image file is created, it is transferred to the CF card using a (generic USB) card reader. CF cards are typically seen as ATA drives, so the device name will be something like ad1 or ad2. Running

```
# dd if=/usr/obj/nanobsd.NANOBSD/_.disk.full of=/dev/ad1 bs=64k
```

transfers the image file to the CF card. `/dev/ad1` is the CF card here. The created partition image can be mounted directly by running

```
# mdconfig -a -t vnode -f /usr/obj/nanobsd.NANOBSD/_
                    .disk.image -u 1
# mount -t ufs /dev/md1a /mnt
```

The `/mnt` directory now contains the root of the CF card's / partiton. Unmount it by running

```
# umount /mnt
# mdconfig -d -u 1
```

Upgrading / Updating NanoBSD

One of the features of NanoBSD is the offline upgrade and roll-back mechanism, allowing for upgrades to the base system with only seconds of downtime.

The nanobsd.sh script generates two image files. One full disk image and one partition image. In the previous steps, the full disk image was used to fill a CF card. On a running NanoBSD system, there is no need to remove the CF card to perform an upgrade.

The second partition can be upgraded while the system is running from the first partition. When the system is ready to be rebooted (in the maintenance window), booting from the second partition will start the upgraded system. Assuming the first partition is the active partition, run

```
# sh /root/upgradep2 < _.disk.image
```

on the NanoBSD system to upgrade the second partition. (`/root/upgradep1` will upgrade the first partition).

The system will be set to boot from this partition. When the system is ready to be rebooted, reboot.

```
# reboot
```

If the upgrade was unsuccessful, simply set the boot partition back to the first partition and reboot.

The system will revert to the not-yet upgraded partition.

```
# boot0cfg -v -s 1 ad0
# reboot
```

The active partition can also be selected using the [*F1*] and [*F2*] keys during startup.

NanoBSD for ALIX

The following chapters will show my build system, configuration files and caveats.

The hardware I used is the PC Engines ALIX 2D13 board. I had it lying around from my tests with `pfSense`. It has the following features:

Listing 1. *The complete nanobsd.conf*

```
NANO_NAME=ALIX    # directory will be /usr/obj/nanobsd.ALIX
NANO_DRIVE="ad0"  # target drive for the CF card is ATA
NANO_ARCH=i386    # architecture
NANO_KERNEL=ALIX  # kernel file
NANO_BOOTLOADER="boot/boot0"
NANO_BOOT0CFG="-o nopacket -s 1 -m 3"  # ALIX boot options
NANO_MEDIASIZE=1981728  # 1Gb Sandisk CF card
NANO_SECTS=63
NANO_HEADS=32

CONF_WORLD='
TARGET=i386
TARGET_ARCH=i386
TARGET_CPUTYPE=pentium-mmx
# WITHOUT_ options can be inserted here
# examples are WITHOUT_BLUETOOTH, WITHOUT_I4B and WITHOUT_PROFILE
'
# This function enables three tweaks for embedded systems
cust_embedded_setup() {
  # turn off ascii beastie as boot menu
  echo 'autoboot_delay="4"' >> ${NANO_WORLDDIR}/boot/loader.conf
  echo 'beastie_disable="YES"' >> ${NANO_WORLDDIR}/boot/loader.conf

  # turn on noatime for /cfg for more performance
  sed -i "" -e "/cfg/s/rw/rw,noatime/" ${NANO_WORLDDIR}/etc/fstab

  # No "message of the day" for me
  rm ${NANO_WORLDDIR}/etc/motd
  touch ${NANO_WORLDDIR}/etc/motd
}
customize_cmd cust_embedded_setup
# We only have a serial port for console
customize_cmd cust_comconsole
# We allow root to ssh directly
customize_cmd cust_allow_ssh_root
# Install files in /usr/src/tools/tools/nanobsd/Files
customize_cmd cust_install_files
# Install packages in /usr/src/tools/tools/nanobsd/Pkg
customize_cmd cust_pkg
```

- AMD Geode LX800 + glxsb hardware crypto
- 256 MB RAM
- 3 x VIA Rhine network interface (vr)
- IDE CF card slot (master)
- IDE 44-pin interface (slave)
- RTC / USB / I2C / Serial ports

An ideal board for a dedicated firewall/VPN appliance and so much more. My build system is a virtual machine on my laptop running FreeBSD 8.2 amd64 on scsi disks. This means we will have to cross compile, as the ALIX board has an i386 architecture and an IDE disk interface.

NanoBSD config

The most important configuration aspects for the ALIX platform are the target architecture (i386), the target drive (ad0) and the bootoptions (-o nopacket).

We also have to specify the size of the CF card in sectors. This can be tricky, as the values of the sectors and heads of the CF card are often reported incorrectly by the various system tools. We start with the number of blocks. Running

```
# diskinfo /dev/ad0
/dev/ad0    512    1014644736    1981728    967    64    32
```

will give a `NANO_MEDIASIZE` of 1014644736 / 512 = 1981728 blocks. (diskinfo sees 967 cylinders, 64 sectors and 32 heads.) The safes way to fill the `NANO_SECTS` and `NANO_HEADS` is to put the CF card in the ALIX board and boot from it (see the booting section below). It will report

```
PC Engines ALIX.2 v0.99h
640 KB Base Memory
261120 KB Extended Memory

01F0 Master 044A CF 1GB
Phys C/H/S 1966/16/63 Log C/H/S 983/32/63
```

So the system thinks the card has 983 logical cylinders, 32 heads and 63 sectors. That's what we have to work with. The complete nanobsd.conf looks like this: see Listing 1.

The `cust_embedded_setup` function will turn off the beastie ascii art, set the boot delay to 2 seconds and remove the motd (settings I like on my headless platforms).

Kernel config

A GENERIC kernel works fine for ALIX boards, but we can tune the configuration for a leaner kernel with hardware crypto enabled. The processor is i586 (pentium-mmx) compatible

Special thanks

- Poul-Henning Kamp for giving us NanoBSD. He doesn't like personal pages, but this is his *real* one: *http://people.freebsd.org/~phk/*
- pfSense for getting me interested in NanoBSD and for providing all my firewall needs: *http://www.pfsense.org*
- Paul Schenkeveld for performing excellent work on the use of NanoBSD and extending the use to regular production servers with ZFS and Jails. *http://www.psconsult.nl/talks/NLLGG-BSDdag-Servers/*
- PC-Engines for their nifty boards: *http://www.pcengines.ch/*


```
cpu       I586_CPU
ident     ALIX
options   CPU_GEODE
```

The network interface is a VIA Rhine, so only the vr and miibus devices are needed.

```
device  miibus  # MII bus support
device  vr      # VIA Rhine, Rhine II
```

Memory disks are an essential part of NanoBSD.

```
device  md  # Memory „disks"
```

The Geode processor has a hardware crypto module, so we need to enable the glxsb, crypto and cryptodev devices.

```
device  crypto     # core crypto support
device  cryptodev  # /dev/crypto for access to h/w
device  glxsb      # AMD Geode LX Security Block
```

Booting for the first time

After creating the CF card an inserting it into the board, connect a serial cable and start a terminal program. I like to use the screen command for this

```
# screen /dev/tty.PL2303 38400
```

(Yes, I use a Prolific serial-to-usb converter here.)

The ALIX boards run 38400,8,n,1 out of the box, so the terminal program has to work at this baud rate. The first thing I do is set it to 9600 baud by hitting the `s` key during the memory test and pressing the `9` key for 9600 baud. Save the config and restart the terminal program

```
# screen /dev/tty.PL2303 9600
```

ALIX boots and we are presented with a choice. [*F1*] for FreeBSD or [*F2*] for FreeBSD. These are the two partitions on the CF card that both contain the / file system.

This is the moment to chose which partition to boot if an upgrade of a partition completely failed.

Configuring the live system

The configuration of NanoBSD is equal to configuring FreeBSD. You can revert to Bill Harris' article in the previous issue for a quick start.

Because the / file system is read-only and the /etc file system is a memory disk, it is very important to sync the configuration files in `/etc` to `/cfg` after changes. There is a number of scripts in the `/root` directory to help with this synchronisation.

```
change_password  change the root password and sync it to /cfg
save_cfg          sync changed files in /etc to /cfg
save_sshkeys     sync changed ssh keys in /etc/ssh to /cfg/ssh
updatep1          update the first partition with a new
                     partiton image
updatep2          update the second partition with a new
                     partition image
```

Paul Schenkeveld wrote an excellent sync script called cfgsync that will automatically sync all content of `/etc`, including subdirectories (see the special thanks section below).

Where do we go from here?

After playing with embedded systems, I wondered if it was possible to use this concept on my production servers as well. As it turns out, Paul Schenkeveld has had that same idea and extended it with ZFS (see BSD Magazine issue 02/2011) and Jails for a server with near-zero downtime, even for full systems or ports upgrades. He wrote a paper on it and gave talks at AsiaBSDcon2010 and the Dutch NLLGG in December 2010. If you are interested in NanoBSD, I strongly recommend reading his paper.

There are also a number of well-known projects that use NanoBSD as their base. Examples are the pfSense firewall (see BSD Magazine issue 02/2011) and the FreeNAS file server (again, see BSD Magazine issue 02/2011).

ERWIN KOOI

Erwin Kooi is an information security manager for a large grid operator. He started with FreeBSD 4.4 and is an avid fan ever since.