Introduction to NanoBSD

Daniel Gerzo

$FreeBSD: doc/en_US.ISO8859-1/articles/nanobsd/article.sgml,v 1.7 2010/07/13 04:06:37 maxim Exp $

This document provides information about the **NanoBSD** tools, which can be used to create FreeBSD system images for embedded applications, suitable for use on a Compact Flash card (or other mass storage medium).

Table of Contents

1 Introduction to NanoBSD

NanoBSD is a tool currently developed by Poul-Henning Kamp <phk@FreeBSD.org>. It creates a FreeBSD system image for embedded applications, suitable for use on a Compact Flash card (or other mass storage medium).

It can be used to build specialized install images, designed for easy installation and maintenance of systems commonly called "computer appliances". Computer appliances have their hardware and software bundled in the product, which means all applications are pre-installed. The appliance is plugged into an existing network and can begin working (almost) immediately.

The features of **NanoBSD** include:

- Ports and packages work as in FreeBSD — Every single application can be installed and used in a **NanoBSD** image, the same way as in FreeBSD.
- No missing functionality — If it is possible to do something with FreeBSD, it is possible to do the same thing with **NanoBSD**, unless the specific feature or features were explicitly removed from the **NanoBSD** image when it was created.
- Everything is read-only at run-time — It is safe to pull the power-plug. There is no necessity to run fsck(8) after a non-graceful shutdown of the system.

- Easy to build and customize — Making use of just one shell script and one configuration file it is possible to build reduced and customized images satisfying any arbitrary set of requirements.

2 NanoBSD Howto

2.1 The design of NanoBSD

Once the image is present on the medium, it is possible to boot **NanoBSD**. The mass storage medium is divided into three parts by default:

- Two image partitions: `code#1` and `code#2`.
- The configuration file partition, which can be mounted under the `/cfg` directory at run time.

These partitions are normally mounted read-only.

The `/etc` and `/var` directories are md(4) (malloc) disks.

The configuration file partition persists under the `/cfg` directory. It contains files for `/etc` directory and is briefly mounted read-only right after the system boot, therefore it is required to copy modified files from `/etc` back to the `/cfg` directory if changes are expected to persist after the system restarts.

Example 1. Making persistent changes to `/etc/resolv.conf`

```
# vi /etc/resolv.conf
[...]
# mount /cfg
# cp /etc/resolv.conf /cfg
# umount /cfg
```

> **Note:** The partition containing `/cfg` should be mounted only at boot time and while overriding the configuration files.
>
> Keeping `/cfg` mounted at all times is not a good idea, especially if the **NanoBSD** system runs off a mass storage medium that may be adversely affected by a large number of writes to the partition (i.e. when the filesystem syncer flushes data to the system disks).

2.2 Building a NanoBSD image

A **NanoBSD** image is built using a simple `nanobsd.sh` shell script, which can be found in the */usr*`/src/tools/tools/nanobsd` directory. This script creates an image, which can be copied on the storage medium using the dd(1) utility.

The necessary commands to build a **NanoBSD** image are:

```
# cd /usr/src/tools/tools/nanobsd ❶
# sh nanobsd.sh ❷
# cd /usr/obj/nanobsd.full ❸
# dd if=_.disk.full of=/dev/da0 bs=64k ❹
```

❶ Change the current directory to the base directory of the **NanoBSD** build script.

❷ Start the build process.

❸ Change the current directory to the place where the built images are located.

❹ Install **NanoBSD** onto the storage medium.

2.3 Customizing a NanoBSD image

This is probably the most important and most interesting feature of **NanoBSD**. This is also where you will be spending most of the time when developing with **NanoBSD**.

Invocation of the following command will force the `nanobsd.sh` to read its configuration from the `myconf.nano` file located in the current directory:

```
# sh nanobsd.sh -c myconf.nano
```

Customization is done in two ways:

- Configuration options
- Custom functions

2.3.1 Configuration options

With configuration settings, it is possible to configure options passed to both the `buildworld` and `installworld` stages of the **NanoBSD** build process, as well as internal options passed to the main build process of **NanoBSD**. Through these options it is possible to cut the system down, so it will fit on as little as 64MB. You can use the configuration options to trim down FreeBSD even more, until it will consists of just the kernel and two or three files in the userland.

The configuration file consists of configuration options, which override the default values. The most important directives are:

- `NANO_NAME` — Name of build (used to construct the workdir names).
- `NANO_SRC` — Path to the source tree used to build the image.
- `NANO_KERNEL` — Name of kernel configuration file used to build kernel.
- `CONF_BUILD` — Options passed to the `buildworld` stage of the build.
- `CONF_INSTALL` — Options passed to the `installworld` stage of the build.
- `CONF_WORLD` — Options passed to both the `buildworld` and the `installworld` stage of the build.
- `FlashDevice` — Defines what type of media to use. Check the `FlashDevice.sub` file for more details.

2.3.2 Custom functions

It is possible to fine-tune **NanoBSD** using shell functions in the configuration file. The following example illustrates the basic model of custom functions:

```
cust_foo () (
```

```
 echo "bar=baz" > \
  ${NANO_WORLDDIR}/etc/foo
)
customize_cmd cust_foo
```

A more useful example of a customization function is the following, which changes the default size of the `/etc` directory from 5MB to 30MB:

```
cust_etc_size () (
 cd ${NANO_WORLDDIR}/conf
 echo 30000 > default/etc/md_size
)
customize_cmd cust_etc_size
```

There are a few default pre-defined customization functions ready for use:

- `cust_comconsole` — Disables getty(8) on the VGA devices (the `/dev/ttyv*` device nodes) and enables the use of the COM1 serial port as the system console.
- `cust_allow_ssh_root` — Allow `root` to login via sshd(8).
- `cust_install_files` — Installs files from the `nanobsd/Files` directory, which contains some useful scripts for system administration.

2.3.3 Adding packages

Packages can be added to a **NanoBSD** image using a custom function. The following function will install all the packages located in `/usr/src/tools/tools/nanobsd/packages`:

```
install_packages () (
mkdir -p ${NANO_WORLDDIR}/packages
cp /usr/src/tools/tools/nanobsd/packages/* ${NANO_WORLDDIR}/packages
chroot ${NANO_WORLDDIR} sh -c 'cd packages; pkg_add -v *;cd ..;'
rm -rf ${NANO_WORLDDIR}/packages
)
customize_cmd install_packages
```

2.3.4 Configuration file example

A complete example of a configuration file for building a custom **NanoBSD** image can be:

```
NANO_NAME=custom
NANO_SRC=/usr/src
NANO_KERNEL=MYKERNEL
NANO_IMAGES=2

CONF_BUILD='
NO_KLDLOAD=YES
NO_NETGRAPH=YES
NO_PAM=YES
'
```

```
CONF_INSTALL='
NO_ACPI=YES
NO_BLUETOOTH=YES
NO_CVS=YES
NO_FORTRAN=YES
NO_HTML=YES
NO_LPR=YES
NO_MAN=YES
NO_SENDMAIL=YES
NO_SHAREDOCS=YES
NO_EXAMPLES=YES
NO_INSTALLLIB=YES
NO_CALENDAR=YES
NO_MISC=YES
NO_SHARE=YES
'

CONF_WORLD='
NO_BIND=YES
NO_MODULES=YES
NO_KERBEROS=YES
NO_GAMES=YES
NO_RESCUE=YES
NO_LOCALES=YES
NO_SYSCONS=YES
NO_INFO=YES
'

FlashDevice SanDisk 1G

cust_nobeastie() (
 touch ${NANO_WORLDDIR}/boot/loader.conf
 echo "beastie_disable=\"YES\"" >> ${NANO_WORLDDIR}/boot/loader.conf
)

customize_cmd cust_comconsole
customize_cmd cust_install_files
customize_cmd cust_allow_ssh_root
customize_cmd cust_nobeastie
```

2.4 Updating NanoBSD

The update process of **NanoBSD** is relatively simple:

1. Build a new **NanoBSD** image, as usual.
2. Upload the new image into an unused partition of a running **NanoBSD** appliance.

   The most important difference of this step from the initial **NanoBSD** installation is that now instead of using the `_.disk.full` file (which contains an image of the entire disk), the `_.disk.image` image is installed (which contains an image of a single system partition).

3. Reboot, and start the system from the newly installed partition.
4. If all goes well, the upgrade is finished.
5. If anything goes wrong, reboot back into the previous partition (which contains the old, working image), to restore system functionality as fast as possible. Fix any problems of the new build, and repeat the process.

To install new image onto the running **NanoBSD** system, it is possible to use either the `updatep1` or `updatep2` script located in the `/root` directory, depending from which partition is running the current system.

According to which services are available on host serving new **NanoBSD** image and what type of transfer is preferred, it is possible to examine one of these three ways:

2.4.1 Using ftp(1)

If the transfer speed is in first place, use this example:

```
# ftp myhost
get _.disk.image "| sh updatep1"
```

2.4.2 Using ssh(1)

If a secure transfer is preferred, consider using this example:

```
# ssh myhost cat _.disk.image.gz | zcat | sh updatep1
```

2.4.3 Using nc(1)

Try this example if the remote host is not running neither ftp(1) or sshd(8) service:

1. At first, open a TCP listener on host serving the image and make it send the image to client:

   ```
   myhost# nc -l 2222 < _.disk.image
   ```

   > **Note:** Make sure that the used port is not blocked to receive incoming connections from **NanoBSD** host by firewall.

2. Connect to the host serving new image and execute `updatep1` script:

   ```
   # nc myhost 2222 | sh updatep1
   ```