# FreeBSD and Solid State Devices

Abstract

This article covers the use of solid state disk devices in FreeBSD to create embedded systems.

Embedded systems have the advantage of increased stability due to the lack of integral moving parts (hard drives). Account must be taken, however, for the generally low disk space available in the system and the durability of the storage medium.

Specific topics to be covered include the types and attributes of solid state media suitable for disk use in FreeBSD, kernel options that are of interest in such an environment, the rc.initdiskless mechanisms that automate the initialization of such systems and the need for read-only filesystems, and building filesystems from scratch. The article will conclude with some general strategies for small and read-only FreeBSD environments.

---

# Table of Contents

1. Solid State Disk Devices . . . . . 1
2. Kernel Options . . . . . 2
3. The `rc` Subsystem and Read-Only Filesystems . . . . . 2
4. Building a File System from Scratch . . . . . 3
5. System Strategies for Small and Read Only Environments . . . . . 5

# 1. Solid State Disk Devices

The scope of this article will be limited to solid state disk devices made from flash memory. Flash memory is a solid state memory (no moving parts) that is non-volatile (the memory maintains data even after all power sources have been disconnected). Flash memory can withstand tremendous physical shock and is reasonably fast (the flash memory solutions covered in this article are slightly slower than an EIDE hard disk for write operations, and much faster for read operations). One very important aspect of flash memory, the ramifications of which will be discussed later in this article, is that each sector has a limited rewrite capacity. You can only write, erase, and write again to a sector of flash memory a certain number of times before the sector becomes permanently unusable. Although many flash memory products automatically map bad blocks, and although some even distribute write operations evenly throughout the unit, the fact remains that there exists a limit to the amount of writing that can be done to the device. Competitive units have between 1,000,000 and 10,000,000 writes per sector in their specification. This figure varies due to the temperature of the environment.

Specifically, we will be discussing ATA compatible compact-flash units, which are quite popular as storage media for digital cameras. Of particular interest is the fact that they pin out directly to the IDE bus and are compatible with the ATA command set. Therefore, with a very simple and low-cost

adaptor, these devices can be attached directly to an IDE bus in a computer. Once implemented in this manner, operating systems such as FreeBSD see the device as a normal hard disk (albeit small).

Other solid state disk solutions do exist, but their expense, obscurity, and relative unease of use places them beyond the scope of this article.

# 2. Kernel Options

A few kernel options are of specific interest to those creating an embedded FreeBSD system.

All embedded FreeBSD systems that use flash memory as system disk will be interested in memory disks and memory filesystems. As a result of the limited number of writes that can be done to flash memory, the disk and the filesystems on the disk will most likely be mounted read-only. In this environment, filesystems such as /tmp and /var are mounted as memory filesystems to allow the system to create logs and update counters and temporary files. Memory filesystems are a critical component to a successful solid state FreeBSD implementation.

You should make sure the following lines exist in your kernel configuration file:

```
options         MD_ROOT         # md device usable as a potential root device
```

# 3. The `rc` Subsystem and Read-Only Filesystems

The post-boot initialization of an embedded FreeBSD system is controlled by /etc/rc.initdiskless.

/etc/rc.d/var mounts /var as a memory filesystem, makes a configurable list of directories in /var with the mkdir(1) command, and changes modes on some of those directories. In the execution of /etc/rc.d/var, one other rc.conf variable comes into play - `varsize`. A /var partition is created by /etc/rc.d/var based on the value of this variable in rc.conf:

```
varsize=8192
```

Remember that this value is in sectors by default.

The fact that /var is a read-write filesystem is an important distinction, as the / partition (and any other partitions you may have on your flash media) should be mounted read-only. Remember that in Solid State Disk Devices we detailed the limitations of flash memory - specifically the limited write capability. The importance of not mounting filesystems on flash media read-write, and the importance of not using a swap file, cannot be overstated. A swap file on a busy system can burn through a piece of flash media in less than one year. Heavy logging or temporary file creation and destruction can do the same. Therefore, in addition to removing the `swap` entry from your /etc/fstab, you should also change the Options field for each filesystem to `ro` as follows:

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
/dev/ad0s1a             /               ufs     ro              1       1
```

A few applications in the average system will immediately begin to fail as a result of this change. For instance, cron will not run properly as a result of missing cron tabs in the /var created by /etc/rc.d/var, and syslog and dhcp will encounter problems as well as a result of the read-only filesystem and missing items in the /var that /etc/rc.d/var has created. These are only temporary problems though, and are addressed, along with solutions to the execution of other common software packages in System Strategies for Small and Read Only Environments.

An important thing to remember is that a filesystem that was mounted read-only with /etc/fstab can be made read-write at any time by issuing the command:

```
# /sbin/mount -uw partition
```

and can be toggled back to read-only with the command:

```
# /sbin/mount -ur partition
```

# 4. Building a File System from Scratch

Since ATA compatible compact-flash cards are seen by FreeBSD as normal IDE hard drives, you could theoretically install FreeBSD from the network using the kern and mfsroot floppies or from a CD.

However, even a small installation of FreeBSD using normal installation procedures can produce a system in size of greater than 200 megabytes. Most people will be using smaller flash memory devices (128 megabytes is considered fairly large - 32 or even 16 megabytes is common), so an installation using normal mechanisms is not possible-there is simply not enough disk space for even the smallest of conventional installations.

The easiest way to overcome this space limitation is to install FreeBSD using conventional means to a normal hard disk. After the installation is complete, pare down the operating system to a size that will fit onto your flash media, then tar the entire filesystem. The following steps will guide you through the process of preparing a piece of flash memory for your tarred filesystem. Remember, because a normal installation is not being performed, operations such as partitioning, labeling, file-system creation, etc. need to be performed by hand. In addition to the kern and mfsroot floppy disks, you will also need to use the fixit floppy.

1. Partitioning Your Flash Media Device

   After booting with the kern and mfsroot floppies, choose `custom` from the installation menu. In the custom installation menu, choose `partition`. In the partition menu, you should delete all existing partitions using `d`. After deleting all existing partitions, create a partition using `c`

and accept the default value for the size of the partition. When asked for the type of the partition, make sure the value is set to 165. Now write this partition table to the disk by pressing `w` (this is a hidden option on this screen). If you are using an ATA compatible compact flash card, you should choose the FreeBSD Boot Manager. Now press `q` to quit the partition menu. You will be shown the boot manager menu once more - repeat the choice you made earlier.

2. Creating Filesystems on Your Flash Memory Device

   Exit the custom installation menu, and from the main installation menu choose the `fixit` option. After entering the fixit environment, enter the following command:

   ```
   # disklabel -e /dev/ad0c
   ```

   At this point you will have entered the vi editor under the auspices of the disklabel command. Next, you need to add an `a:` line at the end of the file. This `a:` line should look like:

   ```
   a:      123456  0       4.2BSD  0       0
   ```

   Where *123456* is a number that is exactly the same as the number in the existing `c:` entry for size. Basically you are duplicating the existing `c:` line as an `a:` line, making sure that fstype is `4.2BSD`. Save the file and exit.

   ```
   # disklabel -B -r /dev/ad0c
   # newfs /dev/ad0a
   ```

3. Placing Your Filesystem on the Flash Media

   Mount the newly prepared flash media:

   ```
   # mount /dev/ad0a /flash
   ```

   Bring this machine up on the network so we may transfer our tar file and explode it onto our flash media filesystem. One example of how to do this is:

   ```
   # ifconfig xl0 192.168.0.10 netmask 255.255.255.0
   # route add default 192.168.0.1
   ```

   Now that the machine is on the network, transfer your tar file. You may be faced with a bit of a dilemma at this point - if your flash memory part is 128 megabytes, for instance, and your tar file is larger than 64 megabytes, you cannot have your tar file on the flash media at the same time as you explode it - you will run out of space. One solution to this problem, if you are using FTP, is to untar the file while it is transferred over FTP. If you perform your

transfer in this manner, you will never have the tar file and the tar contents on your disk at the same time:

```
ftp> get tarfile.tar "| tar xvf -"
```

If your tarfile is gzipped, you can accomplish this as well:

```
ftp> get tarfile.tar "| zcat | tar xvf -"
```

After the contents of your tarred filesystem are on your flash memory filesystem, you can unmount the flash memory and reboot:

```
# cd /
# umount /flash
# exit
```

Assuming that you configured your filesystem correctly when it was built on the normal hard disk (with your filesystems mounted read-only, and with the necessary options compiled into the kernel) you should now be successfully booting your FreeBSD embedded system.

# 5. System Strategies for Small and Read Only Environments

In The `rc` Subsystem and Read-Only Filesystems, it was pointed out that the /var filesystem constructed by /etc/rc.d/var and the presence of a read-only root filesystem causes problems with many common software packages used with FreeBSD. In this article, suggestions for successfully running cron, syslog, ports installations, and the Apache web server will be provided.

## 5.1. Cron

Upon boot, /var gets populated by /etc/rc.d/var using the list from /etc/mtree/BSD.var.dist, so the cron, cron/tabs, at, and a few other standard directories get created.

However, this does not solve the problem of maintaining cron tabs across reboots. When the system reboots, the /var filesystem that is in memory will disappear and any cron tabs you may have had in it will also disappear. Therefore, one solution would be to create cron tabs for the users that need them, mount your / filesystem as read-write and copy those cron tabs to somewhere safe, like /etc/tabs, then add a line to the end of /etc/rc.initdiskless that copies those crontabs into /var/cron/tabs after that directory has been created during system initialization. You may also need to add a line that changes modes and permissions on the directories you create and the files you copy with /etc/rc.initdiskless.

## 5.2. Syslog

syslog.conf specifies the locations of certain log files that exist in /var/log. These files are not created by /etc/rc.d/var upon system initialization. Therefore, somewhere in /etc/rc.d/var, after the section that creates the directories in /var, you will need to add something like this:

```
# touch /var/log/security /var/log/maillog /var/log/cron /var/log/messages
# chmod 0644 /var/log/*
```

## 5.3. Ports Installation

Before discussing the changes necessary to successfully use the ports tree, a reminder is necessary regarding the read-only nature of your filesystems on the flash media. Since they are read-only, you will need to temporarily mount them read-write using the mount syntax shown in The `rc` Subsystem and Read-Only Filesystems. You should always remount those filesystems read-only when you are done with any maintenance - unnecessary writes to the flash media could considerably shorten its lifespan.

To make it possible to enter a ports directory and successfully run `make install`, we must create a packages directory on a non-memory filesystem that will keep track of our packages across reboots. As it is necessary to mount your filesystems as read-write for the installation of a package anyway, it is sensible to assume that an area on the flash media can also be used for package information to be written to.

First, create a package database directory. This is normally in /var/db/pkg, but we cannot place it there as it will disappear every time the system is booted.

```
# mkdir /etc/pkg
```

Now, add a line to /etc/rc.d/var that links the /etc/pkg directory to /var/db/pkg. An example:

```
# ln -s /etc/pkg /var/db/pkg
```

Now, any time that you mount your filesystems as read-write and install a package, the `make install` will work, and package information will be written successfully to /etc/pkg (because the filesystem will, at that time, be mounted read-write) which will always be available to the operating system as /var/db/pkg.

## 5.4. Apache Web Server

The steps in this section are only necessary if Apache is set up to write its pid or log information outside of /var. By default, Apache keeps its pid file in /var/run/httpd.pid and its log files in /var/log.

It is now assumed that Apache keeps its log files in a directory apache_log_dir outside of /var. When this directory lives on a read-only filesystem, Apache will not be able to save any log files, and may have problems working. If so, it is necessary to add a new directory to the list of directories in /etc/rc.d/var to create in /var, and to link apache_log_dir to /var/log/apache. It is also necessary to set permissions and ownership on this new directory.

First, add the directory `log/apache` to the list of directories to be created in /etc/rc.d/var.

Second, add these commands to /etc/rc.d/var after the directory creation section:

```
# chmod 0774 /var/log/apache
# chown nobody:nobody /var/log/apache
```

Finally, remove the existing apache_log_dir directory, and replace it with a link:

```
# rm -rf apache_log_dir
# ln -s /var/log/apache apache_log_dir
```