NAME

vinum – Logical Volume Manager

SYNOPSIS

kldload vinum

DESCRIPTION

vinum is a logical volume manager inspired by, but not derived from, the Veritas Volume Manager. It provides the following features:

- It provides device-independent logical disks, called *volumes*. Volumes are not restricted to the size of any disk on the system.
- The volumes consist of one or more *plexes*, each of which contain the entire address space of a volume. This represents an implementation of RAID-1 (mirroring). Multiple plexes can also be used for:
  - Increased read throughput. **vinum** will read data from the least active disk, so if a volume has plexes on multiple disks, more data can be read in parallel. **vinum** reads data from only one plex, but it writes data to all plexes.
  - Increased reliability. By storing plexes on different disks, data will remain available even if one of the plexes becomes unavailable. In comparison with a RAID-5 plex (see below), using multiple plexes requires more storage space, but gives better performance, particularly in the case of a drive failure.
  - Additional plexes can be used for on-line data reorganization. By attaching an additional plex and subsequently detaching one of the older plexes, data can be moved on-line without compromising access.
  - An additional plex can be used to obtain a consistent dump of a filesystem. By attaching an additional plex and detaching at a specific time, the detached plex becomes an accurate snapshot of the filesystem at the time of detachment.
- Each plex consists of one or more logical disk slices, called *subdisks*. Subdisks are defined as a contiguous block of physical disk storage. A plex may consist of any reasonable number of subdisks (in other words, the real limit is not the number, but other factors, such as memory and performance, associated with maintaining a large number of subdisks).
- A number of mappings between subdisks and plexes are available:
  - *Concatenated plexes* consist of one or more subdisks, each of which is mapped to a contiguous part of the plex address space.
  - *Striped plexes* consist of two or more subdisks of equal size. The file address space is mapped in *stripes*, integral fractions of the subdisk size. Consecutive plex address space is mapped to stripes in each subdisk in turn. The subdisks of a striped plex must all be the same size.
  - *RAID-5 plexes* require at least three equal-sized subdisks. They resemble striped plexes, except that in each stripe, one subdisk stores parity information. This subdisk changes in each stripe: in the first stripe, it is the first subdisk, in the second it is the second subdisk, etc. In the event of a single disk failure, **vinum** will recover the data based on the information stored on the remaining subdisks. This mapping is particularly suited to read-intensive access. The subdisks of a RAID-5 plex must all be the same size.
- *Drives* are the lowest level of the storage hierarchy. They represent disk special devices.

- **vinum** offers automatic startup. Unlike UNIX filesystems, **vinum** volumes contain all the configuration information needed to ensure that they are started correctly when the subsystem is enabled. This is also a significant advantage over the Veritas™ File System. This feature regards the presence of the volumes. It does not mean that the volumes will be mounted automatically, since the standard startup procedures with `/etc/fstab` perform this function.

KERNEL CONFIGURATION

vinum is currently supplied as a loadable kernel module (LKM), and does not require configuration. As with other LKMs, it is absolutely necessary to match the LKM to the version of the operating system. Failure to do so will cause **vinum** to issue an error message and terminate.

It is possible to configure **vinum** in the kernel, but this is not recommended. To do so, add this line to the kernel configuration file:

device vinum

Debug Options

The current version of **vinum**, both the kernel module and the user program `vinum`(8), include significant debugging support. It is not recommended to remove this support at the moment, but if you do you must remove it from both the kernel and the user components. To do this, edit the files `/usr/src/sbin/vinum/Makefile` and `/usr/src/sys/modules/vinum/Makefile` and edit the *CFLAGS* variable to remove the `-DVINUMDEBUG` option. If you have configured **vinum** into the kernel, either specify the line

options VINUMDEBUG

in the kernel configuration file or remove the `-DVINUMDEBUG` option from `/usr/src/sbin/vinum/Makefile` as described above.

If the *VINUMDEBUG* variables do not match, `vinum`(8) will fail with a message explaining the problem and what to do to correct it.

vinum was previously available in two versions: a freely available version which did not contain RAID-5 functionality, and a full version including RAID-5 functionality, which was available only from Cybernet Systems Inc. The present version of **vinum** includes the RAID-5 functionality.

RUNNING VINUM

vinum is part of the base FreeBSD system. It does not require installation. To start it, start the `vinum`(8) program, which will load the LKM if it is not already present. Before using **vinum**, it must be configured. See `vinum`(8) for information on how to create a **vinum** configuration.

Normally, you start a configured version of **vinum** at boot time. Set the variable *start_vinum* in `/etc/rc.conf` to “`YES`” to start **vinum** at boot time. (See `rc.conf`(5) for more details.)

If **vinum** is loaded as a LKM (the recommended way), the **vinum stop** command will unload it (see `vinum`(8)). You can also do this with the `kldunload`(8) command.

The LKM can only be unloaded when idle, in other words when no volumes are mounted and no other instances of the `vinum`(8) program are active. Unloading the LKM does not harm the data in the volumes.

Configuring and Starting Objects

Use the `vinum`(8) utility to configure and start **vinum** objects.

IOCTL CALLS

ioctl(2) calls are intended for the use of the vinum(8) configuration program only. They are described in the header file /sys/dev/vinum/vinumio.h.

Disk Labels

Conventional disk special devices have a *disk label* in the second sector of the device. See disklabel(5) for more details. This disk label describes the layout of the partitions within the device. **vinum** does not subdivide volumes, so volumes do not contain a physical disk label. For convenience, **vinum** implements the ioctl calls DIOCGDINFO (get disk label), DIOCGPART (get partition information), DIOCWDINFO (write partition information) and DIOCSDINFO (set partition information). DIOCGDINFO and DIOCGPART refer to an internal representation of the disk label which is not present on the volume. As a result, the **−r** option of disklabel(8), which reads the "raw disk", will fail.

In general, disklabel(8) serves no useful purpose on a **vinum** volume. If you run it, it will show you three partitions, 'a', 'b' and 'c', all the same except for the *fstype*, for example:

```
3 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  a:     2048        0    4.2BSD     1024  8192      0   # (Cyl.    0 - 0)
  b:     2048        0      swap                         # (Cyl.    0 - 0)
  c:     2048        0    unused        0     0          # (Cyl.    0 - 0)
```

vinum ignores the DIOCWDINFO and DIOCSDINFO ioctls, since there is nothing to change. As a result, any attempt to modify the disk label will be silently ignored.

MAKING FILE SYSTEMS

Since **vinum** volumes do not contain partitions, the names do not need to conform to the standard rules for naming disk partitions. For a physical disk partition, the last letter of the device name specifies the partition identifier (a to h). **vinum** volumes need not conform to this convention, but if they do not, newfs(8) will complain that it cannot determine the partition. To solve this problem, use the **−v** flag to newfs(8). For example, if you have a volume concat, use the following command to create a UFS filesystem on it:

```
newfs -v /dev/vinum/concat
```

OBJECT NAMING

vinum assigns default names to plexes and subdisks, although they may be overridden. We do not recommend overriding the default names. Experience with the Veritas™ volume manager, which allows arbitrary naming of objects, has shown that this flexibility does not bring a significant advantage, and it can cause confusion.

Names may contain any non-blank character, but it is recommended to restrict them to letters, digits and the underscore characters. The names of volumes, plexes and subdisks may be up to 64 characters long, and the names of drives may up to 32 characters long. When choosing volume and plex names, bear in mind that automatically generated plex and subdisk names are longer than the name from which they are derived.

- When **vinum** creates or deletes objects, it creates a directory /dev/vinum, in which it makes device entries for each volume it finds. It also creates subdirectories, /dev/vinum/plex and /dev/vinum/sd, in which it stores device entries for plexes and subdisks. In addition, it creates two more directories, /dev/vinum/vol and /dev/vinum/drive, in which it stores hierarchical information for volumes and drives.
- In addition, **vinum** creates three super-devices, /dev/vinum/control, /dev/vinum/Control and /dev/vinum/controld. /dev/vinum/control is used by vinum(8) when it has been compiled without the VINUMDEBUG option, /dev/vinum/Control is used by vinum(8) when it has been compiled with the VINUMDEBUG option, and /dev/vinum/controld is used by the

vinum daemon. The two control devices for vinum(8) are used to synchronize the debug status of kernel and user modules.

- Unlike UNIX drives, **vinum** volumes are not subdivided into partitions, and thus do not contain a disk label. Unfortunately, this confuses a number of utilities, notably newfs(8), which normally tries to interpret the last letter of a **vinum** volume name as a partition identifier. If you use a volume name which does not end in the letters 'a' to 'c', you must use the **-v** flag to newfs(8) in order to tell it to ignore this convention.
- Plexes do not need to be assigned explicit names. By default, a plex name is the name of the volume followed by the letters .p and the number of the plex. For example, the plexes of volume vol3 are called vol3.p0, vol3.p1 and so on. These names can be overridden, but it is not recommended.
- Like plexes, subdisks are assigned names automatically, and explicit naming is discouraged. A subdisk name is the name of the plex followed by the letters .s and a number identifying the subdisk. For example, the subdisks of plex vol3.p0 are called vol3.p0.s0, vol3.p0.s1 and so on.
- By contrast, *drives* must be named. This makes it possible to move a drive to a different location and still recognize it automatically. Drive names may be up to 32 characters long.

Example

Assume the **vinum** objects described in the section **CONFIGURATION FILE** in vinum(8). The directory /dev/vinum looks like:

```
# ls -lR /dev/vinum
total 5
brwxr-xr--  1 root  wheel   25,   2 Mar 30 16:08 concat
brwx------  1 root  wheel   25, 0x40000000 Mar 30 16:08 control
brwx------  1 root  wheel   25, 0x40000001 Mar 30 16:08 controld
drwxrwxrwx  2 root  wheel       512 Mar 30 16:08 drive
drwxrwxrwx  2 root  wheel       512 Mar 30 16:08 plex
drwxrwxrwx  2 root  wheel       512 Mar 30 16:08 rvol
drwxrwxrwx  2 root  wheel       512 Mar 30 16:08 sd
brwxr-xr--  1 root  wheel   25,   3 Mar 30 16:08 strcon
brwxr-xr--  1 root  wheel   25,   1 Mar 30 16:08 stripe
brwxr-xr--  1 root  wheel   25,   0 Mar 30 16:08 tinyvol
drwxrwxrwx  7 root  wheel       512 Mar 30 16:08 vol
brwxr-xr--  1 root  wheel   25,   4 Mar 30 16:08 vol5

/dev/vinum/drive:
total 0
brw-r-----  1 root  operator    4,  15 Oct 21 16:51 drive2
brw-r-----  1 root  operator    4,  31 Oct 21 16:51 drive4

/dev/vinum/plex:
total 0
brwxr-xr--  1 root  wheel   25, 0x10000002 Mar 30 16:08 concat.p0
brwxr-xr--  1 root  wheel   25, 0x10010002 Mar 30 16:08 concat.p1
brwxr-xr--  1 root  wheel   25, 0x10000003 Mar 30 16:08 strcon.p0
brwxr-xr--  1 root  wheel   25, 0x10010003 Mar 30 16:08 strcon.p1
brwxr-xr--  1 root  wheel   25, 0x10000001 Mar 30 16:08 stripe.p0
brwxr-xr--  1 root  wheel   25, 0x10000000 Mar 30 16:08 tinyvol.p0
brwxr-xr--  1 root  wheel   25, 0x10000004 Mar 30 16:08 vol5.p0
brwxr-xr--  1 root  wheel   25, 0x10010004 Mar 30 16:08 vol5.p1
```

```
/dev/vinum/sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20000002 Mar 30 16:08 concat.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100002 Mar 30 16:08 concat.p0.s1
brwxr-xr--  1 root  wheel   25, 0x20010002 Mar 30 16:08 concat.p1.s0
brwxr-xr--  1 root  wheel   25, 0x20000003 Mar 30 16:08 strcon.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100003 Mar 30 16:08 strcon.p0.s1
brwxr-xr--  1 root  wheel   25, 0x20010003 Mar 30 16:08 strcon.p1.s0
brwxr-xr--  1 root  wheel   25, 0x20110003 Mar 30 16:08 strcon.p1.s1
brwxr-xr--  1 root  wheel   25, 0x20000001 Mar 30 16:08 stripe.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100001 Mar 30 16:08 stripe.p0.s1
brwxr-xr--  1 root  wheel   25, 0x20000000 Mar 30 16:08 tinyvol.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100000 Mar 30 16:08 tinyvol.p0.s1
brwxr-xr--  1 root  wheel   25, 0x20000004 Mar 30 16:08 vol5.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100004 Mar 30 16:08 vol5.p0.s1
brwxr-xr--  1 root  wheel   25, 0x20010004 Mar 30 16:08 vol5.p1.s0
brwxr-xr--  1 root  wheel   25, 0x20110004 Mar 30 16:08 vol5.p1.s1

/dev/vinum/vol:
total 5
brwxr-xr--  1 root  wheel   25,   2 Mar 30 16:08 concat
drwxr-xr-x  4 root  wheel       512 Mar 30 16:08 concat.plex
brwxr-xr--  1 root  wheel   25,   3 Mar 30 16:08 strcon
drwxr-xr-x  4 root  wheel       512 Mar 30 16:08 strcon.plex
brwxr-xr--  1 root  wheel   25,   1 Mar 30 16:08 stripe
drwxr-xr-x  3 root  wheel       512 Mar 30 16:08 stripe.plex
brwxr-xr--  1 root  wheel   25,   0 Mar 30 16:08 tinyvol
drwxr-xr-x  3 root  wheel       512 Mar 30 16:08 tinyvol.plex
brwxr-xr--  1 root  wheel   25,   4 Mar 30 16:08 vol5
drwxr-xr-x  4 root  wheel       512 Mar 30 16:08 vol5.plex

/dev/vinum/vol/concat.plex:
total 2
brwxr-xr--  1 root  wheel   25, 0x10000002 Mar 30 16:08 concat.p0
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 concat.p0.sd
brwxr-xr--  1 root  wheel   25, 0x10010002 Mar 30 16:08 concat.p1
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 concat.p1.sd

/dev/vinum/vol/concat.plex/concat.p0.sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20000002 Mar 30 16:08 concat.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100002 Mar 30 16:08 concat.p0.s1

/dev/vinum/vol/concat.plex/concat.p1.sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20010002 Mar 30 16:08 concat.p1.s0

/dev/vinum/vol/strcon.plex:
total 2
brwxr-xr--  1 root  wheel   25, 0x10000003 Mar 30 16:08 strcon.p0
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 strcon.p0.sd
```

```
brwxr-xr--  1 root  wheel   25, 0x10010003 Mar 30 16:08 strcon.p1
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 strcon.p1.sd

/dev/vinum/vol/strcon.plex/strcon.p0.sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20000003 Mar 30 16:08 strcon.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100003 Mar 30 16:08 strcon.p0.s1

/dev/vinum/vol/strcon.plex/strcon.p1.sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20010003 Mar 30 16:08 strcon.p1.s0
brwxr-xr--  1 root  wheel   25, 0x20110003 Mar 30 16:08 strcon.p1.s1

/dev/vinum/vol/stripe.plex:
total 1
brwxr-xr--  1 root  wheel   25, 0x10000001 Mar 30 16:08 stripe.p0
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 stripe.p0.sd

/dev/vinum/vol/stripe.plex/stripe.p0.sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20000001 Mar 30 16:08 stripe.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100001 Mar 30 16:08 stripe.p0.s1

/dev/vinum/vol/tinyvol.plex:
total 1
brwxr-xr--  1 root  wheel   25, 0x10000000 Mar 30 16:08 tinyvol.p0
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 tinyvol.p0.sd

/dev/vinum/vol/tinyvol.plex/tinyvol.p0.sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20000000 Mar 30 16:08 tinyvol.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100000 Mar 30 16:08 tinyvol.p0.s1

/dev/vinum/vol/vol5.plex:
total 2
brwxr-xr--  1 root  wheel   25, 0x10000004 Mar 30 16:08 vol5.p0
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 vol5.p0.sd
brwxr-xr--  1 root  wheel   25, 0x10010004 Mar 30 16:08 vol5.p1
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 vol5.p1.sd

/dev/vinum/vol/vol5.plex/vol5.p0.sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20000004 Mar 30 16:08 vol5.p0.s0
brwxr-xr--  1 root  wheel   25, 0x20100004 Mar 30 16:08 vol5.p0.s1

/dev/vinum/vol/vol5.plex/vol5.p1.sd:
total 0
brwxr-xr--  1 root  wheel   25, 0x20010004 Mar 30 16:08 vol5.p1.s0
brwxr-xr--  1 root  wheel   25, 0x20110004 Mar 30 16:08 vol5.p1.s1
```

In the case of unattached plexes and subdisks, the naming is reversed. Subdisks are named after the disk on which they are located, and plexes are named after the subdisk. **This mapping is still to be determined.**

Object States

Each **vinum** object has a *state* associated with it. **vinum** uses this state to determine the handling of the object.

Volume States

Volumes may have the following states:

down The volume is completely inaccessible.

up The volume is up and at least partially functional. Not all plexes may be available.

Plex States

Plexes may have the following states:

referenced A plex entry which has been referenced as part of a volume, but which is currently not known.

faulty A plex which has gone completely down because of I/O errors.

down A plex which has been taken down by the administrator.

initializing A plex which is being initialized.

The remaining states represent plexes which are at least partially up.

corrupt A plex entry which is at least partially up. Not all subdisks are available, and an inconsistency has occurred. If no other plex is uncorrupted, the volume is no longer consistent.

degraded A RAID-5 plex entry which is accessible, but one subdisk is down, requiring recovery for many I/O requests.

flaky A plex which is really up, but which has a reborn subdisk which we do not completely trust, and which we do not want to read if we can avoid it.

up A plex entry which is completely up. All subdisks are up.

Subdisk States

Subdisks can have the following states:

empty A subdisk entry which has been created completely. All fields are correct, and the disk has been updated, but the on the disk is not valid.

referenced A subdisk entry which has been referenced as part of a plex, but which is currently not known.

initializing A subdisk entry which has been created completely and which is currently being initialized.

The following states represent invalid data.

obsolete A subdisk entry which has been created completely. All fields are correct, the config on disk has been updated, and the data was valid, but since then the drive has been taken down, and as a result updates have been missed.

stale A subdisk entry which has been created completely. All fields are correct, the disk has been updated, and the data was valid, but since then the drive has been crashed and updates have been lost.

The following states represent valid, inaccessible data.

crashed A subdisk entry which has been created completely. All fields are correct, the disk has been updated, and the data was valid, but since then the drive has gone down. No attempt has been made to write to the subdisk since the crash, so the data is valid.

down A subdisk entry which was up, which contained valid data, and which was taken down by the administrator. The data is valid.

reviving The subdisk is currently in the process of being revived. We can write but not read.

The following states represent accessible subdisks with valid data.

reborn A subdisk entry which has been created completely. All fields are correct, the disk has been updated, and the data was valid, but since then the drive has gone down and up again. No updates were lost, but it is possible that the subdisk has been damaged. We will not read from this subdisk if we have a choice. If this is the only subdisk which covers this address space in the plex, we set its state to up under these circumstances, so this status implies that there is another subdisk to fulfill the request.

up A subdisk entry which has been created completely. All fields are correct, the disk has been updated, and the data is valid.

Drive States

Drives can have the following states:

referenced At least one subdisk refers to the drive, but it is not currently accessible to the system. No device name is known.

down The drive is not accessible.

up The drive is up and running.

BUGS

vinum is a new product. Bugs can be expected. The configuration mechanism is not yet fully functional. If you have difficulties, please look at the section **DEBUGGING PROBLEMS WITH VINUM** before reporting problems.

Kernels with the **vinum** device appear to work, but are not supported. If you have trouble with this configuration, please first replace the kernel with a non-**vinum** kernel and test with the LKM module.

Detection of differences between the version of the kernel and the LKM is not yet implemented.

The RAID-5 functionality is new in FreeBSD 3.3. Some problems have been reported with **vinum** in combination with soft updates, but these are not reproducible on all systems. If you are planning to use **vinum** in a production environment, please test carefully.

DEBUGGING PROBLEMS WITH VINUM

Solving problems with **vinum** can be a difficult affair. This section suggests some approaches.

Configuration problems

It is relatively easy (too easy) to run into problems with the **vinum** configuration. If you do, the first thing you should do is stop configuration updates:

```
vinum setdaemon 4
```

This will stop updates and any further corruption of the on-disk configuration.

Next, look at the on-disk configuration, using a Bourne-style shell:

```
rm -f log
for i in /dev/da0s1h /dev/da1s1h /dev/da2s1h /dev/da3s1h; do
  (dd if=$i skip=8 count=6|tr -d '\000-\011\200-\377'; echo) >> log
done
```

The names of the devices are the names of all **vinum** slices. The file `log` should then contain something like this:

```
IN VINOpanic.lemis.comdrive1}6E7~^K6T^Yfoovolume obj state up
volume src state up
volume raid state down
volume r state down
volume foo state up
plex name obj.p0 state corrupt org concat vol obj
plex name obj.p1 state corrupt org striped 128b vol obj
plex name src.p0 state corrupt org striped 128b vol src
plex name src.p1 state up org concat vol src
plex name raid.p0 state faulty org disorg vol raid
plex name r.p0 state faulty org disorg vol r
plex name foo.p0 state up org concat vol foo
plex name foo.p1 state faulty org concat vol foo
sd name obj.p0.s0 drive drive2 plex obj.p0 state reborn len 409600b driveoffset 265b plexoffset 0b
sd name obj.p0.s1 drive drive4 plex obj.p0 state up len 409600b driveoffset 265b plexoffset 409600b
sd name obj.p1.s0 drive drive1 plex obj.p1 state up len 204800b driveoffset 265b plexoffset 0b
sd name obj.p1.s1 drive drive2 plex obj.p1 state reborn len 204800b driveoffset 409865b plexoffset 128b
sd name obj.p1.s2 drive drive3 plex obj.p1 state up len 204800b driveoffset 265b plexoffset 256b
sd name obj.p1.s3 drive drive4 plex obj.p1 state up len 204800b driveoffset 409865b plexoffset 384b
```

The first line contains the **vinum** label and must start with the text "`IN VINO`". It also contains the name of the system. The exact definition is contained in `/usr/src/sys/dev/vinum/vinumvar.h`. The saved configuration starts in the middle of the line with the text "`volume obj state up`" and starts in sector 9 of the disk. The rest of the output shows the remainder of the on-disk configuration. It may be necessary to increase the **count** argument of dd(1) in order to see the complete configuration.

The configuration on all disks should be the same. If this is not the case, please report the problem with the exact contents of the file `log`. There is probably little that can be done to recover the on-disk configuration, but if you keep a copy of the files used to create the objects, you should be able to re-create them. The **create** command does not change the subdisk data, so this will not cause data corruption. You may need to use the **resetconfig** command if you have this kind of trouble.

Kernel Panics

In order to analyse a panic which you suspect comes from **vinum** you will need to build a debug kernel. See the online handbook at `/usr/share/doc/en/books/developers-handbook/kerneldebug.html` (if installed) or `http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/developers--handbook/kerneldebug.html` for more details of how to do this.

Perform the following steps to analyse a **vinum** problem:

1. Copy the files `/usr/src/sys/modules/vinum/.gdbinit.crash`, `/usr/src/sys/modules/vinum/.gdbinit.kernel`, `/usr/src/sys/modules/vinum/.gdbinit.serial`, `/usr/src/sys/modules/vinum/.gdbinit.vinum` and `/usr/src/sys/modules/vinum/.gdbinit.vinum.paths` to the directory in which you will be performing the analysis, typically `/var/crash`.

2. Make sure that you build the **vinum** module with debugging information. The standard `Makefile` builds a module with debugging symbols by default. If the version of **vinum** in `/modules` does not contain symbols, you will not get an error message, but the stack trace will not show the symbols. Check the module before starting gdb(1):

```
$ file /boot/kernel/vinum.ko
/boot/kernel/vinum.ko: ELF 32-bit LSB shared object, Intel 80386,
  version 1 (FreeBSD), not stripped
```

If the output shows that `/boot/kernel/vinum.ko` is stripped, you will have to find a version which is not. Usually this will be either in `/usr/obj/sys/modules/vinum/vinum.ko` (if you have built **vinum** with a "`make world`") or `/usr/src/sys/modules/vinum/vinum.ko` (if you have built **vinum** in this directory). Modify the file `.gdbinit.vinum.paths` accordingly.

3. Either take a dump or use remote serial `gdb`(1) to analyse the problem. To analyse a dump, say `/var/crash/vmcore.5`, link `/var/crash/.gdbinit.crash` to `/var/crash/.gdbinit` and enter:

```
cd /var/crash
gdb -k kernel.debug vmcore.5
```

This example assumes that you have installed the correct debug kernel at `/var/crash/kernel.debug`. If not, substitute the correct name of the debug kernel.

To perform remote serial debugging, link `/var/crash/.gdbinit.serial` to `/var/crash/.gdbinit` and enter

```
cd /var/crash
gdb -k kernel.debug
```

In this case, the `.gdbinit` file performs the functions necessary to establish connection. The remote machine must already be in debug mode: enter the kernel debugger and select **gdb** (see `ddb`(4) for more details). The serial `.gdbinit` file expects the serial connection to run at 38400 bits per second; if you run at a different speed, edit the file accordingly (look for the *remotebaud* specification).

The following example shows a remote debugging session using the **debug** command of `vinum`(8):

```
GDB 4.16 (i386-unknown-freebsd), Copyright 1996 Free Software Foundation, Inc.
Debugger (msg=0xf1093174 "vinum debug") at ../../i386/i386/db_interface.c:318
318                 in_Debugger = 0;
#1  0xf108d9bc in vinumioctl (dev=0x40001900, cmd=0xc008464b, data=0xf6dedee0 "",
    flag=0x3, p=0xf68b7940) at
    /usr/src/sys/modules/Vinum/../../dev/Vinum/vinumioctl.c:102
102             Debugger ("vinum debug");
(kgdb) bt
#0  Debugger (msg=0xf0f661ac "vinum debug") at ../../i386/i386/db_interface.c:318
#1  0xf0f60a7c in vinumioctl (dev=0x40001900, cmd=0xc008464b, data=0xf6923ed0 "",
      flag=0x3, p=0xf688e6c0) at
      /usr/src/sys/modules/vinum/../../dev/vinum/vinumioctl.c:109
#2  0xf01833b7 in spec_ioctl (ap=0xf6923e0c) at ../../miscfs/specfs/spec_vnops.c:424
#3  0xf0182cc9 in spec_vnoperate (ap=0xf6923e0c) at ../../miscfs/specfs/spec_vnops.c:129
#4  0xf01eb3c1 in ufs_vnoperatespec (ap=0xf6923e0c) at ../../ufs/ufs/ufs_vnops.c:2312
#5  0xf017dbb1 in vn_ioctl (fp=0xf1007ec0, com=0xc008464b, data=0xf6923ed0 "",
      p=0xf688e6c0) at vnode_if.h:395
#6  0xf015dce0 in ioctl (p=0xf688e6c0, uap=0xf6923f84) at ../../kern/sys_generic.c:473
#7  0xf0214c0b in syscall (frame={tf_es = 0x27, tf_ds = 0x27, tf_edi = 0xefbfcff8,
      tf_esi = 0x1, tf_ebp = 0xefbfcf90, tf_isp = 0xf6923fd4, tf_ebx = 0x2,
      tf_edx = 0x804b614, tf_ecx = 0x8085d10, tf_eax = 0x36, tf_trapno = 0x7,
      tf_err = 0x2, tf_eip = 0x8060a34, tf_cs = 0x1f, tf_eflags = 0x286,
      tf_esp = 0xefbfcf78, tf_ss = 0x27}) at ../../i386/i386/trap.c:1100
#8  0xf020a1fc in Xint0x80_syscall ()
#9  0x804832d in ?? ()
#10 0x80482ad in ?? ()
#11 0x80480e9 in ?? ()
```

When entering from the debugger, it is important that the source of frame 1 (listed by the `.gdbinit` file at the top of the example) contains the text "`Debugger ("vinum debug");`".

This is an indication that the address specifications are correct. If you get some other output, your symbols and the kernel module are out of sync, and the trace will be meaningless.

For an initial investigation, the most important information is the output of the **bt** (backtrace) command above.

Reporting Problems with Vinum

If you find any bugs in **vinum**, please report them to Greg Lehey ⟨grog@lemis.com⟩. Supply the following information:

- The output of the **vinum list** command (see `vinum`(8)).
- Any messages printed in `/var/log/messages`. All such messages will be identified by the text "`vinum`" at the beginning.
- If you have a panic, a stack trace as described above.

AUTHORS

Greg Lehey ⟨grog@lemis.com⟩.

HISTORY

vinum first appeared in FreeBSD 3.0. The RAID-5 component of **vinum** was developed by Cybernet Inc. (`http://www.cybernet.com/`), for its NetMAX product.

SEE ALSO

`disklabel`(5), `disklabel`(8), `newfs`(8), `vinum`(8)