NAME

vinum − Logical Volume Manager

SYNOPSIS

```
kldload vinum
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 file system. By attaching an additional plex and detaching at a specific time, the detached plex becomes an accurate snapshot of the file system 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 file systems, **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 kernel loadable module (kld), and does not require configuration. As with other klds, it is absolutely necessary to match the kld 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:

```
pseudo-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.

vinum previously required matching debug support between the kernel module and the userland program. This is no longer required. **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** program, which will load the kld 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.

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

The kld can only be unloaded when idle, in other words when no volumes are mounted and no other instances of the **vinum** program are active. Unloading the kld 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` calls are intended for the use of the **vinum** 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** will complain that it cannot determine the partition. To solve this problem, use the **-v** flag to **newfs**. For example, if you have a volume `concat`, use the following command to create a ufs file system 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 arbitary 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`(8) creates or deletes objects, it creates a directory `/dev/vinum`, in which it makes device entries for each volume. It also creates the subdirectories `/dev/vinum/plex` and `/dev/vinum/sd`, in which it stores device entries for the 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 two super-devices, `/dev/vinum/control` and `/dev/vinum/controld`. `/dev/vinum/control` is used by `vinum`(8), and `/dev/vinum/controld` is used by the **vinum** daemon.
- 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**, 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** 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
crwxr-xr--  1 root  wheel   91,   2 Mar 30 16:08 concat
crwx------  1 root  wheel   91, 0x40000000 Mar 30 16:08 control
crwx------  1 root  wheel   91, 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
crwxr-xr--  1 root  wheel   91,   3 Mar 30 16:08 strcon
crwxr-xr--  1 root  wheel   91,   1 Mar 30 16:08 stripe
crwxr-xr--  1 root  wheel   91,   0 Mar 30 16:08 tinyvol
drwxrwxrwx  7 root  wheel       512 Mar 30 16:08 vol
crwxr-xr--  1 root  wheel   91,   4 Mar 30 16:08 vol5

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

/dev/vinum/plex:
total 0
crwxr-xr--  1 root  wheel   91, 0x10000002 Mar 30 16:08 concat.p0
crwxr-xr--  1 root  wheel   91, 0x10010002 Mar 30 16:08 concat.p1
crwxr-xr--  1 root  wheel   91, 0x10000003 Mar 30 16:08 strcon.p0
crwxr-xr--  1 root  wheel   91, 0x10010003 Mar 30 16:08 strcon.p1
crwxr-xr--  1 root  wheel   91, 0x10000001 Mar 30 16:08 stripe.p0
crwxr-xr--  1 root  wheel   91, 0x10000000 Mar 30 16:08 tinyvol.p0
crwxr-xr--  1 root  wheel   91, 0x10000004 Mar 30 16:08 vol5.p0
crwxr-xr--  1 root  wheel   91, 0x10010004 Mar 30 16:08 vol5.p1

/dev/vinum/sd:
total 0
crwxr-xr--  1 root  wheel   91, 0x20000002 Mar 30 16:08 concat.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100002 Mar 30 16:08 concat.p0.s1
crwxr-xr--  1 root  wheel   91, 0x20010002 Mar 30 16:08 concat.p1.s0
crwxr-xr--  1 root  wheel   91, 0x20000003 Mar 30 16:08 strcon.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100003 Mar 30 16:08 strcon.p0.s1
crwxr-xr--  1 root  wheel   91, 0x20010003 Mar 30 16:08 strcon.p1.s0
crwxr-xr--  1 root  wheel   91, 0x20110003 Mar 30 16:08 strcon.p1.s1
crwxr-xr--  1 root  wheel   91, 0x20000001 Mar 30 16:08 stripe.p0.s0
```

```
crwxr-xr--  1 root  wheel   91, 0x20100001 Mar 30 16:08 stripe.p0.s1
crwxr-xr--  1 root  wheel   91, 0x20000000 Mar 30 16:08 tinyvol.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100000 Mar 30 16:08 tinyvol.p0.s1
crwxr-xr--  1 root  wheel   91, 0x20000004 Mar 30 16:08 vol5.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100004 Mar 30 16:08 vol5.p0.s1
crwxr-xr--  1 root  wheel   91, 0x20010004 Mar 30 16:08 vol5.p1.s0
crwxr-xr--  1 root  wheel   91, 0x20110004 Mar 30 16:08 vol5.p1.s1

/dev/vinum/vol:
total 5
crwxr-xr--  1 root  wheel   91,   2 Mar 30 16:08 concat
drwxr-xr-x  4 root  wheel       512 Mar 30 16:08 concat.plex
crwxr-xr--  1 root  wheel   91,   3 Mar 30 16:08 strcon
drwxr-xr-x  4 root  wheel       512 Mar 30 16:08 strcon.plex
crwxr-xr--  1 root  wheel   91,   1 Mar 30 16:08 stripe
drwxr-xr-x  3 root  wheel       512 Mar 30 16:08 stripe.plex
crwxr-xr--  1 root  wheel   91,   0 Mar 30 16:08 tinyvol
drwxr-xr-x  3 root  wheel       512 Mar 30 16:08 tinyvol.plex
crwxr-xr--  1 root  wheel   91,   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
crwxr-xr--  1 root  wheel   91, 0x10000002 Mar 30 16:08 concat.p0
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 concat.p0.sd
crwxr-xr--  1 root  wheel   91, 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
crwxr-xr--  1 root  wheel   91, 0x20000002 Mar 30 16:08 concat.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100002 Mar 30 16:08 concat.p0.s1

/dev/vinum/vol/concat.plex/concat.p1.sd:
total 0
crwxr-xr--  1 root  wheel   91, 0x20010002 Mar 30 16:08 concat.p1.s0

/dev/vinum/vol/strcon.plex:
total 2
crwxr-xr--  1 root  wheel   91, 0x10000003 Mar 30 16:08 strcon.p0
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 strcon.p0.sd
crwxr-xr--  1 root  wheel   91, 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
crwxr-xr--  1 root  wheel   91, 0x20000003 Mar 30 16:08 strcon.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100003 Mar 30 16:08 strcon.p0.s1

/dev/vinum/vol/strcon.plex/strcon.p1.sd:
total 0
crwxr-xr--  1 root  wheel   91, 0x20010003 Mar 30 16:08 strcon.p1.s0
```

```
crwxr-xr--  1 root  wheel   91, 0x20110003 Mar 30 16:08 strcon.p1.s1

/dev/vinum/vol/stripe.plex:
total 1
crwxr-xr--  1 root  wheel   91, 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
crwxr-xr--  1 root  wheel   91, 0x20000001 Mar 30 16:08 stripe.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100001 Mar 30 16:08 stripe.p0.s1

/dev/vinum/vol/tinyvol.plex:
total 1
crwxr-xr--  1 root  wheel   91, 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
crwxr-xr--  1 root  wheel   91, 0x20000000 Mar 30 16:08 tinyvol.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100000 Mar 30 16:08 tinyvol.p0.s1

/dev/vinum/vol/vol5.plex:
total 2
crwxr-xr--  1 root  wheel   91, 0x10000004 Mar 30 16:08 vol5.p0
drwxr-xr-x  2 root  wheel       512 Mar 30 16:08 vol5.p0.sd
crwxr-xr--  1 root  wheel   91, 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
crwxr-xr--  1 root  wheel   91, 0x20000004 Mar 30 16:08 vol5.p0.s0
crwxr-xr--  1 root  wheel   91, 0x20100004 Mar 30 16:08 vol5.p0.s1

/dev/vinum/vol/vol5.plex/vol5.p1.sd:
total 0
crwxr-xr--  1 root  wheel   91, 0x20010004 Mar 30 16:08 vol5.p1.s0
crwxr-xr--  1 root  wheel   91, 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 don't completely trust, and which we don't 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 won't 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 fulfil 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

1. **`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.
2. Kernels with the **`vinum`** pseudo-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 kld module.
3. Detection of differences between the version of the kernel and the kld is not yet implemented.
4. 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 with the **vinum dumpconfig** command, for example:

```
# vinum dumpconfig
Drive 4:        Device /dev/da3h
                Created on crash.lemis.com at Sat May 20 16:32:44 2000
                Config last updated Sat May 20 16:32:56 2000
                Size:        601052160 bytes (573 MB)
volume 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 configuration on all disks should be the same. If this is not the case, please save the output to a file and report the problem. 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/handbook/kerneldebug.html (if installed) or http://www.FreeBSD.org/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**:

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

   If the output shows that /modules/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** 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**. 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=0xf66b7940) 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's 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.
- 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.

AUTHOR

HISTORY

vinum **vinum** www.cybernet.com

SEE ALSO

vinum8) disklabel5) disklabel8) newfs8)