The Coda HOWTO

Peter Braam, braam@cs.cmu.edu, Robert Baron rvb@cs.cmu.edu, Jan Harkes jaharkes@cs.cmu.edu, Marc Schnieder smarc@cs.cmu.edu. v0.50, Dec 12, 1998

This HOWTO should give you help in running Coda. We try to cover enough detail for you to install, build and run Coda, and also explain how to troubleshoot and prepare useful bug reports.

Contents

1 Coda's ingredients

1.1 What is Coda ?

Coda is a distributed file system, i.e. it makes files available to a collection of client computers as part of their directory tree, but ultimately maintains the authoritative copy of the file data on servers. Coda has some features that make it stand out: it supports *disconnected operation*, i.e. full access to a cached section of the file space during voluntary or involuntary network or server outages. Coda will automatically reintegrate the changes made on disconnected clients when reconnecting. Further Coda has read write, failover server replication, meaning that data is stored and fetch from any of a group of servers and Coda will continue to operate when only a subset of all servers is available. If server differences arise due to *network partitions* Coda will resolve differences automatically to a maximum extent possible and aid users in repairing what can't be done automatically. Coda is very differently organized from NFS and Windows/Samba shares. Coda does have many similarities to AFS and DCE/DFS.

1.2 Getting clued in with the Coda terminology

A single name space

All of Coda appears under a single directory `/coda` on the client (or under a single drive under Windows). Coda does not have different exports or shares as do NFS and Samba that are individually mounted. Under `/coda` the volumes (aka file sets) of files exported by all the servers (living in your Coda cell) are visible. Coda automatically finds servers and all a client needs to know is the name of one bootstrap server that gives it information how to find the root volume of Coda.

A Coda cell

is a group of servers sharing one set of configuration databases. A cell can consist of a single server or up to hundreds of servers. One server is designated as the **SCM**, the system Control machine. It is distinguished by being the only server modifying the configuration databases shared by all servers, and propagating such changes to other servers. At present a Coda client can belong to a single cell. We hope to get a cell mechanism into Coda whereby a client can see files in multiple cells.

Coda volumes:

File servers group the files in volumes. A volume is typically much smaller than a partition and much larger than a directory. Volumes have a root and contain a directory tree with files. Each volume is "Coda mounted" somewhere under /coda and forms a subtree of the /coda. Volumes can contain mountpoints of other volumes. A volume mountpoint is not a Unix mountpoint or Windows drive - there is only one drive or Unix mountpoint for Coda. A Coda mountpoint contains enough information for the client to find the server(s) which store the files in the volume. The group of servers serving a volume is called the *Volume Storage Group* of the volume.

Volume Mountpoints

One volume is special, it is the root volume, the volume which Coda mounts on `/coda`. Other volumes are grafted into the `/coda` tree using the **cfs makemount**. This command installs a volume mountpoint in the Coda directory tree, and in effect its result is similar to **mkdir mountpoint ; mount device mountpoint** under Unix. When invoking the **cfs makemount** the two arguments given are the name of the mountpoint and the name of the volume to be mounted. Coda mountpoints are persistent objects, unlike Unix mountpoints which needs reinstating after a reboot.

Data storage

The servers do not store and export volumes as directories in the local disk filesystem, like NFS and Samba. Coda needs much more meta data to support server replication and disconnected operation and it has complex recovery which is hard to do within a local disk filesystem. Coda servers store files identified by a number typically all under a directory `/vicepa`. The meta data (owners, access control lists, version vectors) and directory contents is stored in an RVM data file which would often be a raw disk partition.

RVM

stands for *Recoverable Virtual Memory*. RVM is a transaction based library to make part of a virtual address space of a process persistent on disk and commit changes to this memory atomically to persistent storage. Coda uses RVM to manage its metadata. This data is stored in an RVM data file which is mapped into memory upon startup. Modifications are made in VM and also writtent to the RVM LOG file upon committing a transaction. The LOG file contains committed data that has not yet been incorporated into the data file on disk.

Client data

is stored somewhat similarly: meta data in RVM (typically in `/usr/coda/DATA`) and cached files are stored by number under `/usr/coda/venus.cache`. The cache on a client is persistent. This cache contains copies of files on the server. The cache allows for quicker access to data for the client and allows for access to files when the client is not connected to the server.

Validation

When Coda detects that a server is reachable again it will *validate* cached data before using it to make sure the cached data is the latest version of the file. Coda compares cached version stamps associated with each object, with version stamps held by the server.

Authentication

Coda manages authentication and authorization through a token. Similar (the details are very different) to using a Windows share, Coda requires users to log in. During the log in process, the client acquires a session key, or token in exchange for a correct password. The token is associated with a user identity, at present this Coda identity is the uid of the user performing the log in.

Protection

To grant permissions the cache manager and servers use the token with its associated identity and match this against priviliges granted to this identity in access control lists (ACL). If a token is not present, anonymous access is assumed, for which permissions are again granted through the access control lists.

1.3 Organization of the client

1.3.1 The kernel module and the cache manager

Like every filesystem a computer enabled to use the Coda filesystem needs kernel support to access Coda files. Coda's kernel support is minimal and works in conjunction with the cache manager **Venus**. User requests enter the kernel, which will either reply directly or ask the cache manager **venus** to assist in service.

Typically the kernel code is in a kernel module, which is either loaded at boot time or dynamically loaded when Venus is started. Venus will even mount the Coda filesystem on `/coda`.

1.3.2 Utilities

To manipulate acl's, the cache, volume mountpoints and possibly the network behaviour of a Coda client a variety of small utilities is provided. The most important one is the **cfs** command.

There is also a **clog** program to authenticate to the Coda authentication server. The **codacon** programm allows one to monitor the operatoin of the cache manager, and **cmon** program gives summary information about a list of servers.

1.4 The server organization

The main program is the Coda fileserver **codasrv**. It is responsible for doing all file operations, as well as volume location service.

The Coda authentication server **auth2** handles requests from **clog** for tokens, and changes of password from **au** and **cpasswd**. Only the the auth2 process on the SCM will modify the password database.

All servers in a Coda cell share the configuration databases in `/vice/db` and retrieve them from the SCM when changes have occurred. The **updateclnt** program is responsible for retrieving such changes, and it polls the **updatesrv** on the SCM to see if anything has changed. Sometimes the SCM needs a (non-shared) database from another server to update a shared database. It fetches this through an **updatesrv** process on that server using **updatefetch**.

1.4.1 Utilities

On the server there are utilities for volume creation and management. These utilities consist of shell scripts and the volutil command. There is also a tool to manipulate the protection databases.

2 Getting Coda

All code can be obtained from: *ftp://ftp.coda.cs.cmu.edu/pub/coda* `<ftp://ftp.coda.cs.cmu.edu/pub/coda>`. For any platform you need:

- documentation and manual pages *coda-doc*
- user level client code for coda: *coda-client*
- a server: *coda-debug-server* rpms
- kernel code for the client: *coda-kernel-module* rpms code

NOTE: Coda is *platform independent*. You can mix and match servers and clients on any supported platform.

2.1 Linux

2.1.1 Getting RPM packages

We have RPMS for a RedHat Linux at *ftp://ftp.coda.cs.cmu.edu/pub/coda/linux/* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/linux/>`. You should get:

- documentation and manual pages *coda-doc* rpms
- a client: *coda-debug-client* rpms
- a server: *coda-debug-server* rpms
- kernel code for the client: *coda-kernel-module* rpms code

Install these as ususal.

You may need to build a Coda kernel module for which you will need source. Look at the section 6.1.1 (building a kernel module.)

2.2 FreeBSD

2.2.1 Obtaining Packages

As of this writing, the elf packages have been built but not tested as heavily as the aout packages. You may obtain the newest compiled packages from the *Coda site* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/`

`freebsd/3.0-elf/i386/>` or other mirrors. **NOTE:** Below I am using a specific Coda release, 4.6.6.1, you will no doubt find a different release on the ftp site when you access it; so use the newer release version.

- Coda-intro is a quick get started guide to using Coda.

  ```
  pkg_add coda-intro-4.6.6.1.tgz
  ```

- The Coda client is needed by a machine to use the Coda filesystem. Aprox Size = 3.6Meg

  ```
  pkg_add coda-client-4.6.6.1.tgz
  ```

- The Coda server is needed by a machine to create and serve the filesystem to client machines. Aprox Size = 5.5Meg

  ```
  pkg_add coda-server-4.6.6.1.tgz
  ```

- Finally, we have a wealth of documentation about Coda and need to generate still more. Aprox Size = 1.5Meg

  ```
  pkg_add coda-doc-4.6.5.3.tgz
  ```

If you substitute 3.0-aout for 3.0-elf in the ftp directory, you will get the aout packages. The aout packages for the client and server are about 1.5Meg larger than the elf packages.

NOTE: pkg_add may fail silently. Please make sure you have at least 25MB of free space available in `/var/tmp` to install the each package.

2.3 NetBSD

2.3.1 Obtaining Pre-compiled Packages

You may obtain the newest pre-compiled packages from the Coda site *Coda site* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/netbsd/current/i386/>` or other mirrors. **NOTE:** Below I am using a specific Coda release, 4.6.6.1, you will no doubt find a different release on the ftp site when you access it; so use the newer release version.

- Coda-intro is a quick get started guide to using Coda.

  ```
  pkg_add coda-intro-4.6.6.1.tgz
  ```

- The Coda client is needed by a machine to use the Coda filesystem. Aprox Size = 5Meg

  ```
  pkg_add coda-client-4.6.6.1.tgz
  ```

- The Coda server is needed by a machine to create and serve the filesystem to client machines. Aprox Size = 6.8Meg

  ```
  pkg_add coda-server-4.6.6.1.tgz
  ```

- Finally, we have a wealth of documentation about Coda and need to generate still more. Aprox Size = 1.5Meg

  ```
  pkg_add coda-doc-4.6.5.3.tgz
  ```

Note: pkg_add may fail silently. Please make sure you have at least 25MB of free space available in /var/tmp to install the each package.

2.4 Windows 95

Get the installer files for servers and clients from *ftp://ftp.coda.cs.cmu.edu/pub/coda/win95* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/win95>`.

The Server has not been tested on Windows 95 and might not run at all. Use the Server for Windows NT instead.

Install these by double clicking on the installer files.

NOTE: Before you start, make sure you know the IP address of your client and of the server you are connecting to. The default server address offered is that of the testserver at CMU.

2.5 Windows NT

Do all your work as administrator.

First get the Cygwin B19 kit from *http://www.cygnus.com/misc/gnu-win32/* `<http://www.cygnus.com/misc/gnu-win32/>` and install this on the NT machine, for example in C:\Cygnus

Get the installer files for servers and clients from *ftp://ftp.coda.cs.cmu.edu/pub/coda/winnt* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/winnt>`.

Install these by double clicking on the installer files. (Note: these replace the original cygwin32.dll with a patched version. A future version of Coda will strive to use the default version).

3 Configuring and starting Coda

WARNING:

CODA IS BARELY READY FOR PRODUCTION USE. THIS RELEASE IS JUST FOR THOSE INTERESTED IN EXPLORING THOSE FEATURES WHICH WORK. IT CONTAINS KERNEL CODE, AND SERVERS RUNNING WITH ROOT PRIVILEGES, AND COULD LEAD TO DATA LOSS.

To get Coda running you will go through 3 steps:

1. Get a Coda enabled kernel for the client. (The server does not need a special kernel)
2. Configure and run the client cache manager Venus
3. Configure and run the file server, authentication server and update servers
4. Connect a client to your new server.

3.1 Getting a Coda enabled kernel

Before you can use a Coda client you need a new filesystem driver in your kernel. For the most part this driver redirects requests to the user level cache manager **venus**. Precompiled modules exist for commonly used kernels. If no module exists for you, you'll find instructions below where to find it.

3.1.1 Linux

Get a *coda-fs-module-k2.?.?-c?.?.?.arch.rpm* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/linux>` rpm package and install it. `k2.?.?` should match your kernel version (use uname -a) and `c?.?.?` should match you Coda version.

Linux kernels change often and many people have custom kernels for their environment. Our modules will generally only work on RedHat Linux kernels and you may have to build a module for your kernel. Look at the section 6.1.1 (building a kernel module.)

3.1.2 FreeBSD

You may obtain a Coda lkm from your FreeBSD distribution:

```
cd /lkm
```

Check if `coda_mod.o` is there already. Otherwise, you may obtain a Coda lkm from the Coda site:

```
ftp://ftp.coda.cs.cmu.edu/pub/coda/freebsd/3.0-aout/i386/coda_mod.o
```

You then install the lkm with:

```
modload -v -e coda_mod -o /var/run/lkm.coda /lkm/coda_mod.o
```

You can build support for the Coda VFS layer into your kernel. This is discussed later in the *Building FreeBSD* section.

3.1.3 NetBSD

NOTE: The GENERIC NetBSD kernel should have Coda enabled. Do an:

```
nm -o /netbsd | grep coda_open
```

If this is present, you are done with this section. **PROCEED NO FURTHER**

If you need to load an lkm it should be in the NetBSD distribution:

```
cd /usr/lkm
```

Check if coda.o is there already. Otherwise, you may obtain a Coda lkm from the Coda site:

```
ftp://ftp.coda.cs.cmu.edu/pub/coda/netbsd/current/i386/coda-1_3H.o
```

You then install the lkm with:

```
modload -v -e coda_lkmentry -o /var/run/lkm.coda /usr/lkm/coda-1_3H.o
```

You can build support for the Coda VFS layer into your kernel, though this should be automatic. This is discussed later in the *Building NetBSD* section.

3.1.4 Windows 95 & NT

The kernel module is part of the Coda client installer.

3.2 Linux and the BSD's: running Venus, the client cache manager

These are partial instructions on how to setup and configure the Coda filesystem. Refinements to the setup created here are discussed in *http://ftp.coda.cs.cmu.edu/doc/html/manual.html* `<http://ftp.coda.cs.cmu.edu/doc/html/manual.html>` You will probably not need these refinements in the first instance.

The `venus-setup` script does all the hard work, it will setup the coda control files, create `/dev/cfs0` to communicate with the kernel, ... It also initializes a directory for cache files. In your first Coda run we recommend a small cache, say 20MB. The cache size should be at least 10Meg, typically 60-200Meg is used. Do not go above 300Meg. All the files created will be placed under `/usr/coda`. You should make sure that there is enough space in the file system on which `/usr/coda` resides to hold a fully populated cache.

```
venus-setup <comma_separated_host_list> <cache_size_in_kb>
```

`venus-setup` and `venus` (below) are in `/usr/sbin`. Make sure that `/usr/sbin` is in your path or that you use fully qualified pathnames. We strongly recommend that you try testserver.coda.cs.cmu.edu as the `comma_separated_host_list` first, and keep the cache size to 20000.

NOTE: `venus-setup` will edit `/etc/services` to add some additional services.

The following assumes you are running X-Windows. However, you could run these commands from virtual consoles as well, by omitting the `xterm -e` in front of the commands below.

Start Venus with:

```
venus &
```

An `-init` flag can be given when `venus` is started; it flushes the local cache contents. `venus-setup` forces an init to happen when venus is first started. The `-init` flag can be given if Coda cannot recover it's cache after a crash, or after editing the `vstab` file manually.

Observe the venus log with:

```
xterm -e tail -f /usr/coda/etc/console
```

It will tell you when venus has started and give status.

Type:

```
xterm -e codacon &
```

to see the communications between the Venus and Vice.

It is possible to see the upcalls from the kernel to Venus by turning up logging in Venus, but they are not very interesting. (To turn on minimal debugging, type:

```
vutil -d 1
```

and then `tail -f /usr/coda/coda.cache/venus.log`.)

To halt venus, type:

```
vutil shutdown
umount /coda  (Linux only)
```

Or you can kill -9 venus, if you must.

NOTES for Linux users:

- Before restarting Venus `/coda` must be unmounted. If umounting `/coda` gives trouble, make sure to exit all process that hang on to Coda, e.g. by having files open or being cd'd into /coda. A utility like `lsof` and `fuser` can help with this.
- `/proc/fs/coda` has interesting Coda statistics.
- You can enable kernel debugging with `vutil -kdebug 4095` and call tracing with `vutil ktrace 1`. The messages appear in `/var/log/messages`

3.3 Windows 95: Starting and Configuring a Coda client

During installation you will be prompted for the IP address of your client and of your Coda server(s). Enter this as indicated.

All executables can be found in the directory `C:\usr\coda\bin.`

1. Type

   ```
   relay.exe
   ```

   in a command Window. This is a 32bit DOS application which you can also start by double clicking on the file in explorer.

2. If this is the first time you start Coda run

   ```
   venus.exe -child
   ```

   in a command window.

3. Run

   ```
   venus.exe -cf 500 -init
   ```

   (this gives up to 500 cache files, try with more if everythings works fine.)

4. When Venus prints "Venus starting ..." run

   ```
   mount.exe N:
   ```

   This maps the Coda filesystem to the `N:` drive.

You are now ready to browse through the Coda filesystem using the explorer!

1. When you want to restart Venus without initializing the cache just run `venus.exe`. If `venus` crashes restarting it is normally all you need to do.

2. If relay crashes you will want to unmount the `N:` drive run:

   ```
   unmount.exe
   ```

   This will stop `relay.exe` (if it is still running) and unload `codadev.vxd` which was loaded by `relay.exe`.

3.4 Codasrv – the file server

3.4.1 Configuring your server

To set up an SCM server, you will run a script *vice-setup*. This script creates configuration files under the `/vice` directory and creates files and directories on your system for storage of file data and metatdata. To answer the questions *vice-setup* is asking you need to have the following thought through:

file space

an empty directory (viz /vicepa) where the fileserver will put files. There must be as much free space on this filesystem as the data you wish to store in Coda.

RVM metadata storage

a file or raw partition for RVM metadata. You can use a file but it will be quite slow on a larger server. This partition must be around 4% of the total size of the files you wish to store under /vicepa (e.g. on a 2GB server we use around 80M of rvm data). For first installations we recommend the default 22M options, and using *files* for RVM log and data. (NOTE: Windows NT Setup creates the file c:\coda\rvm\DATA. Use this for RVM metadata.)

virtual memory

The metadata, held in the RVM data file, is memory mapped. You need that amount of space as virtual memory on your system, in addition to virtual memory to run the server (6MB) and other software.

RVM transaction log

a LOG file, preferably a raw partition on a disk by itself. This needs not be large, a few M's are fine. (NOTE: Windows NT Setup creates the file c:\coda\rvm\LOG. Use this for RVM transaction log.)

A server number

All servers in a coda cell need to have a unique number to identify them. The servername to identifier mappings have to be defined by the administrator in the file `/vice/db/servers` on the SCM. The format of this file is as follows:

```
servernameX.domain.name    1
servernameY.domain.name    2
...
```

There are currently several limitations to which identifiers are actually usable:

- all numbers must fit in a single byte.
- 0 and -1 (255) are used in error conditions.
- 127 is used to identify 'replicated volumes'.

This leaves us with a usable range of 1-126 and 128-245 for server identifiers.

secret tokens

two secret tokens of _exactly_ 8 characters (eg elephant).

NOTE: you are now ready to run the setup script. We **strongly** recommend that you stick to default choices offered as configuring a server differently is quite difficult.

Then run:

```
vice-setup
```

and answer its questions. Note down the commands that vice-setup prints out for you at the end.

Note: For Windows NT you will need the Cygwin B19 Shell which can be started from the "Start" menu. The shell uses the c:\coda directory as the root mountpoint "/".

3.4.2 Running Vice:

Start the rpc2portmap server, update server, client and the auth server, as well as the fileserver by typing:

Linux

```
/etc/rc.d/init.d/auth2.init start
/etc/rc.d/init.d/update.init start
/etc/rc.d/init.d/codasrv.init start
```

BSDs

```
/etc/rc.vice start
```

Windows NT

```
codastart
```

Now observe the log:

```
xterm -e tail -f  /vice/srv/SrvLog &
```

The `SrvLog` should show *File Server started*. If not, you have a problem.

Determine with `ps` that `codasrv`, `auth2`, `rpc2portmap`, `updatesrv` and `updateclnt` are running.

3.4.3 Making your root volume

During your configuration session, you commmunicated a name for the root volume to the program. This root volume now needs to be craeted: the precise command to do this was printed out by the vice-setup program, below we assume your file space is in `/vicepa`, and your root volume is `coda:root`.

```
createvol_rep coda:root E0000100 /vicepa
```

NOTE: E0000100 is the Volume Storage Group set up for you by vice-setup. With more servers you can define other groups in /vice/db/VSGDB – see the Coda User Manual.

3.5 Connecting your client to a new Coda server

Now you are ready to point a Venus (client) at this server. You do this by typing

```
venus-setup  server-name cache-szie-in-kb
```

NOTE: Windows 95 users should type the IP address and not the hostname of the server.

Start Venus as explained above. From the coda client side, the root volume will appear under /coda. To use it you must now authenticate to Coda, since it is write protected.

The vice-setup program installed an administrative Coda user on the server. It has a uid you chose and has been assigned password changeme. You may clog into Coda with this uid:

```
clog "adminuser"
```

Validate that you have tokens with the `ctokens` utility.

You can now create files in `coda`, because the administrative user is on the access control list (ACL) of the `/coda` directory. Read the next section to find out how to do more with Coda.

4 Exploring Coda's features

4.1 Restarting a server

On the server the command `volutil shutdown` will stop the server. If it is really toasted, you might need to kill the *codasrv* process.

To restart the server issue the command `startserver`. This shell script invokes `codasrv` with the correct arguments. If it complains about a server already running do the following: first check with `ps auxww | grep codasrv` if this is the case. If not, remove the file `/vice/srv/pid` and reissue the `startserver` command.

4.2 Getting more volumes

It is a good idea to create a few extra volumes on your server and mount these in the coda directory tree. In particular if you want to explore reintegration, conflict resolution and replication servers we recommend that you do that in new volumes and not in the volume mounted on the root of the Coda filesystem.

To make a new volume:

1. make sure your codasrv, updatecln and updatesrv are running.
2. type `createvol_rep volname vsgaddr partitionname`. Typically you determine your own volume name, the VSG address can be found in /vice/db/VSGDB and the partition name will be /vicepa.
3. mount the volume using `cfs makemount path volumename`. This creates the mount point, and mounts the volume.

To **explore volumes** you can use other cfs commands, such as `cfs whereis path` and `cfs listvolume path`. To see the FID of a file type `cfs getfid path`.

4.3 Adding a user

This is currently slightly involved. Follow the following steps:

1. add the user to /vice/db/user.coda
2. add the user to groups in /vice/db/group.coda (do not separate users with commas)
3. run `cd /vice/db; pwd2pdb -u user.coda -g group.coda > vice.pdb`
4. run `pcfgen vice.pdb`.
5. Give the new user a password by using the `au nu` program (from either a client or a server).

The user should now be able to login to Coda just like the first user set up at installation time.

4.4 Exploring ACL's.

You can set and list Access Control Lists on directories using `cfs setacl dir user rights [user rights...]`. To show the ACLs type `cfs listacl directory`.

4.5 Monitoring Coda

Use `codacon` to see many RPC's and a few other actions taken by the client. The file `/usr/coda/etc/console` also has interesting information.

To see how a server is doing use `cmon`, in the form `cmon server:25`, 25 depicts the width of the column used by the server.

4.6 Using Coda disconnected and reintegrating.

First create a new volume, not equal to your root volume, to explore this. Should you get conflicts in your root volume then they will be hard to repair. See below how to add a volume.

1. change directory into this volume. Do "ls -l" on the directory where you want to start work while disconnected. Get tokens using `clog`.
2. type `cfs disconnect` or disconnect your network. After 30 seconds you can see that *codacon* (which you should always run) displays that your server is not reachable anymore.
3. do some work disconnected, create some files, edit them etc.
4. type `cfs reconnect`. Venus will discover that the net is up, but you can speed that up by typing `cfs checkservers`. Monitoring codacon, you will see that your changes are being re-integrated.

To start Coda while disconnected, you need the ip addresses and hostnames of your servers in `/etc/hosts`.

4.7 Repairing conflicts

Files and directories can get into conflict due to disconnections of clients or servers from the net, as well as through overlapping open/write/close sequences on two clients. A object that is in conflict is representd as a dangling symbolic link and `X->@vol.vnode.unique` is the file the symlinks points to.

How do we get rid of this conflict?

1. type `cfs beginrepair X`. This changes X from a symlink into a directory. By doing `ls X` you will see either `local global` or `server1 server2 server3`. In the first case we have a local global conflict and in the second case a server server conflict.
2. If the objects in the directory `X` are files, you have a file conflict, they can also be directories, in which case you can find the content underneath.
3. If you are nervous, this is a good moment to make copy of your files. They can be found under the directory X while the repair session is in progress.
4. `cfs endrepair X` closes up the repair session.
5. All local global conflicts are repaired with `repair`. Type `repair` and follow its cryptic instructions.
6. Server-server conflicts on files are fixed with either `filerepair` or with `removeinc`. Server-server conflicts on directories are fixed with `repair`.

4.8 Exploring replication

First you will have to add a second server to your Coda cluster. Install the software and use `vice-setup` again. This time your server is *not* going to be the SCM. Proceed answering the questions until done.

On the SCM add the following:

1. Your server needs a server number, to be added to the `/vice/db/servers` file **ON THE SCM**.
2. Make two new entries in the `/vice/db/VSGDB` file. One for your new server by itself, one of the form: `E0000104 scm-server second-server`.
3. Start updatesrv and updateclnt on the second server.
4. Start codasrv on the second server
5. Make a new volume **from the SCM** using `createvol_rep` giving the address of the volume as `E0000104`.
6. Mount the volume as above.

You can now use this volume and your files will automatically be stored on multiple servers. To temporarily disable a server, and see that things continue to function normally, either shut the server down with `volutil shutdown` or disconnect its network. You can also isolate the server using our network filters with `filcon isolate -s server-name`. Using `filcon clear server-name` clears the filter.

Modifications made to coda files during the server outage will be resolved when the files are first accessed. You see message of the form `Resolve path` in the codacon output. By typing `cfs checkservers` you can see if the server is available again.

5 Troubleshooting

The Coda filesystem is still under development, and there certainly are several bugs which can crash both clients and servers. However, many problems users observe are related to semantical differences of the Coda filesystem compared to well-known NFS or SMB network filesystems.

This section will point out several logs to look at for identifying the cause of problems. Even if the source of the problem cannot be found, the information gathered from Coda's logging mechanisms will make it easier for people on the coda mailinglist <*coda-discuss@coda.cs.cmu.edu*> to assist in solving the problem(s).

Some of the more common problems are illustrated in detail. At the end of this section some of the more involved debugging techniques will be addressed. This will be helpful to developers to isolate problems more easily.

At the end there is a whole section describing how to solve some problems with Windows95, **only the Coda related stuff!**.

5.1 Basic Troubleshooting

Most problems can be solved, or at least recognized by using the information logged by the clients and servers. The first step in finding out where the problems stems from is doing a **tail -f** on the logfiles.

It must also be noted that, when coda clients and servers crash they do not 'dump core', but start sleeping so that we developers can attach debuggers. As a result, a crashed client or server still shows up in the **ps auxwww** output, and only the combination of lack of file-service and error messages in logfiles indicate that something is really wrong.

5.1.1 Client debugging output

- **codacon** is a program which connects to venus and provides the user with run-time information. It is the initial source of information, but cannot be used to look back into the history. It is therefore advisable to always have a codacon running in a dedicated xterm.

  ```
  client$ xterm -e codacon
  ```

- `/usr/coda/etc/console` is a logfile which contains mostly error or warning messages, and is a place to look for errors which might have occured. When assertions in the code fail, it is logged here.
- `/usr/coda/venus.cache/venus.log` contains more in-depth information about the running system, which can be helpful to find out what the client is or was doing.

5.1.2 Server logs

- **cmon** is an ncurses program that can be run on a client to gather and display statistics from a group of servers. When a server goes down it will not respond to the statistics requests, which makes this a simple method for monitoring server availability.

  ```
  client$ xterm -e cmon server1:100 server2:100 server3:100 ...
  ```

- `/vice/srv/SrvLog and /vice/srv/SrvErr` are the server logfiles.
- `/vice/auth2/AuthLog`
- `/vice/srv/portmaplog`
- `/vice/srv/UpdateClntLog`
- `/vice/srv/UpdateLog`

5.2 Client Problems

5.2.1 Client does not connect to `testserver.coda.cs.cmu.edu`.

When you have set up your client for the first time, and it can not connect to the testserver at CMU, there are a couple of possible reasons. You might be running an old release of coda, check the coda web-site to see what the latest release is.

Another common reason is that your site is behind a firewall, which blocks, or allows only outgoing, udp traffic. Either try coda on a machine outside of the firewall, or set up your own server.

The third reason is that the testserver might be down, for maintenance or upgrades. That does not happen often, but you can check whether it is up, and how long it has been running using cmon.

```
cmon testserver.cs.cmu.edu:100
```

5.2.2 Venus comes up but prints cannot find RootVolume

All of the reasons in the previous item could be the cause. It is also possible that your `/etc/services` file is not allright. It needs the entries:

```
# Iana allocated Coda filesystem port numbers
rpc2portmap     369/tcp
rpc2portmap     369/udp    # Coda portmapper
codaauth2       370/tcp
codaauth2       370/udp    # Coda authentication server

venus           2430/tcp   # codacon port
venus           2430/udp   # Venus callback/wbc interface
venus-se        2431/tcp   # tcp side effects
venus-se        2431/udp   # udp sftp side effect
codasrv         2432/tcp   # not used
codasrv         2432/udp   # server port
codasrv-se      2433/tcp   # tcp side effects
codasrv-se      2433/udp   # udp sftp side effect
```

5.2.3 Trying to access a file returns *Connection timed out* (ETIMEDOUT).

The main reason for getting *Connection timed out* errors is that the volume where the file is located is disconnected from the servers. However, it can also occur in some cases when the client is in write-disconnected mode, and there is an attempt to read a file which is open for writing. See 5.4 (Volume is disconnected/Volume is write-disconnected) for more information.

5.2.4 Commands do not return, except by using ^C./

When command are hanging it is likely that venus has crashed. Check `/usr/coda/etc/console` and `/usr/coda/venus.cache/venus.log`.

5.2.5 Venus fails when restarted.

If venus complains (in `venus.log` about not being able to open `/dev/cfs0`, it is because `/coda` is still mounted.

```
# umount /coda
```

Another reason for not restarting is that another copy of venus is still around, and venus is unable to open it's network socket. In this case there will be a message in `venus.log` stating that RPC2_CommInit has failed.

5.2.6 Venus doesn't start.

A reason is that you do not have the correct kernel module. This can be tested by inserting the module by hand, and then listing the available modules. 'coda' should show up in that listing. Otherwise reinstall (or recompile) a new module.

```
# depmod -a
# insmod coda.o
# lsmod
Module                  Size  Used by
coda                   50488   2
```

If the kernel-module can be loaded without errors, check `venus.log`. A message stating 'Cannot get rootvolume name' indicated either a misconfigured server or the codasrv/codasrv-se ports are not defined in `/etc/services`, which should contain the following entries. See above for the entries needed.

5.2.7 I'm disconnected and Venus doesn't start

Put the hostnames of your servers in `/etc/hosts`.

5.2.8 I cannot get tokens while disconnected.

Take vacation until we release version 5.2 of Coda. We will add this feature.

5.2.9 Hoard doesn't work

Make sure you have version 5.0 of Coda or later. Before you can hoard you must make sure that:

- You started Venus with the flag `-primaryuser "youruid"`
- You have tokens

5.3 Server Problems

5.3.1 The server crashed and prints messages about "AllocViaWrapAround"

This happens when you have a resolution log that is full. In the `SrvLog` file you will usually be able to see which volume is affected, take down it's volume id (you may need to consult `/vice/vol/VRList` on the SCM to do this. Kill the dead (zombied) server, and restart it. The moment it is up you do:

```
filcon isolate -s "this server" # to prevent clients from again
                                # overwriting the log
volutil setlogparms "volid" reson 4 logsize 16384
filcon clear -s "this server"
```

Unless you do "huge" things 16k will be plenty.

5.3.2 server doesn't start due to salvaging problems

If this happens you have several options. If the server has crashed during salvaging it will not come up by trying again, you must either repair the damaged volume or not attach that volume.

Not attaching the volume is done as follows. Find the volume id of the damaged volume in the SrvLog. Create a file named `/vice/vol/skipsalvage` with the lines:

```
1
0xdd000123
```

Here `1` indicates that a single volume is to be skipped and `0xdd000123` is the volume id of the replica that should not be attached. If this volume is a replicated volume, take all replicas offline, since otherwise the clients will get very confused.

You can also try to repair the volume with `norton`. Norton is invoked as:

```
norton LOG DATA DATA-SIZE
```

These parameters can be found in /vice/srv.conf.

The Norton manual pages give details about norton's operation and there is online guidance available which is possibly more helpful.

NOTES:

1. Often corruption is replicated. This means that if you find a server has crashed and does not want to salvage a volume, your other replicas may suffer the same fate: the risk is that you may have to go back to tape (you do make tapes, right?). Therefore **first** copy out good data from the available replicas, **then** attend to repairing or skipping them in salvage.

2. Very often you have to take both a volume and its most recent clone (generated during backup) offline, since corruption in a volume is inherited by the clone.

3. If you find that a replica of a volume is corrupt, *do not attempt to merely replace that replica*. We have found that this corrupts the volume databases. It is better to make a new replicated volume and copy of the data from the healthy replicas (keep the server with the bad replica down).

5.3.3 How to restore a backup from tape

Tuesday I lost my email folder - the whole volume *moose:braam.life* was corrupted on server *moose*, it wouldn't salvage. Here is how I got it back.

First I tried mounting `moose.braam.life.0.backup` but this was corrupted too.

On the SCM in `/vice/vol/VRList` I found the replicated volume number `f0000427` and the volume number `ce000011` (ficitious) for the volume.

I logged in as root to bison, our backup controlller. I read the backuplog for Tuesday morning in /vice/backuplogs/backuplog.DATE and saw that the incremental dump for August 31st had been fine. At the end of that log, I saw the name `f0000427.ce000011` listed as dumped under /backup (a mere symlink) and /backup2 as spool directory with the actual file. The backup log almost shows how to move the tape to the correct place and invoke restore:

```
cd /backup2
mt -f /dev/nst0 rewind
restore -b 500 -f /dev/nst0 -s 3 -i
```

The `-s 3` option varies according to which */backup[123]* volume the backup is restored from. This invokes the restore command. Typing help allowed me to add then extract the file I wanted. It took a little while before the file was back. From the restore prompt do:

```
    restore> cd 31Aug1998
    restore> add viotti.coda.cs.cmu.edu-f0000427.ce000011
    restore> extract
    Specify volume #: 1
<verb>

In /vice/db/dumplist I saw that the last full backup had been on
Friday Aug28. I went to the machine room and inserted that tape
(recent tapes are above bison).  This time f0000427.ce000011 was a
```

```
200MB file (the last full dump) in /backup3. I extract the file as
above.

Then I merged the two dumps:

<verb>
     merge /restore/peter.mail /backup2/28Aug1998/f0000427.ce000011 \
           /backup3/31Aug1998/f0000427.ce000011
```

This took a minute or two to create /restore/peter.mail. Now all that was needed was to upload that to a volume:

```
volutil -h moose restore /restore/peter.mail /vicepa vio:braam.mail.restored
```

Back to the SCM, to update the volume databases:

```
bldvldb.sh viotti
```

Now I could mount the restored volume:

```
cfs mkm restored-mail vio:braam.mail.restored
```

and copy it into a read write volume using cpio or tar.

5.3.4 createvol_rep reports RPC2_NOBINDING.

When trying to create volumes, and createvol_rep reports RPC2_NOBINDING, it is an indication that the server is not (yet) accepting connections.

It is useful to look at `/vice/srv/SrvLog`, the server performs the equivalent of `fsck` on startup, which might take some time. Only when the server logs 'Fileserver Started' in SrvLog, it starts accepting incoming connections.

Another reason is that an old server is still around, blocking the new server from accessing the network ports.

5.3.5 RPC2_DUPLICATESERVER in the rpc2portmap/auth2 logs

Some process has the UDP port open which rpc2portmap or auth2 is trying to obtain. In most cases this is an already running copy of rpc2portmap or auth2. Kill all running copies of the program in question and restart them.

5.3.6 Server crashed shortly after updating files in `/vice/db`.

Servers can crash when they are given inconsistent or bad data-files. You should check whether `updateclnt` and `updatesrv` are both running on the SCM and the machine that has crashed. You can kill and restart them. Then restart `codasrv` and it should come up.

5.3.7 Users cannot authenticate or created volumes are not mountable.

Check whether auth2, updateclnt, and updatesrv are running on all fileservers. Also check their logfiles for possible errors.

5.4 Disconnections.

As most common problems are related to the semantical differences arising as a result of 'involuntary' disconnections, this section contains some background information of why volumes become disconnected or write-disconnected. And how to get them to reconnect again.

Volume is fully disconnected.

There are several reasons why a coda client may have disconnected some or all volumes from an accessible server.

- Pending reintegration.When modifications have been made to the volume in disconnected mode, the client will not reconnected the volume until all changes have been reintegrated. Also, reintegration will not occur without proper user authentication tokens. Furthermore, reintegration is suspended as long as there are objects in conflict.

  The most important item here is to have a **codacon** process running, since it will give up-to-date information on what venus is doing. Venus will inform the user about missing coda authentication tokens, *'Reintegration: pending tokens for user <uid>'*. In this case the user should authenticate himself using the **clog** command.

  Conflicts, which require us to use the **repair** tool, are conveyed using the *'local object <pathname> inconsistent'* message. Otherwise codacon should show messages about *backfetches*, and how many modifications were successfully reintegrated.

- Access permissions.The client may also disconnect when a servers reports an error to an operation, when according to the client this is a valid operation. Causes for this are authentication failure; check tokens using **ctokens** and optionally obtain new tokens using **clog**. Or inconsistencies between the data cached on the client and the actual data stored on the server; this will reveal itself as an inconsistent object during subsequent reintegration.

- Lost connections.Sometimes the client does not receive a prompt reply from an accessible server, and marks the server as dead. This will ofcourse disconnect the volume if the last server is lost. Once every five minutes, the client automatically verifies connectivity with all known servers, and can thus recover from lost connections. However, this action can also be triggered by the user by excecuting the **cfs checkservers** command.

  If **cfs checkservers** reports that servers are unreachable, it might be interesting to check with **cmon** if the server is responding at all, since we might be faced with a crashed server. When a server was considered unreachable, but is successfully contacted after 'cfs checkservers', reintegration will automatically start (when a user has tokens, and there are no inconsistencies).

Volume is write-disconnected.

Write-disconnected operation is used as often as *weakly connected mode* to describe this volume state, and they are effectively the same. This is the special situation where a client observes a weak connectivity with a server, and therefore forces the associated volumes in weakly connected mode. Weakly connected volumes postpone writing to the server to significantly reduce waiting on a slow network connection. Read operations are still serviced by the local cache and the servers, as in fully connected mode. Which is why this mode of operation is also called write-disconnected operation.

The write operations are effectively a continuous reintegration (*trickle-reintegration*) in the background. This mode, therefore, requires users to be authenticated and gives more chance for possible file conflicts. The following points are several reasons for write-disconnected operation.

- Weak network connectivity.Venus uses bandwidth estimates made by the rpc2 communication layer to decide on the quality of the network connection with the servers. As soon as the connectivity to one of the servers drops to below the weakly connected treshhold (currently 50 KB/s), it will force all volumes associated with that server into weakly-connected mode. The **cfs wr** command can be used to force the volumes back into fully connected mode, and immediately reintegrate all changes.

When the user was not authenticated, or conflicts were created during the write-disconnected operation, the user must first obtain proper authentication tokens or repair any inconsistent objects before the volume becomes fully connected again. Here again **codacon** is an invaluable tool for obtaining insight into the client's behaviour.

- User requested write-disconnect mode.Users can ask venus to force volumes in write-disconnected mode, exchanging high consistency for significantly improved performance. By using the `-age` and `-time` flags on the **cfs wd** commandline, some control is given about the speed at which venus performs the trickle-reintegration. For instance, to perform the trickle-reintegrate more quickly than the default, where only mutations to the filesystem older than 15 minutes are reintegrated. You could use **cfs wd -age 5**, which will reintegrate all mutations older than 5 seconds.
- Pending reintegration.When a volume is write-disconnected, it will stay write-disconnected until a user properly authenticates using **clog**.

5.5 Advanced Troubleshooting

5.5.1 with rpc2tcpdump

`rpc2tcpdump` is the regular tcpdump, which is modified to decode rpc2 protocol headers. This makes it a very useful tool for analyzing why programs fail to work.

All traffic between `venus` and the coda servers can be viewed using the following command.

```
# tcpdump -s120 -Trpc2 port venus or port venus-se
```

To identify problems with `clog`, for instance which server it is trying to get tokens from.

```
# tcpdump -s120 -Trpc2 port codaauth
```

5.5.2 debugging with gdb

To be able to debug programs that use RVM, most coda related application will go into an endless sleep when something goes really wrong. They print their process-id in the log (f.i. `venus.log` or `SrvLog`), and a user can attach a debugger to the crashed, but still running, program.

```
# gdb /usr/sbin/venus ‘pidof venus‘
```

This makes it possible to get a stack backtrace (`where`), go to a specific stack frame (`frame` <x>), or view the contents of variables, (`print` <varname>). By installing the coda sources in same place as where the binaries were initially built from, it is possible to view the surrounding code fragment from within the debugger using the `list` command.

When using RedHat Linux rpms, you can install the sources in the right place by installing the coda source rpm file.

```
# rpm -i coda-x.x.x.src.rpm
# rpm -bp /usr/src/redhat/SPECS/coda.spec
```

On other platforms look at the paths reported in the backtrace and unpack the source tarball in the correct place.

```
(gdb) where
#0  CommInit () at /usr/local/src/coda-4.6.5/coda-src/venus/comm.cc:175
#1  0x80fa8c3 in main (argc=1, argv=0xbffffda4)
    at /usr/local/src/coda-4.6.5/coda-src/venus/venus.cc:168
(gdb) quit
# cd /usr/local/src
# tar -xvzf coda-4.6.5.tgz
```

5.6 Troubleshooting on Windows 95

5.6.1 Common problems

Unable to mount Coda.

It is only possible to mount coda when `relay.exe` and `venus.exe` are running. Do not type `mount.exe n:` before venus printed the message

```
venus starting...
```

Unable to shutdown Windows95.

This occurs when the device `codadev.vxd` couldn't be unloaded on the request issued by typing `unmount.exe`. `relay.exe` printed the message

```
UNLOAD RESULT 0
UNLOAD RESULT 8
```

when the request was successful. There is no way out of this. Just reboot your machine.

I cannot reboot Windows95 and I think it is due to the VXDs loaded for Coda.

Boot your System in DOS mode by pressing F8 on boot time. Cd to the windows directory and type `edit system.ini`. In the section `[enh386]` you will find the entries

```
device=c:\usr\coda\bin\mmap.vxd
device=c:\usr\coda\bin\mcstub.vxd
```

Comment them out by using a `;` in front of the lines. Try to restart Windows again.

How can I find out why `relay.exe` crashed.

`relay.exe` is a very tiny program which hands requests from the file system driver up to `venus.exe` and the other way around. Crashes in `relay.exe` are most likely due to buffer overflows or null pointer referencing. `relay.exe` prints debug information in a file called `relay.log` which can be found in the directory where relay was started from.

When `relay.exe` crashed, the file system driver is still loaded. Just restart `relay.exe`. It might pretty well not be possible to unmount the system by calling `unmount.exe`.

How can I find out why `venus.exe` crashed.

See troubleshooting venus. When this happens it is possible to restart venus. The file system doesn't need to be unmounted before.

How can I find out more about what has happend

Look in the file `c:\vxd.log`. The file system driver `codadev.vxd` prints information about all requests and answers in this file. Check the requestnumbers to match with those in `relay.log`.

5.6.2 Restrictions

- `hoard`.exe does not work so far.
- Handling large files (in particular executables) does not work well in a low bandwidth scenario.
- `cfs.exe` uses absolute pathnames so far.
- Long filenames are not supported under DOS environment yet. You can access files, but you need to use the long filenames.
- Reintegration after disconnection does not work at present.

6 Building Coda

As a file system, Coda has several major components that need to be built:

- Kernel code on the client
- The cache manager Venus on the client
- The fileserver
- Utilities for client and server administration.

6.1 Building on Linux

6.1.1 Building the kernel module

We now have a reaonably flexible method to build kernel modules. You can build the module for a kernel which you are not running at the time of the build.

First of all get the linux-coda-?.?.?.tgz archive from `ftp://ftp.coda.cs.cmu.edu/pub/coda/src/`. You can unpack this anywhere on your target system.

Prepare the kernel tree

You do need a kernel tree handy, to give the module header information. To get ready:

```
make oldconfig
make dep
```

In the top directory of linux-coda build the coda.o module:

```
make config  --- answer the questions
make coda.o
su
make install
depmod -a
```

Notes

- If you build for a running kernel, you must still have a source tree for that Linux release, otherwise the headers cannot be found. However, that source tree doesn't need modversions.h etc.
- We don't actively maintain 2.0 code anymore. The 2.1 code is preferred, but not yet perfect.

6.1.2 Building the userlevel code

Coda builds most easily on *glibc* systems. We do not actively maintain the `libc` variant anymore. You will need several libraries to link the Coda binaries, and include files from some of the lib???-devel packages. Readline and termcap are probably the most important ones.

To build:

- Unpack `coda-?.?.?.tgz`.
- ./configure –prefix=/usr
- make (make coda for versions 4.6 and older)
- make client-install
- make server-install

6.2 Building on FreeBSD

6.2.1 Building a Whole Kernel

The Coda kernel files are in the FreeBSD -current kernel sources. All you need to do to get a Coda capable kernel is to build a configuration that includes Coda. The lines to enable Coda are not in the GENERIC. There are several Coda lines in the LINT file that you need to copy. In the i386/conf directory type:

```
grep -i coda LINT
```

It currently gives:

```
# Coda stuff:
options         CODA                    #CODA filesystem.
pseudo-device   vcoda   4               #coda minicache <-> venus comm.
```

Add these lines (or whatever the `grep` yields) to the GENERIC kernel configuration or your standard personal configuration. You might also want to turn on DDB and BPF.

I'll assume you call the new config file CODA. You type:

```
config CODA
```

Then `cd` to the build area and do a `make`:

```
cd ../../compile/CODA
make
```

Finally, as root install the kernel with:

```
make install
```

6.2.2 Building an LKM

Alternatively, you could just build an lkm for Coda and insert it into the kernel. To do that you:

```
cd /usr/lkm/coda
```

and type:

```
make
make install
```

This will build coda_mod.o and copy it to /lkm. You load it by typing:

```
modload -v -e coda_mod -o /var/run/lkm.coda /lkm/coda_mod.o
```

Remember, when using an lkm:

- *You must boot /kernel not /kernel.old or /kernel.<something>*
- *There are no symbols available for debugging.*

6.2.3 Building Coda: The Package Way

```
cd /usr/ports/net/coda_client
make install
cd /usr/ports/net/coda_server
make install
```

NOTE: The client and server each need about 85Meg for the work area and another 15Meg to install. They are compiled -g for debugging.

6.2.4 Building Coda: The Bleading Edge Way

First, you need several external packages to build coda: `gnu make`, `gdbm`, `readline`, and `perl`.

You can grab any of our (latest) sources from *ftp://ftp.coda.cs.cmu.edu/pub/coda/src* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/src>`. For this example, I'll assume that you have obtained sources for Coda release 4.6.6. **NOTE:** On our ftp site there are also beta, alpha, and less stable source releases available for the daring.

Untar the source into your favorite build area. (I am assuming this is `/usr/coda`.)

```
tar zxfv coda-4.6.6.tgz -C /usr/coda
```

`Cd` to `/usr/coda` and

```
mkdir obj
```

This is where we will build the binaries. Now `cd` into `obj` and type:

```
../coda-4.6.6/configure
```

This will use the gnu configure system to configure and to shadow the sources that were installed in /usr/coda/coda-4.6.6.

Finally, *assuming you named the gnu make, gmake*, type:

```
gmake coda
```

To install the client suite type:

```
gmake client-install
```

To install the server suite type:

```
gmake server-install
```

6.3 Building on NetBSD

6.3.1 Building a Whole Kernel

The Coda kernel files are in the NetBSD -current kernel sources. All you need to do to get a Coda capable kernel is to build a configuration that includes Coda. The lines to enable Coda may not be in your configuration. They should be in GENERIC, though they may be turned off. Enable them and/or copy them to your configuration file. They should look like.

```
file-system     CODA                    # Coda File System
pseudo-device   vcoda           4       # coda minicache <-> venus comm.
```

But whatever is in GENERIC will be more up to date.

I'll assume you call the new config file CODA. You type:

```
config CODA
```

Then `cd` to the build area and do a `make`

```
cd ../compile/CODA
make
```

Then just copy `netbsd` to root.

6.3.2 Building an LKM

Alternatively, you could just build an lkm for Coda and insert it into the kernel. To do that you:

```
cd /usr/sys/lkm/vfs/coda
```

and type:

```
make
make install
```

This will build coda.o and copy it to /usr/lkm. You load it by typing:

```
modload -v -e coda_lkmentry -o /var/run/lkm.coda /usr/lkm/coda.o
```

Remember, when using an lkm:

- *You must boot /netbsd not /netbsd.old or /netbsd.<something>*
- *There are no symbols available for debugging.*

6.3.3 Building Coda: The Package Way

```
cd /usr/pkgsrc/net/coda_client
make install
cd /usr/pkgsrc/net/coda_server
make install
```

NOTE: The client and server each need about 85Meg for the work area and another 15Meg to install. They are compiled -g for debugging.

6.3.4 Building Coda: The Bleading Edge Way

First, you need several external packages to build coda: `gnu make`, `gdbm`, `readline`, and `perl`.

You can grab any of our (latest) sources from *ftp://ftp.coda.cs.cmu.edu/pub/coda/src* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/src>`. For this example, I'll assume that you have obtained sources for Coda release 4.6.6. **NOTE:** On our ftp site there are also beta, alpha, and less stable source releases available for the daring.

Untar the source into your favorite build area. (I am assuming this is `/usr/coda`.)

```
tar zxfv coda-4.6.6.tgz -C /usr/coda
```

`Cd` to `/usr/coda` and

```
mkdir obj
```

This is where we will build the binaries. Now `cd` into `obj` and type:

```
../coda-4.6.6/configure
```

This will use the gnu configure system to configure and to shadow the sources that were installed in /usr/coda/coda-4.6.6.

Finally, *assuming you named the gnu make, gmake*, type:

```
gmake coda
```

To install the client suite type:

```
gmake client-install
```

To install the server suite type:

```
gmake server-install
```

6.4 Building a Client on Windows 95

WARNING: The software provided to run Coda on Windows 95 is an early pre-alpha snapshot, made available to those interested. Coda runs kernel level code and priviliged processes which can cause damage to your system. Backup all data before playing with Coda.

Coda for Windows was made possible through a major effort by Michael Callahan `mjc@stelias.com`. Michael extended the DOS protected mode environment with memory mapping and TCP/IP socket support (this involves both Windows kernel support as well as low level support in DJGPP start libraries). He also wrote a kernel module for Coda. Michael's work was further debugged, extended and packaged by the Coda team.

Short rationale

Why DOS applications?? It would seem more straightforward to implement the Coda client cache manager, a user level program named Venus, as a Win32 application. Sadly on Windows 95 we ran into the following (fairly well known) problem. When a user application calls a Win32 file system call, the application may acquire a mutex in a win16 system dll. The request should reach the kernel, and make its way up to Venus. Venus is then unable to service the request because it cannot grab the mutex. Deadlock results. (See Schulman's, Unauthorized Windows 95, IDG.)

Implementing all of Venus as a kernel level cache manager seemed an invitation for disaster. Instead, by running the cache manager in a virtual DOS machine, as a DOS Protected Mode Interface application, one can bypass these problems, since such applications do not share the Windows dynamic libraries which gain the mutex. The price of following this path is high. There was no freely available socket support for such DPMI applications and no memory mapping support. These are provided by Michaels VXD's and standard library calls are now incorporated in the DJGPP toolchain.

6.4.1 Building the user level code

What do you need:

1. a Linux machine with a few 100M of free space.
2. download the tool chains made available under: *ftp://ftp.coda.cs.cmu.edu/pub/tools/{95,win32}* `<ftp://ftp.coda.cs.cmu.edu/pub/tools/>`
3. You need the rpms for:
   - djgpp package.
   - djgpp-win95ext.
   - cygwin.
   - libgdbm-nt.
   - gdb-djgpp a remote debugging environment for DPMI applications.
4. It is highly recommended to set up Samba to share Linux generated binaries with the Windows box.

Get the latest tarball of Coda sources. Example: *ftp://ftp.coda.cs.cmu.edu/pub/coda/src/coda-?.?.?.tgz.* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/src/>`

You can rebuild Venus under Linux with

```
./configure --host=windows95
make
```

and the client utilities and server with

```
./configure --host=nt
```

The Venus built here is for NT and is not usable on Win95.

6.4.2 Building the kernel level code (VXDs)

What do you need:

- Visual C++ 5.0 (with Internet Explorer).
- MS assembler 6.11.
- The Win95 DDK (comes with professional MSDN).
- You also need to download a collection of header files vxdtdi.zip for networking from. *ftp://ftp.microsoft.com/developr/drg/WinSock/MS-Extensions/VXDTDI.ZIP.* `<ftp://ftp.microsoft.com/developr/drg/WinSock/MS-Extensions/VXDTDI.ZIP>`

Note: The 95DDK needs the VC2.0 linker, which is on the 95DDK in the directory MSVC20 (also get the files Link.err and DBI.DLL, that will be needed by the linker). You'll also need the latest MS assembler 6.11 which is on the 95DDK in the directory MASM611C.

The sources for the VxD's you will build are in: *ftp://ftp.coda.cs.cmu.edu/pub/coda/src/vc95.zip.* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/src/vc95.zip>` To build sock.vxd you'll need the header files provided by microsoft and can be found in VXDTDI.ZIP. You'll have to learn how to build VxDs, which can be done from Developer Studio:

- Change the path of the Linker in all .mak files
- In the MS-Studio environment (options) add:
  - executable path of the assembler 6.11
  - include path of the 95DDK inc32 (Inc32)
  - include path of the 95DDK inc16 (Inc16)
  - lib path of the 95DDK (lib)
  - include path where VXDTDI Headers are
  - include path of the 95DDK (Block/Inc)

Use the Developer Studio Project "95projects". Build the VXDs one by one.

(If you fancy finishing off Michael's nice codastart project, make sure to let us know! It is not essential for now.)

6.4.3 Installing Coda after build

- Create the directories C:\usr\coda\{bin, etc, spool, venus.cache}.
- Put Venus.exe Relay.exe, Mount.exe, Unmount.exe, sock.vxd, mc.vxd, mcstub.vxd, mmap.vxd in the directory called C:\usr\coda\bin.
- Create the file C:\usr\coda\etc\vstab which includes a line like

  ```
  /coda:/dev/cfs0:<ip adress of server 1>,..,<ip adress of server n>,<cachesize>:1
  ```

  Use

  ```
  /coda:/dev/cfs0:128.2.209.213:100000:1
  ```

  to work with the CMU-testserver.

- Create the file C:\usr\coda\venus.cache\myhost which includes your IP adress.
- Add the two VXDs mcstub.vxd, mmap.vxd to the C:\Windows\System.ini file in the section 386Enh by typing "device=C:\usr\coda\bin\mcstub.vxd".
- Add the path `c:\usr\coda\bin` to the PATH environment variable in `c:\autoexec.bat`.
- Restart the computer.
- The programs clog.exe, ctokens.exe, cunlog.exe, cfs.exe, hoard.exe, codacon.exe, venus-setup.bat will be build with the coda server for NT.

You are now at the point where the installer would have left you. To start Coda read the section 3.3 (Windows95: Starting and Configuring a Coda client).

6.5 Building and installing a Coda Server for NT

Note: At present there is no client kernel source code available yet. We will release some very experimental binaries soon. When are a little further with it, we will release the sources, of course.

You can however build a Venus (which you can with the supplied kernel Coda FSD for NT) and you can build a server.

Coda has alpha support for Coda servers running on NT and Windows 95. The Win32 binaries are constructed using the Cygnus Win32 kit, which effectively translates Unix system calls to Win32 calls.

We build the Win32 binaries on Linux workstations using a cross compiler, just for our convenience. At the moment this is not for the faint of heart, since quite a few things have to be done by hand. Of course we encourage playing and will try to help and fix bugs.

6.5.1 The build process

You will need:

1. Get the cross compiling kit: *ftp://ftp.coda.cs.cmu.edu/pub/tools/win32/cygwin-b19_glibc-2.i386.rpm.* `<ftp://ftp.coda.cs.cmu.edu/pub/tools/win32/cygwin-b19_glibc-2.i386.rpm>`
2. *ftp://ftp.coda.cs.cmu.edu/pub/tools/win32/libgdbm-nt-1.7.3-2.i386.rpm.* `<ftp://ftp.coda.cs.cmu.edu/pub/tools/win32/libgdbm-nt-1.7.3-2.i386.rpm>` (These are rpms for RedHat 5.0. Sources are available of course.).
3. Get the coda-?.?.? tar ball *ftp://ftp.coda.cs.cmu.edu/pub/coda/src.* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/src>`
4. Unpack & build

```
./configure --host=nt
make coda
```

5. drink coffee, ignore or, better even, fix the compiler warnings.

Native building should be possible; you'll have to make a few changes to configs/coda.m4 and create a new configs/Makeconfig.ntnative file; then run autoconf and get cracking. But don't you dare send us patches with ^M characters made by Windoze machines!

6.5.2 Manual configuration after building

1. Install into an area for NT binaries on your Linux box still (drink coffee, ignore or better even fix the compiler warnings)

   Put the binaries in a suitable place using: make BINDIR=TARGET/bin SBINDIR=TARGET/bin server-install

2. get samba running on that RH machine and export the build area for Coda.

3. get a patch to *select* and replace

   ```
   C:\Cygnus\B19\H-i386-cygwin32\bin\cygwinb19.dll
   ```

   with the one from: *ftp://ftp.coda.cs.cmu.edu/pub/coda/support/nt-patches* `<ftp://ftp.coda.cs.cmu.edu/pub/coda/support/nt-patches>`

4. get bash going. It can be started from the "Start" menu.

5. organise your mountpoints for Cywin. You will find the root volume mounted on `c:\`. Change this to `c:\coda` by typing:

   ```
   umount /
   mount -b C:\coda  /
   ```

   The -b, for binary mounting is vital.

6. For your convenience add the entry

   ```
   SET HOME=C:\Coda
   ```

   in the file `cygnus.bat` somewhere in the Cygwin path.

   You can also create the file `.bashrc` in the `c:\coda` directory which gets called, when the Cygwin shell gets started. We set the export path (described below) and cd to the home directory that way.

7. Create some directories, create `/bin/sh` and export the path by typing:

   ```
   mkdir /bin /vice /rvm /vicepa /tmp
   touch /rvm/LOG
   touch /rvm/DATA
   cp //C/Cygnus/B19/H-i386-cygwin32/bin/bash /bin/sh
   export PATH=/bin:$PATH
   ```

8. Install your Coda stuff build as described above. Suppose the coda-?.?.? area is a mapped drive on the NT machine with drive letter H: type in bash:

   ```
   cd //H/
   cp * /bin
   ```

   or use the Windows Explorer. The binary files might not have the extension `.exe`. In that case just change the name, but distinguish between bash shell scripts and binaries.

You are now at the point where the installer would have left you. To configure and start the server read the section 3.4 (Codaserv – the server).