The Vice 2 File Server

This document describes the distributed *Vice* 2 file server. It explains how to install it from the tape, and gives the operating instructions to keep it running. It also describes what has not been distributed at this time, and how to get those features as well. There is a short section on trouble-shooting and bug reporting and a list of the major things that are still being worked on.

There are also a few papers of interest at the end.

The first is a description of protection in *vice* called **File and Directory Protection.**

The next is a description of the fs program, which allows setting and displaying information about the file server called **fs: A File System Program.**

Then there is a description of up a program used to update files based on their dates called **Up: A Backup Facility.**

After this is a brief description of how *vice* works from the user point of view called **Vice Overview.**

And finally there is a brief overview of how the various parts of the file system are controlled called **Vice Operation Overview.**

What you have

The materials that come with this distribution are:

This document

2 streaming tapes containing a dump of hd0a and hd0g from a 70 megabyte disk of an RT file server

1 streaming tape containing the workstation binaries

1 streaming tape containing the sources

In addition you will need the two diskettes that come with the 4.2 distribution called *Installation Boot* and *Miniroot.*

Server Installation

Installation assumes that you have an with a 70 megabyte disk for the root and some number of other disks to hold the data the file system stores. The first two streaming tapes should be restored to hd0a and hd0g. The

ANOVA for Total Quality

source	df	mean square	F	prob.
Subjects	14	3.3317		NS
Conditions	2	37.9556	4.28	<.02
Subject X Condition	28	8.8603		

Newman-Keuls Tests of Significance
for Total Quality

	Advanced Workstation	Pen and Paper	PC
Advanced Workstation	———	.52	3.828*
Pen and Paper		———	3.307**
PC			———

*p<.05
**p <.10

root disk will mount the hd0a partition as /, use the hd0b partition as swap and mount the hd0g partition as /usr.

At this point you should restore the tape, which is in dump format.

To do this, you boot the diskette marked *installation BOOT* and when it prompts you insert the *Installation MINIROOT* diskette and press enter. This will then bring up a Unix system. At the shell prompt enter *restore.tape root user.* This will initialize the hd0a and hd0g partitions and load them from the tapes. When it requests the boot tape insert the one for the hd0a partition, and when it requests the user tape, insert the one for the hd0g partition. At this point you should use the *newfs* command to initialize the partitions that will be used to store *vice* data. To do this cd */etc* and issue the command:

newfs /dev/hd*n*c *disktype*

where:
the *n* in hd*n*c is 1 or 2 for the disk to use
disktype is the type of disk you are inititializing.

e.g. to initialize a 70r disk on the 1 drive issue the command:

```
newfs /dev/hd1c hd70r
```

This should be done for all disks that are going to be used for *vice* partitions.

Each disk's "c" parition should then be added to */etc/fstab* and mounted on a *vice* directory. All partitions that *vice* uses start with the string */vicep*. In the above example to make the system recognize the 70 megabyte partition the line:

/dev/hd1c:/vicepa:rw:1:2

should be added to */etc/fstab*.

One other note is that if you are going to be using the system on an ethernet instead of on a token ring, the file */UseEthernet* should be added to the root. This is to tell the */etc/rc* script to initialize the ethernet instead of the token ring. This can be accomplished by issuing the commands:

```
/etc/mount /dev/hd0a /mnt
echo >/mnt/UseEthernet
```

This distribution assumes that the server will be called *vice1*. Attempting to change this name is very difficult. The machine name of this server and

its IP address must also be added to */etc/hosts* (the file server needs only to know its own IP address and that of other file servers, so it is not essential to have a complete host table). Probably the best way to get this is to add an entry for vice1 to */etc/hosts* and add it to the server by *tar*ing it to a diskette and putting it on the server from the diskette. There is a */etc/hosts.example* in the shipped system that has entries for a couple of servers and workstations. This example uses IP addresses reserved for C-MU so it should not be used directly.

There will be two users on the system. Root will have the password andy150 and admin will have the password install.

The system is now ready to be brought up as a running file server. Type */etc/reboot*.

Setting Up Your Workstation

To run *venus* on a workstation, one needs to make a few changes to that workstation's operating environment. Specifically, one needs the following:

A new kernel, compiled with the *vice* hooks (both VIRTUE and VICE).

A directory for *venus* to use for a cache. This directory should be on a partition containing at least 10 megabytes of space.

Several new entries in /etc/services, one naming the **filesrv** service, and one naming the **venus.itc** service. The first is the service name provided by the file server for receiving requests, and the second is the service name provided by *venus* for debugging and monitoring information. One called **auth2** that is used by the authentication code. Three (**vexec, vlogin, vshell**) that are used to pass tokens between machines to accomplish remote authentication. And one more **opcons** for the monitor server.

Three devices, /dev/fs0, /dev/fs1 and /dev/fs2, for communications with the file system server process, **venus**.

An empty directory, usually named **/vice**, on which to mount the remote file system.

A copy of the program named **venus2**, used for connecting to the file server.

A copy of the program named **fs**, used for doing special operations to the file system that have no standard Unix counterparts, such as setting the access control list on a directory. This program may itself be stored in *Vice*.

The *passwd* file generated on the server with the *bldpass.sh* command.

An updated */etc/hosts* that contains the addresses of the workstations and the hosts.

Detailed instructions on how to build a *vice* hooks kernel are given later on in this document, and will not be repeated here. Once the kernel has been prepared, it should be installed as */vmunix* on your workstation and booted.

After you have a workstation with the modified /vmunix on your workstation, you can load in all of the programs as follows:

```
cd /
tar xvf /dev/st0
```

This will cause the following to occur:

/bin replace *login* with one that understands *vice* authentication.

/etc add *netd* and *netd.conf* for remote services
add *vstab*
put in a *hosts.example* - you should put your own hosts file here
add *services.add* - these services should be added to your services file. **Note:** the service numbers are not important, but ensure that you have the same *services* file on the file servers as on the workstations.

/usr add *andrew*, *include* and *lib* directories if they do not exist
andrew - see description below
include - update copy of *include/sys/inode.h* which is needed to compile the kernel until the next release from Palo Alto
lib add *netd* directory used by */etc/netd*

If you do not have a /usr/andrew one will be created and the following will be added to it:

bin *passwd*, *su* programs that understand *vice* authentication
log, *unlog* to set and reset authentication

up a program to do program copies based on dates
fs a program to deal with special *vice* file server requests
vopcon a program to monitor the status of the fiel servers

etc *checkvenus*, *startvenus*, *venus2* the *venus* daemon and programs to start it and keep it up to date on the workstation

lib *consoles/Default.Vopcon* contains a template to run the *vopcon* program

The reason for the programs is:

log, passwd, login, su, unlog, the programs in etc and netd.
These are the programs that are modified to deal with the authentication server and to pass tokens between machines on operations such as rsh.

venus2, startvenus and checkvenus
These are the programs to bring up *venus* and in the case of checkvenus to fetch a new copy before *venus* starts on a workstation

up and fs
These are programs to use to do things in the *vice* file system that are different from the standard Unix file system.

vopcon and lib
These are programs that are needed to run a file server monitor on a workstation that has the Andrew window manager.

After this the following modifications must also be made.

Merge the services in */etc/services.add* with your own copy of */etc/services*. Build a */etc/hosts* file that contains the file servers as well as your own workstations. /etc/hosts.example is just an example of a host table. Since it contains addresses assigned to C-MU you should never actually use it but use a host table that contains your own assigned addresses.

At C-MU we use the directory /user/venus2.cache as the *venus* cache directory. *Mkdir* this directory and give it any protection you desire; *venus* does not need any particular access to it since it runs as root anyway. When *venus* is operating, this directory will be filled with many files. Several hundred will have names such as V0, V139, etc. and represent the files actually cached by *venus*. There is a file named CacheInfo, which contains some information on what files are actually cached by *venus*. In

order to completely flush the *venus* cache, it is only necessary to delete CacheInfo; one need not explicitly delete all of the V* files. ***Under no circumstances should you modify the contents of this directory while venus is running, unless you enjoy kernel panics.***

The new entries in /etc/services looks like this, on our file servers and workstations:

```
filesrv   2001/tcp
fsbulk        2002/tcp
filesrv2  2003/tcp
venus.itc 2106/tcp
vexec     712/tcp
vlogin    713/tcp
vshell    714/tcp
updsrv        2114/tcp
auth2     2101/tcp
opcons    2115/tcp
```

Any ports may be used for these services, but it is imperative that *tget* agree on the file server and the workstations. A list of these is included in the file */etc/services.add* that was brought in on the *tar* command. The list also contains the services needed by PC server.

The three devices, /dev/fs0, 1 and 2, are used for communications between *venus* and the kernel. Each device is used by one *venus* process, so having three devices puts a limit of three concurrent *venus* processes. Under normal circumstances, one only runs one *venus* process on a workstation at a time. These devices are character devices, with major device number 16 (on our kernel), and minor device numbers 0, 1 and 2, respectively. Their protection mode must allow reading and writing by root. The command

```
mknod /dev/fs1 c 16 1
```

for example, can be used to create the device /dev/fs1 on a *vice* kernel.

The empty directory /vice can be created with the standard *mkdir* command.

The final phase of installing *venus* involves placing the venus2 object file on your local disk, and placing the commands to start it in your workstation's /etc/rc file. *Venus* takes a large number of command-line options, which may be given in any order.

-h <hostname>
Venus should contact this file server for volume location information. Hostname may also be a list of hosts, separated by commas (and *no spaces*). At least one of these servers must be running for *venus* to locate a volume (a collection of files). One need not list all of a site's file servers in this list, just enough that one is reasonably confident that at least one

is running.

-k <kernel device name>
Venus will use this kernel device (e.g. /dev/fs0) in communicating with the kernel.

-cf <number of cache files>
This switch tells *venus* how many files it may use in the cache. In general, you should not provide this switch, but instead should use the "-c" argument to limit the number of blocks used by the cache.

-cs <number of status cache entries>
This switch tells *venus* how many status cache entries to allocate. Each one uses about 100 bytes of memory. The default is 320. For most configurations, 320 stat cache entries is sufficient.

-c <number of 1K byte units of disk space available to the cache>
This switch tells *venus* how much disk space it can use in the cache directory. For various reasons, one should try to leave about 5-10% of the disk free on the cache device (this is above and beyond the 10 % that the fast Berkeley file system tries to leave free).

-d
This tells *venus* not to fork. Normally *venus* forks and exits immediately after opening the kernel device, but when debugging *venus*, it is important to prevent this, since debuggers deal very poorly with multiple processes.

-r <root volume name>
This switch tells *venus* what volume to use for the root volume. The default is obtained from the file server, but can be overridden here.

-f <cache directory name>
This switch gives the directory *venus* should use for the cache directory.

<mount location>
Any path name not preceded by a "-" character is interpreted as the location at which the kernel device will be mounted. Exactly one of these should be specified on the *venus* command line, for example, "/vice".

You usually will not not need to issue this command directly however. A file called /etc/vstab is used to describe your *venus* configuration. It allows you to specify the parameters that you want to use when you start up your *venus* process and it is also used by other programs for information about your *venus*. The format of the line in /etc/vstab is:

```
mountpoint:device:servers:cachedir:cachesize:parms

where:

mountpoint      the mount location for vice
device          the kernel device name to use
servers         the list of servers to start with
cachedir        the directory name for the venus
     cache
cachesize       the number of 1k byte units of disk
     space for the cache
parms           the input to the -i parm on venus
```

Another way to look at what is in /etc/vstab is:

```
mount location:-k:-h:-f:-c:0
```

where the parameters are what is specified directly on *venus* command.

The format of */etc/vstab* is that the parameters are separated by ":"s and there are no extraneous spaces. A */etc/vstab* was installed with the tar command above.

Then to bring up *venus*, you just place a *startvenus -V /venus2* command in your */etc/rc* for the workstation and it will cause *venus* to be initialized. The syntax for the *startvenus* command is:

startvenus -V *venus* -m *mountcommand*

where:
venus is the pathname of the *venus* daemon
mountcommand is the pathname of the mount command

For example to bring up a *venus* that is */usr/andrew/bin/venus2* with a mount command that is */bin2/mount* you would issue:

```
startvenus -V /usr/andrew/bin/venus2 -m
/bin2/mount
```

Even if you should decide to bring *venus* up with an explicit command in your */etc/rc* file, instead of using *startvenus*, you should still ensure that there is an */etc/vstab* with valid parameters, because other programs also use the information in */etc/vstab*.

Building a Vice Kernel

Building a *vice* kernel is quite simple. At the moment, to be safe, one should set both the VIRTUE and VICE flags in the configuration file, since we have never tested a kernel built with only the VICE flag set. Theoretically, however, the VIRTUE flag is now unnecessary when building a pure file system kernel.

In any event, to build a VICE kernel, copy the GENERIC file in /usr/sys/conf to a file named VICE. Then add the following lines to VICE in the options section

```
options         VICE
options         VIRTUE
options         "VICE_CLIENTS=0"
```

and the following to the pseudo-device section nearer the end

```
pseudo-device   rfs   3
```

Then simply go through the standard kernel generation procedure with this new configuration file.

Adding a user

In order for a new user to be able to use the system, the user must be registered with the *protection system*, and some disk space must be allocated to that user by creating a *volume* for the user's files.

The programs to add users exist on the server in /usr/admin. What is needed is to change the passwd file to add the users and update the group file if you wish to change the group definitions. A protection data base and its index is then generated and placed on line. The authentication data base also must be updated to reflect the passwords for the new users. After this it is also necessary to create volumes for the user.

Fortunately you do not have to do all of this. In /usr/admin there is a passwd file that contains the standard Unix users plus an entry for admin. There is a group data set that only contains the group System:Administrators with admin as the only member. If you want to add groups (or add members to the System:Administrator group) you should modify the groups file. Then all you need to do is supply an adduser list. The adduser list is a list of what the file server needs to know to set up the new users. The list format is:

```
userid      password    server partition      user name
```

The fields are tab separated, and the records are separated by the newline characters.

userid -is the login id of the user
password -is the initial password for that userid
server -is the server that will receive that users volume
partition -is the partition name on the server for the volume (e.g. /vicepb)
user name -is the rest of the record.

Once the adduser list is prepared all you have to do is issue the *bldpass.sh* command to do all of the processing needed.

The *bldpass.sh* script will create the volumes and add the user to the appropriate data bases. It will also emit a list that can be used to initialize the volumes on line. Bldpass.sh makes a few assumptions about users, first it assumes that the user is going to have his volume named user.*uid*. Second it assumes that his home directory is going to be /vice/usr/*uid*. In both of these assumptions uid is the first field in the adduser list. The script that bldpass.sh emits called *initlist* should be run on a workstation. It will take care of initializing each of the user volumes. The initlist contains a call to *inituser.sh* for each user in the adduser list. A default for inituser.sh is shipped, but you should customize it for your location.

The groups file is specific to *vice* (*not* /etc/group) and lists the names of groups, the group number of the group (always a negative number) and the list of the names of the members of that group, as they appear in the passwd file. As distributed the only group is the System:Administrators group. The sample groups file corresponding to this situation can be found in /vice/db/groups. System administrators are the people who are authorized to have special privileges in *vice*: a system administrator can always change an access list, can set or change quotas on volumes, and can change the owners of files. This group list as distributed has a user in it named *admin*. You may also find it useful to add other groups here, to allow common access to files.

Groups and user names, stored in the protection data base, are used by the file server to interpret *access lists*. An access list is stored with each directory, and specifies a list of users and groups and the rights of those users and groups to access that directory and files within the directory. As mentioned, system administrators always have the right to change any access list. For more information on access lists, see the documentation on the *fs* command.

Volume Commands

Volume are the unit of disk allocation within *vice*, and are used to support administration of the system. In a larger system, volume operations are controlled from a central *system control machine*, the software for which is

not being distributed at this time. For this system, the control machine operations are included on the *vice1* server.

To create a volume, log on to the server and cd to /vice/bin. Issue the command :

createvol volname server partition

volname is the name of the volume.
server is the name of the server where the volume will be built
partition is the name of the partition to use /vicep?

At C-MU, we name volumes with names such as "user.joe", where joe is the user's name, for user volumes; we use names like "ibm032.bin" to name system, machine specific, volumes. The name does not matter; however, you will find it useful to have a reasonably consistent naming scheme if your system grows at all.

As delivered the system comes empty. The first volume you may want to create is a root volume. This could be accomplished by issuing the command

createvol root vice1 /vicepa

To remove a volume, use the *purgevol* command. Its syntax is

purgevol *volname*

volname is the name you wish the volume to be referred to as, probably the user name of function of the volume.

This command is only necessary if you are through with a volume and no longer want to use it.

To move a volume from one server to another or from one partition to another use the command:

movevol *volname server partition*

volname is the name of the volume you wish to move

server is the name of the server that is suppose to have the volume after the move

partition is the partition on server that is to hold the volume

after the move

The *vol-lookup* command is only used for debugging, but it is nice to know if the volume was actually created.

Its syntax is:

vol-lookup *volname*

The last three commands described are not normally used by the operators, but they are used automatically by the system.

The *vol-salvage* command is used to ensure that the files in the system are consistent. It is the equivalent of the UNIX fsck command. Its syntax is

vol-salvage [-f]

the -f flag is used to force all volumes to be salvaged, normally only those volumes that might need salvaging are salvaged. Note that the fsck command is still used at reboot. It is also important to note that the fsck command has been modified to recognize *vice* files stored in a partition. **Use of an unmodified fsck will destroy all *vice* files on a system.**

The *bldvldb.sh* command is used to generate a new Volume Location Data Base. It is run automatically when volumes are created.

The *makebackups* command is invoked nightly (around midnight) to make read-only backup volumes of each volume in the system. These are created using the volume *cloning* mechanism, which does not actually create new copies of all of the files, but only copies of the volume structure that points to the files.

At C-MU we do backup using a separate staging machine, and then dump the volumes from there to tape. We are not distributing that code right now, but, the file system itself contains some primitives that would allow a backup strategy to be developed. There are two commands to convert volumes to a byte stream image and then back to a volume. There is a third command that will take a full dump a partial dumps and combine them into one dumpfile. There is also a file that contains a list of all of the volumes in the system and their type. By scanning the list of volumes and converting all of the read/write volumes to byte streams it is possible to build a system that will allow the volumes to be dumped on a regular basis. By basing the dump scheme on the information in the file it will not be necessary to change your procedures as new volumes are created. The commands are:

vol-dump *volnumber* [-*i* lower-time-bound] > *dumpfile*

where:
volnumber is the number of the volume to dump
-i is the time in Unix epoch time to start the dump from. If not specified or if zero is specified, the entire volume is dumped. This allows partial dumps to be made of only that data that was changed since the time specified. This time can be earlier than the end time of the full dump. For instance if you took a full dump at midnight on Sunday and on Tuesday took a partial dump of all files that had changed since 11:00PM on Sunday until Tuesday midnight, the *vol-merge* command below would emit a *dumpfile* that contained all of the files in the volume, as of midnight on Tuesday.
dumpfile is where the dump is to be placed. This shows redirection to a file, but it could be to a pipe or anything else, the file is written on standard out by the program

vol-restore [*-n*] *rw|ro partition* < *dumpfile*

where:
-n means leave the name the same as in dumpfile otherwise the name will be appended with **.restored.**
rw|ro means to restore the volume as a read/write volume, or as a read/only volume
partition is the disk partition to restore the volume to
dumpfile is the result of a vol-dump command

vol-merge *full-dump partial1 partialn ...* > *dumpfile*

where:
full-dump is the name of a full dump file
partialn is the name of one (or more partial dumps of the same volume
dumpfile is a new dump file that contains the merged results of the full and partial dumps

The file with the list of volumes is */vice/vol/AllVolumes* that exists on the control server. Its format is:

name number server partition size minquota maxquota type cdate mdate fetches

where:
name is the name of the volume
number is the volume number for the volume
server is the server that has the volume
partition is the partition on the server
size is the size of the volume in 1K blocks
minquota is the minimum quota value (currently unused)

maxquota is the maximum number of blocks allowed on the volume
type is R for read/only W for read/write or B for backup
cdate is the creation date in Unix epoch format
mdate is the last move date in Unix epoch format
fetches is the number of times the file has been fetched from the server

The fields are blank separated and there are newline characters between records. By writing a small awk script it is possible to generate a series of commands to dump all RW volumes. After that the restore consists of reloading the volume from the dumped copy. When a volume is restored, you can either restore it with the same name, or with a different name. If you choose a different name, you can just mount it and retrieve the files from it, if you restore it with the same name, you should ensure that there is not already a volume with that name in the system.

A scheme that would give you a daily backup is to take a full dump of all volumes on Saturday. That would give you all of the files as of Friday midnight. On Monday through Friday take a partial dump of all volumes as of 11:00PM on the previous Saturday. By merging the full dump with any of the partial dumps, you would be able to restore the files as of any day of the week. For instance if you wanted the files as of Tuesday midnight, you would merge the full dump with the dump taken on Wednesday and that would create a dumpfile of the volume as of Tuesday midnight. By then restoring that dumpfile, you would have a volume that contains all of the files at the date required.

Multiple Servers

At a large location such as C-MU we have devoted a separate machine to coordinate all of our servers. In this release the distributed server (vice1) has the coordination responsibility. This means that it will be running extra programs that handle cross server responsibilities, such as updating server binaries, distributing the volume location data base, etc. There are several different aspects to this:

Update server

First is a pair of programs called updatesrv and updateclnt. Updatesrv runs on vice1 (here after called the control server, since it also has the control functions running) and updateclnt runs on all other servers. This pair of programs are responsible for keeping files in a list of directories in sync. The main copy of the files are kept at the control server and the other servers check periodically to see if they have the latest copy. This means that by just updating files on the control machine, the other servers will automatically get the latest copy. That is how all of the binaries and data bases are distributed. The update programs are also used to keep the

clock on the other servers in sync. If you change the time here, it will cause the clock to be set on all of the other servers (and from there it will cause the clock to be set on all of the workstations).

Status server

The filestats program polls all of the on line servers and keeps a status block on them that the vopcon program can retrieve to allow status to be displayed for each of the servers.

Authentication server

The authentication server runs on all file servers, but it is only at the control server that passwords can be changed. This is transparent to the user since the Unix *passwd* command is still used to change them.

Other

Since the volume location data base is distributed from the control machine, all scripts to creat volumes, add users, etc. should be run on that machine.

Adding a new server

When a new server is to be added, it can be created using the same procedure as was used to create the first server. It must have a different name (vice2 has a nice ring to it). To add it to the system, simply bring it up (with its new name), add the name to vice1's host table, and then issue the *addserver.sh* command on vice1. The syntax is:

addserver.sh *hostname servernumber*

where:
hostname is the new server's hostname
servernumber is a number that is not already allocated in /vice/db/servers

The command will update the appropriate files on the server and make it known to the world. To get it to be recognized and displayed by the file server monitor, *vopcon*, it will be necessary to restart the filestats process. This can be done by just killing *vopcon*, as the monitor process will automatically restart it.

Authentication

Authentication with *vice* is different than in standard Unix. *Vice* has a central password server that is used during login to check your password. Each file server has a read only copy of the authentication data so any

server can answer login requests. Only the control machine has the read/write copy, so you can only change your password there. The local */etc/passwd* file no longer has passwords in it, it only contains the other information that Unix needs. You receive a new copy of *login*, *su* and *passwd*, which should be installed on you workstations. These programs know how to deal with the authentication server. Authentication only lasts for 25 hours. You must reauthenticate after that, either by logging out and back in again or by using the *log* command. To use *log*, just type the command and it will prompt you for your password. If you are leaving your workstation the *unlog* program will remove your authentication information from the machine.

It is also necessary to transfer authentication information between machines for commands like *rsh* and *rcp*. By running the */etc/netd* server and using the version of */etc/netd.conf* supplied, these changes will be transparent to you. They come on the workstation tape in the directory *etc*. They use the programs in *lib/netd* on the workstation tape.

What is not in this distribution

This section describes major functions that are running at C-MU but that we have not distributed, and why. The purpose of this distribution is to allow a few places to began to look at *vice* and see how it runs. It is not yet a full scale distribution. We expect to learn much about what an actual distribution will take from this.

Backup and Restore

The backup and restore that we do at C-MU is automated. It is not distributed because is that it is designed for a very large configuration and requires a separate machine. We plan to make this easily distributable. The basic backup and restore functions are distributed however. The automatic make backup command that runs from crontab. What this command does is to make a copy of all of the files in a volume. It does not actually copy any data however, it just builds another structure to point at the same data and then increments the reference count on the data. It operates much like a hard link does in the UNIX system. Having done this it then makes that data available in a subdirectory of the root volume called OldFiles. What this gives is a read-only copy of the files. By running this job in the middle of the night, you wind up with a copy of yesterdays files. It is then run the next day, and it deletes the previous days read-only copies and builds a new set. It is these read-only copies that are used for backup. This has a couple of nice effects. First it always appears that the dumps all happen at about the same time every day, and second, the dumps do not have to worry about the data changing, since the dumps are taken from read-only copies of the data.

We do distribute the commands to dump and restore volumes. Please see the section on dump and restore under **Volume Commands** to see how to tailor these to your needs.

What is going on

This section describes what the processes on your file servers do. It also describes what files are used by these processes and what they contain.

A ps aux of a running file server should look like this

```
USER       PID %CPU %MEM   SZ  RSS TT STAT  TIME COMMAND
root       136 18.7  4.8  192  128 p0 R     0:00 ps aux
root       131 10.4  3.5  180   90 p0 S     0:01 -csh (csh)
root        45  0.1  2.2   72   54 ?  S     0:00 /etc/netd
root        69  0.0  3.5  180   90 co I     0:01 -csh (csh)
root        70  0.0 31.8 1232  888 ?  S <   0:09 file
root         1  0.0  0.9   38   18 ?  I     0:00 /etc/ini
root         0  0.0  0.2    0    0 ?  D     0:00 swapper
root         2  0.0  0.3  100    0 ?  D     0:00 pagedaemon
root        34  0.0  1.4   56   32 ?  I     0:01 /etc/cron
root        66  0.0  6.5  250  176 ?  I     0:00 auth2
root        68  0.0  9.8  348  268 ?  I     0:01 filestats
root        64  0.0  1.2   48   26 ?  I     0:00 filemonitor
root        57  0.0  1.3   50   30 ?  I     0:00 authmon
root        61  0.0  1.1   46   24 ?  I     0:00 statmon
root        31  0.0  0.5   26    6 ?  S     0:00 /etc/update
root        58  0.0  5.9  246  160 ?  I     0:01 updatesrv -p /
```

The following processes are the ones that are specific to the file servers:

> The process called *file* is the actual file server code. The process called *filemonitor* is used to monitor the *file* process. If the *file* process exits for any reason, the *filemonitor* process will restart it. If the *file* process exited abnormally, the *filemonitor* also salvages the file system, before restarting the *file* process. The presence of a file called *SHUTDOWN* in the directory /vice/file is used to indicate that the *file* process exited normally.
>
> The process called *auth2* is the authentication server and it has the *authmon* process to restart it if it should quit. It will have been invoked with the -chk option on all machines except the control machine.
>
> The *updatesrv* process is used to keep selected files in sync on

the servers. It can also be used to allow work stations to fetch new *venus* programs.

The *filestats* and *statmon* programs only run on the control server and they are used to feed information to the vopcon program to keep a display of the server status.

The *vol-makebackups* process is invoked from crontab around midnight to create read-only backup volumes of all the read-write volumes, so it will not usually appear the *ps* listing.

What is kept where

This section offers a tour of the /vice directory and describes what the use of the files are. In the /vice directory are the following subdirectories

auth2 Used by authentication

Contains the log file and two small files that contain the process ids of the authentication server and its monitor

bin where *vice* binaries are kept

This contains the various binaries and shell scripts that are used by the file server.

db where common data needed for the servers are kept

These are the various data files needed by a running file server. Included in these are all the files needed for a complete distributed server with authentication and backup/restore.

VLDB this is the ***V***olume ***L***ocation ***D***ata ***B***ase

auth2.pw and ***auth2.pwa*** are files that contain the users passwords. They are used for the authentication servers to verify a user during login. The ***auth2.pwa*** file is a processed version of the ***auth2.pw*** file. If an ***auth2.pwa*** file exists, the server uses it, otherwise the server uses the ***auth2.pw*** a file and creates a new ***auth2.pwa*** file.

auth2.tk this is a token that is common to the file servers and the authentication servers.

files this is a small file that is used with multiple servers to replicate a set of files between servers.

serverkey this is also used in authentication to establish the file servers credentials

servers this is a list of the active servers and what range of volume numbers they are assigned. As servers are added they should be put in this list

vice.pcf this is the index data file for the protection data base.

vice.pdb this is the protection data base. We are going to merge these two files very soon.

file where file server data and control shell scripts are kept

This is where a number of log files are stored and where some files containg current process ids are kept. It also contains a number of shell scripts used to control the file server when local debugging is being done.

BackupLog a log file for the backup process that runs from crontab

FileLog a log file of the file server process

SalvageLog a log file created by the process of salvaging the file system

UpdateLog a log file that is used when there are replicated servers to indicate what files have been updated on this server.

UpdateMonitor the pid of the monitor process for the update server.

UpdatePid the pid of the update server.

pid the pid of the file server

monitor the pid of the file server monitor process

debugon tells the file server to start dumping

debug messages into FileLog. These give more information about the running server and should only be used for debugging. Repeated calls to the script increase the volume of messages.

debugoff tells the file server to reset its debugging messages.

list gives a list of the connected users in FileLog

restart does a clean restart of the file server when a salvage is not required

shutdown brings down the file server to a clean shutdown

startup starts the file server up

stats dumps some running information into FileLog about the various requests that have been handled by the server and the resources that it is using

Two files of interest in this directory are:

SHUTDOWN which indicates that the file server was shutdown cleanly and does not need to be salvaged.

FULLSALVAGE which indicates that a full salvage of all volumes should be done, not just a salvage of recently changed volumes, which is the default.

spool used by the volume package

This directory is used by the volume utilities.

vol used by the volume package

This directory is used by the volume package in the running server to keep track of information about the system.

AllVolumes, BackupList, RWList and ***VolumeList*** are lists that are used to keep track

of this servers volumes.

fs.lock is a file that is used to establish locking between the file server process and various other volume utilities

maxvolid is the maximum volume id that this server has created

partitions is a list of partition information

Other files of interest in the file system.

/ROOTVOLUME the volume name of the volume that is the root of the *vice* file system.

How to see what is happening

There are a few things that can be done to monitor the file servers. First is just to run a vopcon on an andrew workstation and see what is happening on the display. This allows you to monitor many of the operational parameters of the server and see if it is operating correctly and not overloaded.

However, there are some hooks to see what the file server is doing at any time. If you log on to the server and cd */vice/file* there are some debugging aids. You probably should not have to use these, but it may be necessary if you report a difficult problem.

First if you are checking the file server you should do a tail on *FileLog* which is a log where the file server records its status periodically. The command *tail -f FileLog &* will cause the contents of the last few lines of FileLog to be printed on the screen, and then any new lines will also be printed as they are stored. There are a few scripts in this directory that will help you to see what is going on. First a *stats* will cause a number of counters that the file server monitors to be dumped into the log. The dump is formatted and fairly self explanatory. The file server will automatically do this dump after every 4096 requests.

If you type *debugon* it will cause the file server to start putting out a message to the log for every request that comes in over the network. If you type *debugon* a second time, the amount of information will increase to include some internal information and also a line at the completion of every request. This is useful to see if the workstations are actually getting requests through to the server and what type of requests are coming in. To go back to the standard level of messages the *debugoff* command will reset the debugging level to zero. Always ensure that you type *debugoff*

before you leave or the log will grow rapidly and fill your partition with debugging information.

The other command that is useful is *list*. *List* will cause a list of which users are on which workstations to print out. This is useful to see if connections are being made. It will also indicate what level of authentication is in effect on the connection and when the token will expire. Connections that are not authenticated do not have an expiration time.

What we are doing

This section describes the changes we are making that will be coming later.

Smaller RPC

We are currently redoing the RPC code to make it much smaller. The reason for this is that one of the major loads on a workstation is the size of the *venus* process that is running there. A large portion of the size of *venus* is due to the RPC code. We are currently testing a version that is much smaller.

Programmable group control

Currently you are either in a group or not. It is sometimes nice to be able to enable and disable your participation in a group. We are testing a version of the access list package that allows you to enable and disable your group participation based on calls through *venus*. This will allow, for example, people who are system administrators to turn off their administrator rights, except when they need them. It would also allow access lists to be built that can be turned on and off under program control.

Trouble reporting

If there is trouble with the file server it will automatically restart itself. It should also leave a *core* file in the directory */vice/file*. This *core* file should be transferred here along with a copy of the *FileLog* file that is also in the same directory.

If a *venus* has trouble it leaves a *core* file in its cache directory. It is that core file that we need. This core file will be deleted when the *venus* restarts, so to save it you must boot your system single user and move it someplace else before reboot.

Trouble reports should be sent to me - **Mike West** Phone - **(412) 268-6737**, IBM-TieLine **363-6737**, vnet address **cmumjw** at **pghvm1**, arpa net address **mikew#@andrew.cmu.edu**, until we get a formal trouble reporting structure in place. We would also like to have any comments you have on the file system, either good or bad; and any requests for functional enhancements.

File and Directory Protection

Protecting directories and files in Andrew

Andrew's protection mechanisms allow you to control **who** can do things with the files in a given directory or subdirectory that you own, and **what** actions each person can take.

Who can work with your directories

You can allow individual users or groups of users to access your directories.

A **user** is someone with an Andrew account. Individual users are referenced by their login names.

A **group** is a collection of users. Small groups may contain only individual users. Larger groups may have smaller groups as members as well as individual users. For example, you might establish a group consisting of co-workers and give them access to a particular directory. Then, you decide that you want all those people as well as a few others to have access to another directory. You could make the smaller group a member of the larger group and everyone in the smaller group would automatically receive the permissions granted to the larger group. At present, you cannot define your own groups except by working with a System Administrator; however, a facility for defining groups will be added soon.

In general, the only groups the protection system recognizes now are the system groups such as System:AnyUser, which is the large group composed of everyone with an Andrew account.

The **access list** for a directory contains the names of the users and groups who can work with the files in that directory. You can edit the access list for a directory that you own, add users to it, and change what they can do, by using the *fs* program.

What other users can do with your directories

The **permissions** that you can to give to users or groups control what they can do with the files in a given directory. Permissions are sometimes referred to as "rights." Here are the possible permissions in Andrew's protection system and what they allow users to do:

Lookup (L): obtain status information about the files in the directory.

Read (R): read any file in the directory.

Write (W): write any existing file in the directory.

Insert (I): add new files or subdirectories to the directory.

Delete (D): remove files or subdirectories.

Lock (K): place read locks on any file in the directory. Used mainly by application programs.

Administer (A): modify the access list and ownership of a directory. The owner of the directory ALWAYS has Administer rights, even if he or she is locked out of the directory, and can therefore reset the protections.

Defaults and Examples

Default protections for Andrew directories. Each user automatically receives the protection that the parent directory had. As part of user installation you may wish to define a common protection that is set on user directories.

Example: Assigning permissions to groups and subgroups. You can assign permissions to groups of users as well as to individuals. If you assign permissions to a group, any subgroups of that group which you then create will receive the same permissions. In addition, you may assign the subgroups specific rights of their own.

To illustrate this, let us suppose that user langston has established two groups, one called langston:students and one called langston:assistants. The students need read, lookup, and insert rights to a directory called "notes" so they can examine the files in the directory and also add new files. The assistants need the same rights and also write rights, so they can amend and correct the files when necessary. If user langston makes the assistants part of the students group, she can give them all read, lookup, and insert rights, like this

langston.students rl

Then, when langston assigns langston:assistants write rights, the assistants will automatically receive the other three rights, like this

langston.assistants rliw

Example: Assigning negative rights to specific users. To give a user "negative" rights (that is, to deny that user a particular set of rights to a directory), you must add "-negative" before the user's name in the command line for *fs*. For instance, suppose user langston decides that all Andrew users except dgg should have read and

lookup access to a directory called "notes." This could be accomplished with

```
fs sa /cmu/itc/langston/notes System:AnyUser rl -negative
dgg rl
```

You can also use the "-minus" switch to accomplish the same thing. The counterparts to -negative and -minus (that is, -positive and -plus) also work; for instance, the following command would accomplish the same thing as the example above:

```
fs sa /cmu/itc/langston/notes -minus dgg rl -plus System:AnyUser
```

Fs considers all the rights you add to be positive unless it sees a -negative or -minus; then it considers all the rights that follow to be negative unless it sees a -positive or -plus. Now, if langston lists the access on the notes directory using "fs la /cmu/itc/langston/notes", the following information will appear:

Normal rights:
System:AnyUser rli
langston rlidwka
Negative rights:
dgg rli

Removing negative rights, or restoring rights you have denied: If you have denied a user access to a directory by using "-negative", you must also restore the rights using -negative. That is, to restore dgg's rights in the examples given above, it would not be sufficient for langston to give the command "fs sa /cmu/itc/langston/notes dgg none". Instead, langston would have to issue this command:

```
fs sa /cmu/itc/langston/notes -negative dgg none
```

which says that user dgg should no longer have any negative rights. That will allow dgg to inherit the rights of System:AnyUser and thus have the read and lookup rights that were denied in the previous example.

Controlling access to individual files

We encourage users to control access to directories rather than individual files because of the greater flexibility offered by the access list mechanism. However, there may be a particular case in which you have one sensitive file in a directory that you wish to protect separately. For such cases, the *chmod* command may be helpful. In the Andrew protection system, *chmod* allows you to control whether the file can be read or written at all. If you "turn off" reading for the file, you will prevent anyone, including yourself,

from reading it; the same is true for writing. By default, files can be read by anyone with Read access to the directory in which they appear and written by anyone with Write access to the directory. Here are the commands that you may issue to control reading and writing to a given file:

To turn off **writing** for the file:

chmod -w <filename>

To turn off **reading** for the file:

chmod -r <filename>

To then make the file **writeable** by those with Write access to the directory:

chmod +w <filename>

To then make the file **readable** by those with Read access to the directory:

chmod +r <filename>

fs: A File System Program

What fs is

fs displays information about the file server. Most often, people use it to see how much of their allocated storage space they are currently using and to change the protection on their personal directories. **fs** will only change the protection or "rights" on directories and not on individual files. To change protection on individual files, you need the Andrew version of *chmod.*

Quick reference

Syntax:

To list or change the volume or space allocation for a directory:

fs lv directoryname
fs sv directoryname
[-i minquota] [-a maxquota] [-n name] [-m motd] [-o offmsg]

To create, delete, or examine a mount point:

fs mkmount volumename filename [-rw]
fs rmmount filename
fs lsmount filename

To force *venus* to re-evaluate the meaning of all mount points to backup volumes:

fs checkbackups <any vice file name>

To force *venus* to immediately check the status of all file servers to see if they have crashed or restarted:

fs checkservers <any vice file name>

To flush a file from the *venus* cache:

fs flush filename

To find the location of a file:

fs whereis directoryname

To list or change protections:

fs la directoryname
fs sa directoryname [-negative] username < [

rwidlak] [all] [none] [read] [write] >

Options/arguments:

lv: for "list volume." Shows the current status of the volume on which the directory is stored.

sv: for "set volume." Sets the current status of the volume on which the directory is stored. This option is only available to members of the System:Administrators group.

Volume attributes include the following:

minimum quota -- currently not used.

maximum quota -- the maximum amount of space this volume is permitted to store. If this amount is 0, then no quota is enforced.

name -- the name of the volume. This name is the same name as placed in mount points, so renaming a volume may cause it to essentially disappear from its old mounted position.

motd -- the message of the day. A string an operator may attach to a volume, describing some news relevant to the volume

offmsg -- the offline message for the volume. If a volume is offline, this message should explain why.

flush: Remove the file from the *venus* cache. This command should not be required.

mkmount: create a mount point. Mount *volumename* at the point in the file system described by *filename*. If the *-rw* flag is set, never use the corresponding read-only volume for a volume. Otherwise, if a read-only volume exists (and the parent is also on a read-only volume) then use the read-only volume.

rmmount: delete a mount point. Remove a mount point from the file system. The volume itself is not changed.

lsmount: list the contents of a mount point. This command can be used to tell what volume a mount point refers to.

checkbackups: This command re-evaluates all mount points for backup volumes immediately. Thus if a new backup volume has been created for a particular volume, the mount point to that volume will be evaluated to this very latest volume. *Venus* periodically checks the status of backup volumes in any event. The file name need only refer to a file served by *venus*; it is used by the kernel to route the command to the appropriate *venus* if more than one is running on a workstation.

checkservers: This command is used to force *venus* to check if any server it previously thought was down has come up again. *Venus* checks these suspected down servers every 4 minutes anyway; this command is only required by the very impatient. As with checkbackups, the file name need only refer to a file served by *venus*; it is used by the kernel to route the command to the appropriate *venus* if more than one is running on a workstation.

whereis: Lists the file server or file servers having copies of this file.

la: for "list access." Shows the permissions on the directory specified.

sa: for "set access." Sets the access to the directory specified for the user named. This option is only available to System:Administrators, owners or users with administer rights on this directory.

directoryname: the name of the directory for which you are checking or changing the access. You can supply the "twiddle" (~) as a directory name to check your home directory.

-negative: makes the rights you specify for *username* negative, or denies the named user the rights you specify. See the **Examples** below.

username: the name of the user for whom you are checking or changing the access.

rwidlak: one or more of the following letters which represent different permissions:

r (read): Allows the user to read any file in the directory.

w (write) : Allows the user to edit any existing file in the directory.

i (insert): Allows the user to create new files or subdirectories in the directory.

d (delete): Allows the user to remove files or subdirectories in the directory.

l (lookup): Allows the user to obtain status information about the files in the directory, for example, to list the names of the files in the directory.

a (administer): Allows the user to change the access list on that directory. You automatically have administer rights to any directories you own.

k (lock): Allows read locks to be placed on any file in the directory. Lock is used mainly for application programs. If you are not writing any application programs, you can ignore Lock.

all: Allows the user complete access to the directory.

all: allows the specified user all the above permissions.

none: allows the specified user none of the above permissions.

read: allows the specified user the rights normally associated with the ability to read files (that is, **rl**).

write: allows the specified user the rights normally associated with the ability to write files (that is, **rwild**).

Notes:

All your directories and any files in them have certain protections assigned to them which either allow or do not allow users access to your directories and files. Typically, all users have look up rights on your Mailbox directory. This allows them to send you mail and to see how many pieces of mail you have in your Mailbox. (It does not allow them to see who sent the mail or to read your mail).

Any subdirectory you create will automatically be assigned the same protection as the directory immediately above it (its

parent directory). Any file you create will automatically take on the protection of the directory in which it appears. If you are careful about where you store your files, you will not need to change protections as often. When you **do** need to change protections, you can do so for any directory that you own.

Examples

Substitute the names of your own directories and the appropriate users for the ones used in the examples below.

To list the access on a directory you own:

fs la /cmu/itc/langston/texts

You would see a listing like this:

Normal rights:
System:AnyUser rl
langston rlidwa

This listing tells you that any user on the Andrew system can read (r) and look up (l) files in the directory called "texts". Langston, the owner of the directory, has the following rights: read (r), look up (l), insert (i), delete (d), write (w), and administer (a).

To give another user access to a directory you own:

fs sa /cmu/itc/langston/texts dgg rwild

This line means that user langston has allowed user dgg read, write, insert, lookup, and delete access to the texts directory. Now the command "fs la /cmu/itc/langston/texts" would show this information:

Normal rights:
System:AnyUser rl
langston rlidwa
dgg rwild

To deny another user access to directory you own:

To give a user "negative" rights (that is, to deny that user a particular set of rights to a directory), you must add "-negative" before the user's name in the command line. For instance, suppose user langston decides that all Andrew users except dgg should have read and lookup access to a directory called "notes." This could be accomplished with

fs sa /cmu/itc/langston/notes System:AnyUser rl -negative dgg rl

You can also use the "-minus" switch to accomplish the same thing. The counterparts to -negative and -minus (that is, -positive and -plus) also work; for instance, the following command would accomplish the same thing as the example above:

fs sa /cmu/itc/langston/notes -minus dgg rl -plus System:AnyUser

Fs considers all the rights you add to be positive unless it sees a -negative or -minus; then it considers all the rights that follow to be negative unless it sees a -positive or -plus. Now, if langston lists the access on the notes directory using "fs la /cmu/itc/langston/notes", the following information will appear:

Normal rights:
System:AnyUser rli
langston rlidwka
Negative rights:
dgg rli

Removing negative rights, or restoring rights you have denied:

If you have denied a user access to a directory by using "-negative", you must also restore the rights using -negative. That is, to restore dgg's rights in the examples given above, it would not be sufficient for langston to give the command "fs sa /cmu/itc/langston/notes dgg none". Instead, langston would have to issue this command:

fs sa /cmu/itc/langston/notes -negative dgg none

which says that user dgg should no longer have any negative rights. That will allow dgg to inherit the rights of System:AnyUser and thus have the read and lookup rights that were denied in the previous example.

To remove all rights for all users:

fs sa /cmu/itc/langston/accounts System:AnyUser none

The above command would remove all rights from all users except Langston to the subdirectory called "accounts."

Up: A Backup Facility

What up is

Up is a program that will selectively copy files from one directory to another. That is, used more than once, it only copies files which have changed.

Quick reference

Syntax: **up** [-o] [-v] sourcedirectory targetdirectory

Options/arguments:

-o -- copies the files but does not copy the Vice access list information.

-v -- turns on verbose output.

Notes:

Unlike the *cp* command, *up* preserves the date on any file it copies. That feature is useful if you have programs which assume that two files with the same date have the same contents; you can move files around without changing their dates. In addition, some programs like the Unix *make* program compare dates of files to see whether one was made out of another. In some cases you can save unnecessary recompiling by using *up* if you want to copy a whole directory.

Up is often used by people who want to copy files to their local disk so they can work apart from *Vice*. More information on that, including some important warnings, is in the section of this text called "Maintaining files locally."

Up will copy a file from the source directory to the target directory if 1) the file is not already in the target directory; 2) the write date on the two versions differ; or 3) the lengths of the two versions differ.

Up is designed to preserve *Vice* access list information. When you up files from the local disk, none of them will have this information and you will see an error message. You can ignore it.

If a file in the target directory has write permission for the owner turned **off** (e.g. chmod 400 filename), *up* will not overwrite that file.

Vice Overview

The *Vice* file server is designed to meet the need of giving distributed work stations a shared file system. It gives a time-sharing view of files to a distributed set of workstations. From the workstation point of view, it looks like a large hierarchical file tree. Users and programs running on a workstation can deal with the files as if they were local. Programs written to run against the Unix file system, will work unchanged if the files are in *Vice*. If a user moves from one workstation to another, he still has full access to the same files, and there is no need for him to change any of his normal procedures.

It was necessary to make a few changes in the way files were handled, particularly in the area of file protection. *Vice* adds a protection mechanism that allows a more generalized mapping of users to their rights to deal with files. This addition was felt to be necessary to handle the more complex situations that would evolve when the shared file system grew to handling thousands of users. To support this an authentication server was implemented that takes the place of Unix password verification, by dealing with a central authentication server.

What follows is a scenario of what happens as a user logs on to a workstation that is a *vice* client. This should give a better idea of the way we have chosen to implement *vice*.

First, when the workstation is booted, a local process (called *venus*) is started. The responsibility of this process is to map local Unix requests into *vice* calls. This is done by retrieving files from *vice* when the workstation needs them, and placing them in a local cache. When *venus* starts up, it scans through its cache and finds out which files are currently stored there. Once in the cache, the user requests deal with the local copy of the file, and when the last user closes the file, it is stored back to *vice* if it was changed. The mount command is used to tell the kernel to associate a new device with a directory, and this is the kernel's indication that files accessed in these directories are associated with *vice*. The new devices are */s0-2* and any directory can be used. For the purposes of this description I will assume that the command */etc/mount /dev/fs0 /vice* was the one issued.

When a user comes up to the workstation and logs in, *login* uses the password supplied to go through a hand shake with an *authentication server*, and winds up with a set of tokens that can be used to talk to any *vice* file server. A couple of points of interest here, first the password itself is never transmitted on the network, and second the tokens returned have a time stamp that causes them to expire in 25 hours. This means that if a token is somehow stolen, it is only good for 25 hours. The two tokens returned are a *venus token*, which is used by *venus* to get information

about the user and a *server token*. The *server token* is encrypted with a key only known to the file server and the authentication server. There is enough information in the token for the file server to verify that the user is a valid user, and this saves the file server from having to contact the authentication server. Once *login* has these tokens it passes them to *venus* so that *venus* can use them to authenticate the user with various file servers. *Venus* saves the token in an area that is related to the user.

As login proceeds and the user opens a file in one of the *vice* directories (say his .cshrc file with the file name */vice/usr/foo/.cshrc*), the open request is received by the Unix kernel. The kernel recognizes that /vice is associated with the *fs0* device and passes the call to *venus*. *Venus* receives the request which contains the *vice* path name. It then goes through a process very similar to the kernel's *namei* routine. It looks at the root of the *vice* file system and finds the name *usr* in it. Associated with the name is a *vice* file system *id* or *fid*. (Calls between *venus* and *vice* identify files by a *fid* rather than by name. Name resolution is done entirely in *venus*.) It finds the name *usr* in the root directory and checks its cache for the *fid* associated with it. Assuming it's in the cache it would then look up *foo* in the usr directory, and find it. For the sake of simplification, let us assume that all of the directories are in the cache, but that the file is not. (This is not an unusual assumption, in fact if the user regularly uses this workstation, the *.cshrc* is probably in the cache as well.) Next, *venus* finds the *fid* for *.cshrc* in the *foo* directory, and finds that the *.cshrc* file is not in his cache. *Venus* sends a request to *vice* to fetch the *.cshrc* file into it's cache (freeing up some space if necessary). The request to *vice* specifies the *fid* of the *.cshrc* file.

When *vice* receives the request, the access rights of the user are checked, to ensure that the user is allowed to read it. *Vice* then initiates a transfer of the file to the workstation. *Vice* also remembers that the workstation has a copy of the file. This is used to notify the workstation if another workstation changes the file.

Once *venus* receives the response it passes information back to the kernel to indicate the open was successful and passes a Unix file descriptor that allows the user program to deal directly with the file in the cache. From here until close the program deals entirely with the local file. At close time *venus* notes that the cache copy of the file is not being used, and will remove the file if space is needed in the cache.

Since the server will notify the workstation if the file changes, any subsequent requests for the file can be satisfied immediately, there is not even the need for a request to *vice* to ask if the cached copy is current. This is why the directory search mechanism works, because most of the time the directories needed are already in the cache, and lookup is just a local operation. In fact, well over 95% of the time, all files are in the local

cache.

As a consequence of file caching, the semantics of file sharing between processes on different workstations are somewhat different than in a standard Unix system. (For processes on the same workstation there is no essential difference.) When the file is initially created, it is stored as a zero length file on the server (to allow processes on other workstations to try to obtain and advisory lock on the file , and to implement the exclusive open mode), but until the last close of the file is received from a given workstation, the file is not updated further at the server. This implies that as long as the file is kept open at a workstation, no other workstation will see any changes made by that workstation.

Vice Operation Overview

The *Vice* file system is a number of machines working together to provide a view to its clients of a single hierarchical file system. Most of the machines are *file servers*. There is also a server *control machine* where the servers get the latest copies of data bases and binaries. The control machine also runs a program that allows various *consoles* to display the latest status of the servers. The file servers are the machines that actually store the files. Files are stored in *volumes* which are just a connected part of the hierarchical file structure. The functions of the *control machine* may be combined with one of the *file servers* to create a *control server*.

Terms

I will use the following terms frequently so I will define them here.

File Server The machines that contain the data that makes up the file system hierarchy. There are many of these machines, each contains a number of different volumes.

Control Machine The central machine where all of the common routines and data to run the servers is maintained. This may be one of the servers.

Volumes A logical structure used to contain parts of the hierarchical structure of files. All files in a volume are logically connected. Usually used for related files, like the files of one user. The volume is used extensively. It is the unit of data that is moved from server to server, the place that quotas are assigned, the unit of data that can be made read only for replication and in general the unit that *operators* of the file system work with.

Console A machine that is used to run the server monitor program. The console will indicate if a server is down. It is also useful for determining how well a server is running when it is up.

The volume is the critical unit of the file system for operators. It is volumes where quotas are applied. It is volumes that are restored if a user loses a file. It is volumes that represent a users set of files. If a partition on a server is getting too full it is a volume that must be moved to free up space on the partition.

Machines

Since there are several types of machines that make up a file system I will describe each type and the function of the programs that run on them.

File Server

These are the machines that actually store the users files. The primary processes that they run are *file* and *filemon*. *File* is the actual file server that works with *venus* to present the users with their files. *Filemon* is a program that will automatically restart file if it is stopped or abends. *Update* and *UpdateMonitor* are used to keep the common files up to date. They connect to the control machine and ensure that all of the local copies of common files are correct. /vice/bin/updatesrv is a local server that is used by workstations to fetch current copies of *venus* before *venus* is started.

Control Machine

This machine is used to centrally administer the file servers. It has a process running on it that the file servers use to keep files on their local file systems up to date. Also each of the file servers sets its clock to match the one on this machine.

Other files in the directories are maintained automatically as follows:

> A central copy of the authentication data bases is kept here and propagated to all of the file servers using the update programs.
>
> It is here that the ***VLDB*** (***V***olume ***L***ocation ***D***ata ***B***ase) is also kept. It is built based on information gathered from each of the servers.
>
> The *vice.pdb* and *vice.pcf* files are built by the add user utilities and propagated to the other servers. For more information on the *vice.pdb* and the *vice.pcf* see the section on adding users.
>
> The *auth2.pw* and *auth2.pwa* files are built by auth2 and propagated to the other servers which only have read/only copies of them.

All other files in the control directories are updated manually.

The primary use of this machine is to make the administration of the file servers easier.

Console

This machine does not have to be a dedicated machine. Any machine that is running a copy of *vopcon* is a console machine. For now all you can do from one of these machines is monitor the file servers to see how they are running. A complete list of the options for *vopcon* and how they are used see the section on monitoring below.

Operation Description

From a user's point of view the unit transferred between *vice* and *venus* is the file. *Venus* fetches and stores files from *vice*. From an operations point of view the unit that is dealt with is the volume. All operations deals with volumes. When users are added to the system, it is a volume that is created for the user. When quotas are put on, they are put on a volume. When data is dumped and restored, it is once again the volume that is used as the unit of interchange. When a user is moved from one server to another, this is accomplished by moving his volume. The point of this is that operations should think of volumes when they think of administering *vice*. This turns out to be an essential simplification, particularly when you are dealing with a large system. In a very large system there might be as many as 20,000 volumes, but there could be in excess of 500,000 files and directories.

The primary tasks of administering a *vice* system is the creation of volumes when users are added to the system, the placement of those volumes on a particular server, and perhaps the moving of volumes between servers to handle space or load balancing considerations. In addition, the adding of users requires the updating of the central authentication data base where passwords are stored, and the local copies of the Unix */etc/passwd* file, which contains other information about the user. All of this is done in the bldpass.sh script that is provided. The design allows all of this to be accomplished from the control machine. The intent is that file servers run themselves, and the file system is operated from the control machine and monitored with a console. The file servers themselves should run automatically and only need attention for hardware problems. To accomplish this processes on file servers are automatically restarted if they should fail. All system change, new servers, new users and etc. is introduced to the system from the control machine.