CMU-ITC-84-020

VICE File System Services

MJ West

Information Technology Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

August 7, 1984

This is a preliminary document
The interfaces may change

Introduction

This paper deals with the Vice File System; what services it offers, what data structures it uses and how it is implemented. The intent is to keep this document up to date to reflect the state of the Vice file system.

The structure has been chosen to simplify implementation and present the correct semantics of the intended services. It is left for later to optimize the basic services and to work on performance. By striving first for simplicity of implementation and correct semantics it is possible to ensure that the semantics are complete. The initial base system is 4.2BSD UNIX.

The paper is organized to first describe an overview of the Vice file system, then to describe an overview of the implementation, then a description of the services and finally the data structures used to support those services.

It is intended that this be implemented on a cluster machine. Separate papers will describe:

the protocols used for communication.
the workstation implementation.
the access list package
the authorization server.

Overview

The VICE system consists of a number of cluster machines that are logically connected to each other, each of the cluster machines have many server processes running on them, one of these servers is the file server, and that is what this paper describes.

It is assumed that each of the cluster machines is on a lobe of a local area network that may contain some workstations. Workstations refer requests for files to the closest cluster machine and the file server on that cluster either provides the file, or refers the workstation to a cluster that has the file.

File Server

The file server is a set of processes running on a cluster machine. The processes are:

- a process that does initialization and waits for workstation connects.
  - a process for each workstation to handle its requests.
- a process to wait for connect requests from other clusters.
  - a process for each connected cluster file server.
- a process to handle backup requests (requests to other clusters).
- a process to handle lock requests.

Network appearance

Each file server will have two identities on the network. There will be one name for the file system that is used by workstations and another that is used by the file system on other clusters. If a network name is considered to be a cluster ID and server ID pair, then each cluster that contains a file system will have a file server name to handle workstation requests, and a cluster server name to handle the cluster to cluster requests from other file servers.

New VICE specific information

The VICE file system directory logically contains a single hierarchical tree. The file system is distributed across the various clusters that make up the VICE file system. The upper part of the directory tree and common system files are replicated at all VICE clusters. The rest of the directory and most files are maintained at a single cluster machine.

> Associated with each file and directory is some additional information. The new information is:
>
> the name of the custodian which keeps the primary copy of the directory and its files.
>
> the backup cluster machine(s) which keeps a backup copy of the directory and its files.
>
> descriptive information about the file or directory that is maintained by other services.
>
> a type field to indicate the type of the directory and any member files. The types are either *replicated*, or *normal*
>
> the date when the data was last changed, and the date when the status was last changed
>
> the owner of the directory
>
> an access list that is used to determine who can do what to the data

The new information is kept in separate files in the same directory structure. This is done by creating a new directory for each directory in the system. The new directory will be named "*.admin*". This new directory will contain a "*.admin*" file for the information about the directory, and a file for information about each file that has the same name as the file.

The cluster and type information, and the descriptive information about the directory are kept in the *.admin* file in the new directory. The descriptive information about the files are kept in a file in the .admin directory. The file name of the file in the *.admin* directory will be the same as the file name in the standard directory. The information that is kept in the inode will be left there.

The ability to set and fetch this information will be through the GetFileStat, GetMember and SetFileStat calls. It will not be allowed to open a directory and read it as a file as is currently possible in UNIX. All programs that currently do that will have to be changed to use the GetFileStat, GetMember and SetFileStat calls or the workstation code will have to be written to issue the calls and build a block that looks like the current directory.

Access Lists

Protection is provided throught the use of the access list package. Each directory will have associated with it an access list that will determine the ability of users and groups to modify it and the files it contains. The levels of access provided are:

Access	Meaning
Read	**Fetch files**
Write	**Store old files**
Insert	**Store new files or dirs**
Delete	**Remove old files or dirs**
Lookup	**Examine dir contents**
Lock	**Obtain Read locks**
Protect	**Change access lists**

For a more complete description of the package see the document User's, Groups, and Access Lists.

File Types

There are two types of files:

Normal

These files reside at the custodian cluster, in addition there may be a list of sites that have Backup copies of the file.

Replicated

Files and directories which are located on all clusters. All clusters, except the custodian, treat these files differently. They can respond to fetch requests for the file without going to the custodian to check that it has the latest copy. Whenever a change is made to a replicated file, the custodian sends the same change to all cluster file servers. It does this by simply passing the same request it receives on to all clusters.

The type of a file is the same as its parent directory. Any directories will have their own type information.

Dates

The system maintains three dates:

data last modified date (STAT_MDATE)

This date is kept by the system for files and directories. It cannot be changed by the user. For files it is the time of the last store. For a directory it is the latest of:
when the directory was created.
when a new member was added (file or directory)
when a member was deleted (file or directory)

status last modified date (STAT_SDATE)

This date is kept by the system for files and directories. It cannot be changed by the user. It is updated whenever the data last modified date is. In addition it is updated if any status information about a file or directory is changed. In addition to the above calls, the following also modify the status date:
when a SetFileStat is done.
when a SetAccessList is done.

user last modified date (STAT_UDATE)

This date is only kept on files. It can be changed by the user either with a SetFileStat or passed in as data on a Store. If the user does not modify it, it will be the same as the system last modified date. This allows the user to have a modify date that reflects his knowledge of the data.

The name in parentheses behind the dates is the define in filedef.h that describes these dates.

Custodian

The *custodian* of a file or directory is the cluster where that file is synchronized and controlled. It has a copy of the file. The name of the custodian cluster and any backup clusters are kept in the .admin file. *The custodian of a file is the same as its parent directory.* All files have a custodian, even replicated ones. The major difference is that for replicated files, the copies are updated by the custodian, and for normal files the process of Fetch ensures that the latest copy is retrieved.

Backup

A function not described in the base document is backup clusters. It is possible for a file to be assigned one or more backup clusters. The backup clusters are used to fetching of a file even if the custodian is not available or the media that holds the file is physically damaged. When a backup cluster receives a fetch request for a file, it checks with the custodian to ensure it has the latest copy, if it does, it satisfies the request, if it doesn't it retrieves a copy of the file and satisfies the request. If the custodian is partitioned, it returns the copy of the file it has along with an indicator saying that the file may be out of date.

Locks

Locks are handled by a separate process. All lock requests are sent to the new process and it keeps a list of the outstanding locks, and either grants the request or rejects it. It is also able to free all of the locks for a given workstation or user. There is also a function to test if a given workstation holds a read or write lock on a given pathname. If an operation requires a lock and it is not held, the lock will be requested. If the lock is available the operation will be performed and the lock will be released. If the lock is not available, the operation will be aborted using the return from lock.

Operation Overview

Once initialized, the system operates as follows:

> A process is waiting for workstation connect requests. Whenever a workstation connects to the cluster it forks a process to handle that workstations requests. The process stays active handling those requests until the workstation issues a disconnect.
>
> A process is waiting for requests from other clusters. Whenever a connect request is received from another cluster it forks a process to handle the requests. The process stays around until the other cluster disconnects. The main difference between this and workstation requests is that clusters will usually disconnect after each request.
>
> A process is waiting for requests to update replicated or backup files or directories. Whenever a replicated or backup file is updated, a message is stored in the queue of this process and it takes care of updating all effected clusters.
>
> A process is waiting to handle lock requests. It keeps a table of the outstanding locks in storage. It is able to lock on a pathname and unlock by pathname, workstation, or user and workstation. It can also check whether a lock is held.

Examples

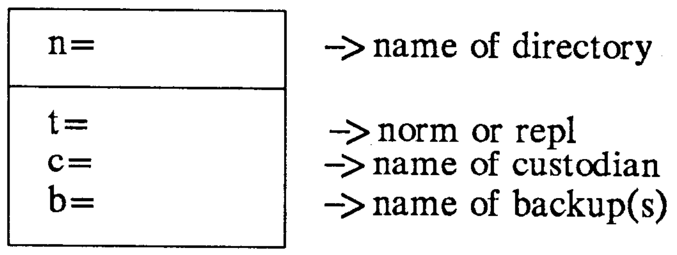
A client that wanted to Fetch a file would need the following code:

Too be filled in with an example using RPGEN.

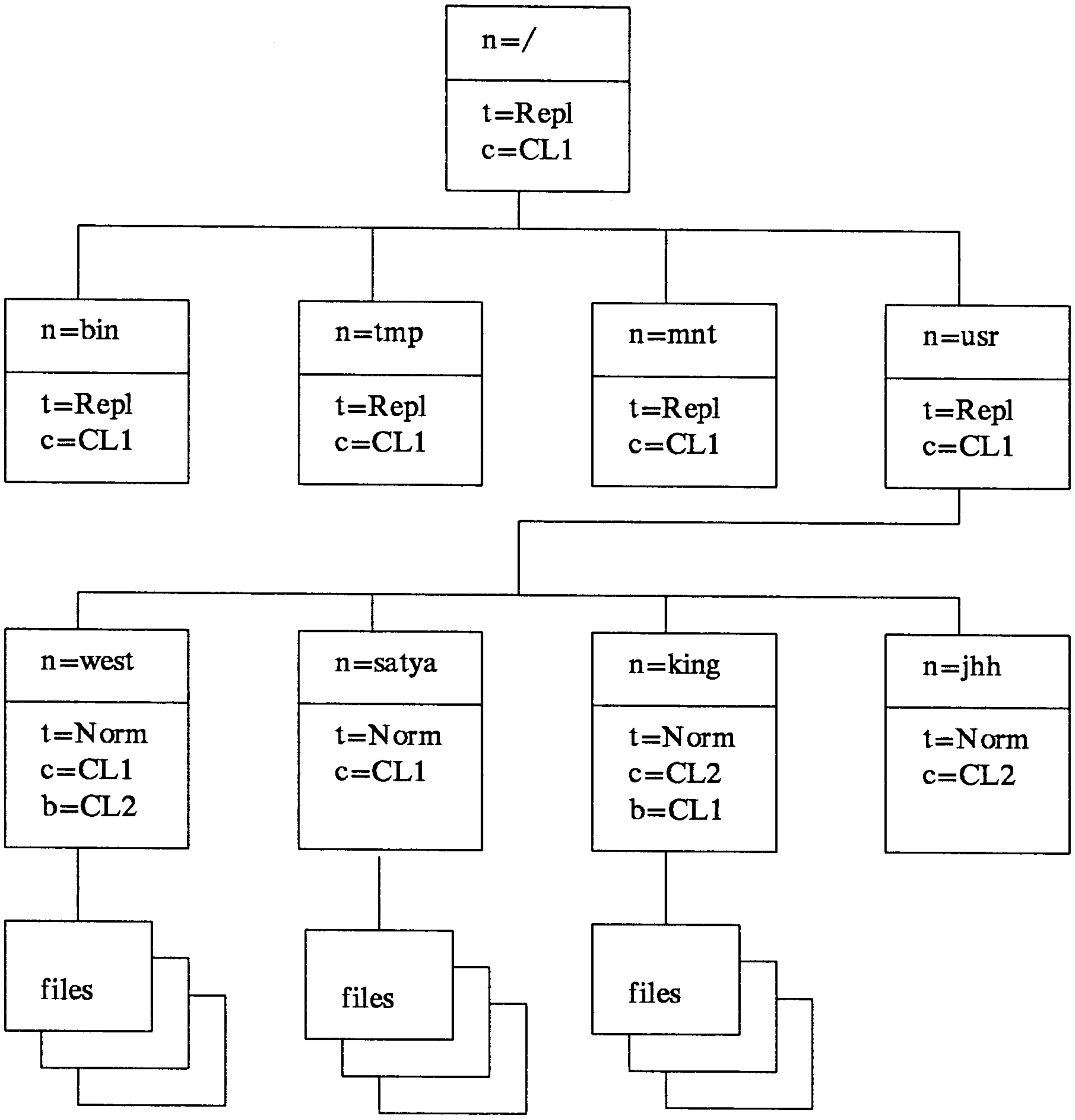
Structure

Directory

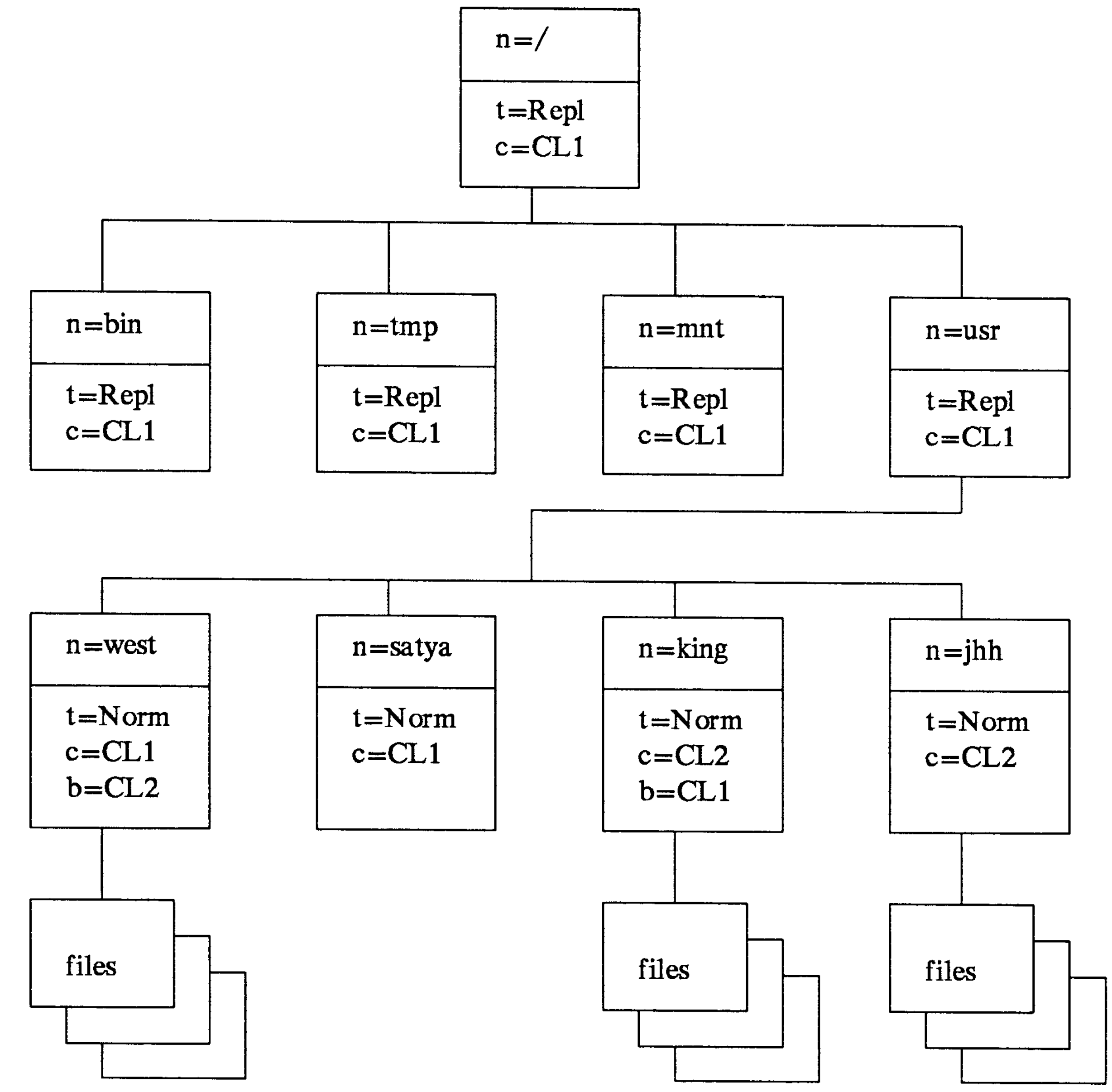
The directories structure is similar to that in UNIX. A new directory is added to each directory to allow some extended information to be saved. The new directory contains a file for the directory information and a file for every file (not sub-directory) in the directory. The new file is like a directory extension and contains additional information about the directory and any files that are contained in it. Current information is left intact. A pictorial representation of a directory looks like the following:



Directory Representation



Directory structure for CL1

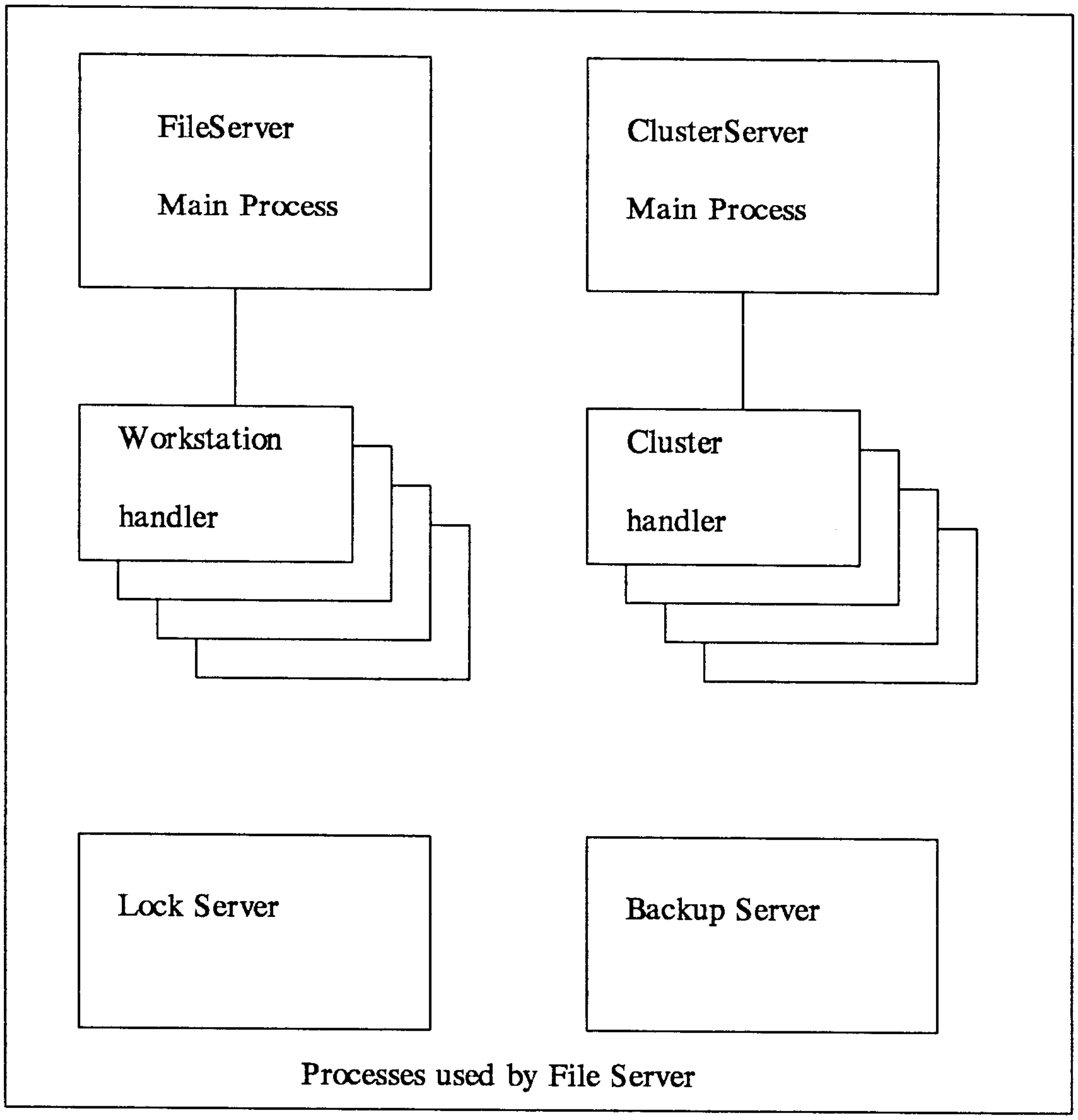


Directory structure for CL2

This shows that the directories that are replicated are at all clusters. Directories and files that are normal are only at the custodian and any backup clusters.

Processes

The structure has two processes that listen for connect requests from other machines, the main and cluster processes. Each of these processes have child processes that are used to handle the requests. In addition there are processes that are used for special purposes such as Lock, WhereIs and backup requests.



Processes used by File Server

The process structure in the Vice file system.

This is a brief discussions of the processes used on a cluster machine by Vice.

The FileServer Main Process listens for workstations to connect.

For each connected work station there is a process to handle its requests.

The ClusterServer Main Process listens for other servers to connect.

For each connected cluster there is a process to handle its requests.

The lock server handles lock requests for this cluster.

The BackupServer connects to other clusters to move replicated and backed up requests around the network.

Service routines

The service routines described below are reached by using the protocols described in the document titled "RPC User Manual Outline". The input for the routines is placed in the RPC_RequestBlock.

The RPC_RequestBlock must contain the following information to invoke the service routines:

RPC_RequestHdr

BodyLength field contains the length of the body.
SubSystem field in the request block header is set to FileSystem (FILESYSID).
OpCode field in the request block header is the name of the service routine.

Body is the parameters described in the following routines as input parameters

The return from the routines is passed in the RPC_ResponseBlock.

The RPC_ResponseBlock must contain the following information to invoke the service routines:

RPC_ResponseHdr

BodyLength field contains the length of the body of the response.
RPC_ReturnCode field that indicates the result of the request.

Body is the parameters described in the following routines as output parameters.

All requests may receive FS_PARMBAD or FS_FAIL in the return code field. FS_PARMBAD indicates that the Opcode field was invalid. FS_FAIL indicates some sort of error that should not occur such as insufficient storage. Any requests that require a lock may receive the lock return codes if a lock is not held and not available.

The include file FILEDEF.H will contain the constants necessary to use the service routine names for OpCode and FileSys to set in the SubSystem field of the RPC_RequestHdr. It will also contain the defines to describe any input fields that are constants and to describe the return codes from the various routines

Lock

Format

Lock	(char *pathname*, char *mode*) returns (char *timestamp*)

Input Parameters

pathname	Names the file or directory to be locked.
mode	Contains a value of READ or WRITE. The defines are in the include file FILEDEF.H.

Output Parameters

timestamp	Time the file or directory was last modified in seconds since 1/1/1970.

Function

Lock is used to serialize use of a file. Lock requests can only be sent to the custodian of a file. (When caching is implemented, a cluster that is a cache for a file, will also forward lock requests to the custodian.) A lock request results in further lock requests to the same pathname being honored according to the rules below.

The lock can only be released on the same connection as it was obtained. It will automatically be released if the connection goes away.

A discussion of what authorization is needed to request what type of locks is included in the section Commands – Access & Lock requirements.

The request returns a timestamp from the file, the timestamp that is returned is the last modified time. It returns a time of zero if the file does not exist. The timestamp is only returned if the return code is zero.

Lock rules –

A read request will:

get a return code of FS_HOLDLCK if the user already holds a read or write lock on the same pathname.

get a return code of FS_WRITLCK if someone else holds a write lock on the pathname.

get a return code of FS_SUCCESS otherwise.

A write request will:

> get a return code of FS_HOLDLCK if the user already holds a write lock on pathname.
>
> get a return code of FS_PROMLCK if the user holds the only read lock on pathname. The lock will be changed to a write lock.
>
> get a return code of FS_READLCK or FS_WRITLCK if someone else holds a read or write lock on pathname.
>
> get a return code of FS_SUCCESS otherwise.

Errors

FS_NOTCUST	Cluster is not a custodian of pathname
FS_NOTAUTH	Not authorized for requested kind of lock
FS_NOPARENT	Parent directory does not exist
FS_PARMBAD	Pathname or mode invalid
FS_READLCK	File is read locked
FS_WRITLCK	File is write locked
FS_HOLDLCK	You already hold the lock
FS_PROMLCK	The lock was promoted from read to write

Unlock

Format

Unlock	(char *pathname*)

Input Parameters

pathname	The string representing the file or directory to have its lock released.

Function

Unlock releases a previously obtained lock. It must be issued over the same connection that the Lock call used.

Errors

FS_NOTCUST	Cluster is not a custodian of pathname
FS_PARMBAD	Pathname invalid
FS_NOLOCK	No lock held

WhereIs

Format

WhereIs	(char *pathname*) returns (char *custodian*, char *backup*, char *prefix*)

Input Parameters

pathname	The string naming the file or directory whose location is to be determined.

Output Parameters

custodian	The custodian of the file or directory
backup	The names of the clusters that contain backup copies of the file or directory separated by tab characters.
prefix	The briefest prefix of pathname all the descendents of which have the returned custodian.

Function

WhereIs is used to find the location of files within the file system.

Errors

FS_PARMBAD	Pathname invalid

Fetch

Format

Fetch	(RPC_BulkDescriptor *file*, char *pathname*) returns (char *sysmodtime*, char *usermodtime*)

Input Parameters

file	The RPC_BulkDescriptor that causes the requested file to be transferred.
pathname	The string representing the file to be fetched.

Output Parameters

sysmodtime	This is the time the file was last stored in seconds since 1/1/70.
usermodtime	This is the user last stored time. The user can override this on a store or a SetFileStat. If not overridden it will be the same as sysmodtime.

Function

Fetch is used by a workstation to request a file from VICE. The file server looks the file up in its local file system. If it has a copy of the file and is the custodian of the file, or the file is a replicated file, it returns it to the requester. If it has a Backup copy of the file, a check is made with the custodian to see if the file is up to date. If it is up to date it is returned, if it isn't up to date a current copy is fetched from the custodian, and it is returned, if the custodian can not be contacted, the copy that is present is returned with an FS_NOCUST return code. If it does not have a copy of the file, it returns an error indication.

Fetch also returns the system and user last modified times. If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTCUST	Cluster is not a custodian for this file (returns WhereIs data)
FS_NOCUST	A copy of the file is returned, but it may not be the latest copy
FS_NOTAUTH	The user is not authorized to read the file
FS_NOTFILE	Pathname is not a file
FS_PATHBAD	Pathname does not exist
FS_PARMBAD	Pathname not formed correctly
FS_NOPARENT	Parent is not a directory

Store

Format

Store	(RPC_BulkDescriptor *file*, char *pathname*, char *usermodtime*)

Input Parameters

file	The RPC_BulkDescriptor from the client that describes the file to be stored.
pathname	The string representing the file to be stored.
usermodtime	The time the user wants stored in the user time field.

Function

Store is used to save a file in the file system. If the file has a type of replicated or backup, any replicated or backup clusters are updated automatically.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTCUST	Cluster is not a custodian of pathname
FS_NOTAUTH	Not authorized to store the file
FS_NOPARENT	Parent directory does not exist
FS_READLCK	File is read locked
FS_PROMLCK	The lock was promoted from read to write
FS_NOTFILE	pathname not a file
FS_PARMBAD	Pathname not formed correctly

Remove

Format

Remove	(char *pathname*)

Input Parameters

pathname	The string representing the file to be removed.

Function

Remove deletes the requested file.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTCUST	Cluster is not a custodian of pathname
FS_NOTAUTH	Not authorized to remove the file
FS_NOPARENT	Parent directory does not exist
FS_READLCK	File is read locked
FS_PROMLCK	The lock was promoted from read to write
FS_PATHBAD	File does not exist
FS_PARMBAD	The pathname was malformed
FS_NOTFILE	Pathname not a file

GetFileStat

Format

GetFileStat	(char *pathname*, char *status*) returns (char *status*)

Input Parameters

pathname	The string representing the file or directory to queried.
status	The parameter showing what status is requested. This string contains the names of the entries to be returned separated by tabs. An asterisk (*) is allowed to be the last character in any field and means that any entry that starts with the characters before the asterisk are to be returned. For more information see FileStat Format in the appendix.

Output Parameters

status	This is the status data returned. The format of the character string is that the string is broken into entries by newline characters and the entries are broken into fields by tab characters. The first field in each entry is the name of the entry. For more information see FileStat Format in the appendix.

Function

GetFileStat formats the data about the file or directory into entries. Each entry contains the name and value of the information. This allows the information to be tailored to the request and be expanded in the future without a need for recoding.

The status is returned based on the status requested. The status is a single character string, entries within the string are separated by newline characters, and fields within the entries are separated by tab characters.

The input status field is used to specify which status fields are requested. Each field contains the name of an entry to return. There is a single wild card character, which is an "*". This character must be the last character in a field request, and results in a match for all entries that start with the characters before the "*". If a single field, that contains only an "*", or is null, is received, all status information is returned.

The output is a character string that contains the requested named status information. The first field in each entry is the name of the entry, subsequent fields contain the status information itself. The following entry names are maintained by the file system, users are allowed to invent new ones of their own.

The system entries maintained for each file or directory are:

whether a directory or file (STAT_CAT)

data last modified date (STAT_MDATE)
status last modified date (STAT_SDATE)
user last modified date (if file) (STAT_UDATE)
length of file (STAT_LENGTH)
custodian (STAT_CUST)
backup clusters (STAT_BACKUP) –may contain multiple fields
type of directory and any member files (STAT_TYPE)
access rights for the owner, anyuser and current user of the file (STAT_OAUTHO, STAT_AAUTH, STAT_UAUTH)
name of the owner of the file (STAT_OWNER)

The names in parentheses refer to the names assigned in filedef.h.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTCUST	Cluster is not a custodian
FS_NOTAUTH	Not authorized to read file or directory
FS_PATHBAD	Pathname does not exist
FS_NOPARENT	Parent is not a directory
FS_PARMBAD	pathname is malformed

GetMember

Format

GetMember	(char *pathname*) returns(char *status*)

Input Parameters

pathname	The string representing the directory to return the list of members for.

OutPut Parameter

status	The names of the members of the directory.

Function

GetMember returns the list of members of a directory. The names are in the same format as data from GetFileStat and will have the STAT_MEMBER name on the entry.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTCUST	Cluster is not a custodian
FS_NOTAUTH	Not authorized to read file or directory
FS_NOTDIR	Pathname is not a directory
FS_PATHBAD	Pathname does not exist
FS_NOPARENT	Parent is not a directory
FS_PARMBAD	pathname is malformed

SetFileStat

Format

SetFileStat	(char *pathname*, char *status*)

Input Parameters

pathname	The string representing the file to be changed.
status	The new status information. The information is in the same format as that returned from GetFileStat. For more information see FileStat Format in the appendix.

Function

SetFileStat is used to maintain a list of properties about a file or directory. Any information that the user desires can be stored in the property list. The information is supplied as named properties and stored in the same manner as the system properties. The stored information is retrieved by GetFileStat.

If the directory is replicated or has backup clusters, the changes are sent to the effected clusters.

The input data is passed as a status string. If an entry has only one field, the field is deleted. If an entry has more than one field the field is replaced or added.

The user cannot change or define properties whose name start with "sys".

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_PATHBAD	Pathname is invalid
FS_PARMBAD	Status information is invalid or pathname is malformed
FS_NOTCUST	Cluster is not a custodian of pathname
FS_NOTAUTH	Not authorized to update the directory status
FS_NOPARENT	Parent directory does not exist
FS_READLCK	File is read locked
FS_PROMLCK	The lock was promoted from read to write

SetBackup

Format

SetBackup	(char *pathname*, char *backup*)

Input Parameters

pathname	The pathname for the directory that is to have its backup information changed.
backup	The new backup information. The first field is backup the next fields contain clusters that are backups for this directory.

Function

SetBackup is used to change the backup clusters for a directory. The information is a single entry in the same format as that retrieved from GetFileStat with a request for backup data. The request will cause the clusters listed in fields after the first to become backups for all of the files in the directory referred to by pathname. Pathname must refer to a directory, all clusters listed must already have copies of the parent directory (because the parent is replicated, or because the parent is also backed up on the clusters). A request with no clusters listed will cause all backup clusters to be removed.

This call requires administrative authorization. The process of moving necessary files is not complete when the call returns. It is done asynchronously, and may take quite a while.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTAUTH	The user must have administrative authorization
FS_NOTCUST	Cluster is not a custodian of pathname
FS_REPL	A replicated directory can not have backup sites
FS_NOPARENT	Parent directory does not exist
FS_READLCK	File is read locked
FS_NOTEMPTY	A backup site cannot be deleted if it contains another directory
FS_PROMLCK	The lock was promoted from read to write
FS_PARMBAD	The backup list is invalid or pathname is malformed.
FS_PATHBAD	Pathname does not exist
FS_NOTDIR	Pathname is not a directory

SetAccessList

Format

SetAccessList	(char *pathname*, char *access*)

Input Parameters

pathname	The pathname for the directory that is to have its authorization data changed.
access	The new access list

Function

SetAccessList is used to change the access list for a directory. The information is in the same format as that retrieved from GetAccessList. Pathname must refer to a directory. This function will be filled out more fully when the authorization document is complete.

A write lock must be held on pathname. This call requires administrative authorization.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTAUTH	The user is not authorized to change auth data
FS_PARMBAD	The auth data is invalid or pathname is malformed
FS_NOTCUST	Cluster is not a custodian of pathname
FS_NOPARENT	Parent directory does not exist
FS_READLCK	File is read locked
FS_PROMLCK	The lock was promoted from read to write
FS_NOTDIR	Pathname is not a directory

GetAccessList

Format

GetAccessList	(char *pathname*) returns (char *access*)

Input Parameters

pathname	The pathname for the directory that is to have its authorization data changed.

Output Parameters

access	The access list

Function

GetAccessList is used to retrieve the access list from a directory. The information is in the same format as that sent to SetAccessList. Pathname must refer to a directory. This function will be filled out more fully when the authorization document is complete.

This call requires administrative authorization.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTAUTH	The user is not authorized to change auth data
FS_PARMBAD	The auth data is invalid or pathname is malformed
FS_NOTCUST	This is not the custodian of pathname
FS_NOPARENT	Parent not a directory
FS_NOTDIR	Pathname is not a directory

TestAuth

Format

TestAuth	(char *pathname*, char *type*) returns (char *datamodified*, char *statusmodified*)

Input Parameters

pathname	The pathname for the directory that is to have its authorization data changed.
type	The type of authorization requested. This can be set to FS_READ or FS_WRITE.

Output

datamodified	Time the file data was last modified in seconds since 1/1/1970.
statusmodified	Time the file status was last modified in seconds since 1/1/1970.

Function

TestAuth is used to check if a user is authorized to a file or directory. The type is the same as used in a Lock request.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_PARMBAD	The pathname is malformed, or type is invalid
FS_NOTCUST	Not the custodian
FS_NOTAUTH	The user is not authorized to pathname
FS_NOPARENT	The pathname does not have a parent

MakeDir

Format

MakeDir	(char *pathname*, char *custodian*, char *normal*)

Input Parameters

pathname	The string representing the directory to be created.
custodian	The cluster that is to be custodian of the new file or directory if it is different from the parent directory.
normal	This changes the type to normal from replicated

Function

MakeDir creates a new directory.

If specified, the custodian operand allows the directory to have a different custodian than its parent. The request must be directed to the cluster that is the custodian of the parent directory. It creates the directory and new .admin file, and sends copies of them to the new custodian. This operand requires the user to be a system administrator.

When it creates the .admin file. The file type in the .admin file for the new directory is set to what the file type is in the parent directory. The custodian and backup clusters are the same as in the parent unless a new custodian is specified on the call. If a new custodian is specified, it has no backup clusters.

If the parent directory has a type of replicated or backup, the request is sent to the effected cluster file servers (all servers for replicated, indicated servers for backup).

The caller must hold a write lock on the requested pathname.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTAUTH	The user not authorized
FS_NOTCUST	Cluster is not a custodian of pathname
FS_READLCK	File is read locked
FS_PROMLCK	The lock was promoted from read to write
FS_NOPARENT	The parent of pathname is not a directory
FS_DUPPATH	Directory already exists
FS_PARMBAD	Pathname is malformed

RemoveDir

Format

RemoveDir	(char *pathname*)

Input Parameters

pathname	The string representing the directory to be removed.

Function

RemoveDir removes a directory from the file system. It only deletes the directory if they are empty.

If the parent directory has a type of replicated or backup, the request is sent to the effected cluster file servers (all servers for replicated, indicated servers for backup).

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTAUTH	The user not authorized
FS_NOTCUST	Cluster is not a custodian of pathname
FS_NOPARENT	Parent directory does not exist
FS_READLCK	File is read locked
FS_PROMLCK	The lock was promoted from read to write
FS_PARMBAD	Pathname is malformed
FS_NOTDIR	Pathname is not a directory
FS_NOTEMPTY	Directory is not empty

ChangeOwner

Format

ChangeOwner	(char *pathname*, char *newowner*)

Input Parameters

pathname	The string representing the directory to change the owner of
newowner	The string representing the new owner

Function

This allows the name of the owner of a directory to be changed

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTCUST	Cluster is not a custodian
FS_PATHBAD	Pathname does not exist
FS_NOTAUTH	The user not authorized
FS_NOPARENT	Parent directory does not exist
FS_READLCK	File is read locked
FS_PROMLCK	The lock was promoted from read to write
FS_PARMBAD	Pathname is malformed

Rename

Format

Rename	(char *oldpathname*, char *newpathname*)

Input Parameters

oldpathname	The string representing the file to be renamed.
newpathname	The string representing the new name of the file.

Function

This function allows a file to be renamed.

The user must be authorized to delete the old file and create the new file. The old file must exist and the new file must not. Both files must have the same custodian.

If the return code is FS_NOTCUST it will return the same data as WhereIs.

Errors

FS_NOTCUST	Cluster is not a custodian
FS_NOTFILE	Old pathname is not a file
FS_NOTAUTH	The user not authorized
FS_NOPARENT	Parent directory does not exist
FS_READLCK	File is read locked
FS_BADCUST	The new name would have a different custodian than the old
FS_PROMLCK	The lock was promoted from read to write
FS_PARMBAD	Pathname is malformed
FS_DUPPATH	New pathname already exists

Data Structures

.admin and descriptor files

These files are used to keep what is actually directory extension information. These files are kept in a .admin directory which is added to each system directory. The .admin file for the directory contains at least the custodian and the type of the file. It also contains the authorization data. The information is propagated to any directory that is constructed as a member of this one. The types are:

Normal

> This is a file or directory that is treated normally. All requests are synchronized by the custodian.

Replicated

> This is a file or directory that exists in all clusters. Clusters are notified by the custodian, if the file changes. Fetches are allowed without checking with the custodian.

The .admin file also contains the name of the custodian cluster and the names of any backup clusters.

The type and custodian information of a *file* is the same as that of its *parent directory*. Any sub-directories contain their own information.

It is also possible for users to supply named properties about files and directories. For directories that information is kept in the .admin file, for files, a shadow file of the same name is kept in the .admin directory to hold the necessary information

Commands – Access & Lock requirements

Access lists are used to control access to files and directories.

The following is a list of the commands and the access needed to use them, as well as any locks that must be obtained.

Request			Access Needed	Lock Needed	Custodian
GetFileStat		File	Read	None	Yes
		Dir	Lookup		
SetFileStat		File	Write	Write	Yes
		Dir	Insert & Delete		
Fetch			Read	None	No
Store			Write (if old) Insert (if new)	Write	Yes
Remove			Delete	Write	Yes
MakeDir			Insert	Write	Yes
RemoveDir			Delete on Parent	Write	Yes
Lock	Read		Lock	None	Yes
	Write	Any	Admin |owner | Admin group		
		None	Insert		
		File	Delete |Insert		
		Dir	(Delete & Insert) | Delete on Parent		
UnLock			None	None	Yes
Whereis			None	None	No
SetAccessList			Admin |Owner	Write	Yes
GetAccessList			Lookup	None	Yes
SetBackup			In Admin group	Write	Yes
TestAuth	Read	File	Read	None	Yes
		Dir	Lookup		
	Write	File	Write		
		Dir	Insert & Delete		

FileStat format

This is the information this is passed back on a GetFileStat request or supplied on a Set-FileStat request. It consists of a number of formatted fields. The information is passed as a character string. The character string is broken into entries by newline characters, and the entries are broken into fields by tab characters. The first field within each entry is the name of the entry, any subsequent fields are data. It is possible for certain entries to contain more than two field (such as the names of members in a directory). Most entries will contain only two field.

In addition to the entries supplied by the file system, users may define and name their own entries. The user entries cannot have the same name as any file system supplied entry. All system entries start with the character string "sys".

The entry names that are supplied are:

syscategory	–whether a directory or file
sysd_modify	–data last modified date
sysd_modify	–status last modified date
date_modify	–user last modified date (file only)
syslength	–length of file
sysowner	–the userid of the file owner
syscustodian	–custodian
sysautho	–access rights of file owner
sysauthu	–access rights of the current user
sysautha	–access rights of any user
sysbackup	–backup clusters
systype	–type of directory and any member files

Of the system defined entries, only backup may contain multiple fields.

User defined entries can be added, changed or deleted.

On input to GetFileStat the entries to be retrieved can be specified. This is done by building a character string that contains a field for each entry desired. Each field contains the name of the entries to return. A simple wild card character "*" is supported. The wild card character must be the last character in the field and will return all entries which match up to the point of the "*". For example a request for "sys*" would return all of the system data. To retrieve all entries, a single field containing only an "*" or a null field is requested. This results in a match on all entries.

Include Definitions

These definitions are used to set and test fields used by the file system.

```
/*
 *     These are the defines used to interface to the File System
 */

/*     define for SubSystem field of RPC_RequestHdr                  */
#define FILESYS "fs"            /*SubSytem ID for the File Sys       */
#define FILESYSID 1
#define FILEPROTO 1

/*                    defines    for    limits    on    user    and    workstation
IDs          */
#define    MAXUID    8                    /*    max    length    of    user
ID           */
#define MAXWSID 32              /* max length of workstation ID      */

/*     defines for OpCode field of RPC_RequestHdr                    */
#define CONNECTFS 1             /* OpCode for ConnectFS              */
#define DISCONNECTFS 2          /* OpCode for DisconnectFS           */
#define FETCH 3                 /* OpCode for Fetch                  */
#define STORE 4                 /* OpCode for Store                  */
#define REMOVE 5                /* OpCode for Remove                 */
#define MAKEDIR 6               /* OpCode for MakeDir                */
#define REMOVEDIR 7             /* OpCode for RemoveDir              */
#define GETFILESTAT 8           /* OpCode for GetFileStat            */
#define SETFILESTAT 9           /* OpCode for SetFileStat            */
#define LOCK 10                 /* OpCode for Lock                   */
#define UNLOCK 11               /* OpCode for UnLock                 */
#define WHEREIS 12              /* OpCode for WhereIs                */
#define SETACCESSLIST 13         /* OpCode for SetAccessList         */
#define GETACCESSLIST 14         /* OpCode for GetAccessList         */
#define RENAME 15               /* OpCode for Rename                 */
#define SETBACKUP 16            /* OpCode for SetBackup              */
#define TESTAUTH 17             /* OpCode for TestAuth               */
#define CHANGEOWNER 18          /* OpCode for ChangeOwner            */
#define GETMEMBER 19            /* OpCode for GetMember              */

/*     defines for mode operand to Lock                              */
#define LOCK_WRITE "WRITE" /* WRITE operand                          */
#define LOCK_READ  "READ"/* READ operand                             */

/*     defines for system maintained properties                      */
#define STAT_MEMBER "sysmember"  /* names of members in a dir        */
#define STAT_CAT    "syscategory" /* category -file or dir           */
#define STAT_LENGTH "syslength"     /* length of file or directory   */
#define STAT_SDATE  "sysd_status" /* status last modified date       */
#define STAT_MDATE  "sysd_modify" /* date last modified date         */
```

```
#define STAT_UDATE  "date_modify" /* user last modified date        */
#define STAT_TYPE   "systype"     /* type                           */
#define STAT_CUST   "syscustodian" /* custodian name                */
#define STAT_BACKUP "sysbackup"   /* backup cluster names           */
#define STAT_OWNER  "sysowner"    /* owner name                     */
#define STAT_UAUTH  "sysauthu"    /* rights for current user*/
#define STAT_OAUTH  "sysautho"    /* rights for owner               */
#define STAT_AAUTH  "sysautha"    /* rights for anyuser             */

/*      defines for categories of files                             */
#define STAT_DIR    "directory"   /* pathname is a directory        */
#define STAT_FILE   "file"        /* pathname is a file             */

/*      defines for types of files                                  */
#define STAT_NORM   "normal"      /* pathname is normal             */
#define STAT_REPL   "replicated"  /* pathname is replicated         */

/*      defines for status field and entry separators               */
#define STAT_FIELD "\t"           /* field separator                */
#define STAT_FIELDC '\t'          /* field separator -character     */
#define STAT_ENTRY "\n"           /* entry separator                */
#define STAT_ENTRYC '\n'          /* entry separator -character     */
#define STAT_RECORD "\0"          /* record terminator              */
#define STAT_RECORDC '\0'         /* record terminator -character   */

/*      return codes from the File System -in RPC-ReturnCode        */
#define FS_SUCCESS 0        /* operation successful                 */
#define FS_PARMBAD 1        /* input parameter is invalid           */
#define FS_NOTCUST 2        /* the call was not directed to the     */
                            /* custodian of the file or directory   */
#define FS_NOTAUTH 3        /* not authorized to perform the request */
#define FS_NEWCUST 4        /* new name has different custodian     */
#define FS_READLCK 5        /* request failed because the file is   */
                            /* read locked                          */
#define FS_WRITLCK 6        /* request failed because the file is   */
                            /* write locked                         */
#define FS_NOPARENT 7       /* request failed because the parent of */
                            /* pathname does not exist              */
#define FS_NOCUST 8         /* custodian is partitioned             */
#define FS_NOLOCK 9         /* a needed lock is not held            */
#define FS_PATHBAD 10       /* pathname is not known                */
#define FS_NOTFILE 11       /* pathname does not resolve to a file  */
#define FS_NOTDIR 12        /* pathname does not resolve to a       */
                            /* directory                            */
#define FS_DUPPATH 13       /* pathname already exists              */
#define FS_NOTEMPTY 14      /* the directory is not empty           */
#define FS_FAIL 16          /* miscellaneous failures               */
#define FS_HOLDLCK 17       /* already hold requested lock          */
#define FS_PROMLCK 18       /* a read lock has been changed to write */
#define FS_REPL 19          /* the directory is replicated          */
#define FS_BADCUST 20       /* the parent is not on the cluster     */
```