CITI Technical Report 90–4

Synopsis of Distributed File System Protocols

Peter Honeyman
honey@citi.umich.edu

ABSTRACT

This manuscript gives a terse description of the following file system protocols:

- NFS Sun Microsystem's Network File System protocol, Version 2 (the current released version), and Version 3 (to be released in the 47 th quarter of 1989).
- RFS AT&T's Remote File Sharing protocol.
- AFS Andrew File System protocol.
- AFP AppleTalk Filing Protocol (AppleShare).
- DDM IBM's Distributed Data Management protocol.
- Sprite An operating system under development at University of California, Berkeley.
- NFILE A file access protocol devised for Symbolics computers.

According to McLuhan, and I have him right here, data collection yields to pattern recognition, so this enumeration should become a classification, turning a crib sheet into a taxonomy.

Please send corrections to honey@citi.umich.edu.

October 10, 1990

Synopsis of Distributed File System Protocols

Peter Honeyman
`honey@citi.umich.edu`

NFS

Message	Description
`NULL`	Do nothing.
`GETATTR`	Get file attributes. Returns the current attributes of the file with the given fhandle.
`SETATTR`	Set file attributes. Sets the attributes of the file with the given fhandle. Returns the new attributes.
`ROOT`	Obsolete.
`LOOKUP`	Lookup file name. Returns an fhandle and file attributes for file name in a directory.
`READLINK`	Read from symbolic link. Returns the string in the symbolic link at the given fhandle.
`READ`	Read from file. Returns some data read from the file at the given fhandle.
`WRITECACHE`	Obsolete.
`WRITE`	Write to file. Returns attributes of a file after writing some data to it.
`CREATE`	Create a file or directory. Creates a file or directory with given attributes and returns those attributes and an fhandle for the new object.
`REMOVE`	Remove a file or directory. Remove named file from parent directory.
`RENAME`	Rename file. Give an object a new name in the named directory.
`LINK`	Create link to an object. Create a hard link from one named object to another in the named directory.
`SYMLINK`	Create symbolic link. Create a symbolic link with the given attributes to the given path name in the named directory.
`MKDIR`	Make a directory. Create a directory with the given name, parent directory, and attributes. Returns a file handle and attributes for the new directory.
`RMDIR`	Remove a directory. Remove the given directory name from the given parent directory.
`READDIR`	Read from directory.
`STATFS`	Get file system statistics. Obsoleted by `GETFSATTR` and `GETFSINFO`.

NFS Mount Protocol

Message	Description
`NULL`	Do nothing.
`MNT`	Add mount entry.
`DUMP`	Return mount entries.
`UMNT`	Remove mount entry.
`UMNTALL`	Remove all mount entries.
`EXPORT`	Return export list.

NFS Version 3 (Draft of 11 Sept 88)

Message	Description
`GETRESINFO`	Get resource information.
`NULL`	Do nothing.
`GETATTR`	Get file attributes. Returns the current attributes of the file with the given fhandle.
`SETATTR`	Set file attributes. Sets the attributes of the file with the given fhandle. Returns the new attributes.
`STOREATTR`	Store file attributes. Save extended attributes without side effects.
`ACCESS`	Check access permission. Check what type of access is allowed to the named object.
`INACTIVE`	Advise of inactive file handle. Advisory only.
`LOOKUP`	Lookup file name. Returns an fhandle and file attributes for file name in a directory.
`READ`	Read from file. Returns some data read from the file at the given fhandle.
`WRITE`	Write to file. Returns attributes of a file after writing some data to it.
`WRITECACHE`	Write to cache. (Optional.) Writes data into the server's cache for a regular file.
`ZERO`	Write zeroes to file.
`CREATE`	Create a file or directory. Creates a file or directory with given attributes and returns those attributes and an fhandle for the new object.
`REMOVE`	Remove a file or directory. Remove named file from parent directory.
`RENAME`	Rename file. Give an object a new name in the named directory.
`LINK`	Create link to an object. Create a hard link from one named object to another in the named directory.
`SYMLINK`	Create symbolic link. Create a symbolic link with the given attributes to the given path name in the named directory.
`READLINK`	Read from symbolic link. Returns the string in the symbolic link at the given fhandle.
`CREATEUNIQ`	Create a file or directory. Create a uniquely named file or directory in the named directory. This is useful, for example, in creating temporary files where the name does not matter.
`READDIR`	Read from directory.
`LINKAGE`	Get file system attributes. Translate a file handle for use by some other service.
`GETFSATTR`	Get file system attributes. Obtain volume capacity and time statistics.
`GETFSINFO`	Get file system information. Obtain NFS-related server information.

AFS

Message	Description
`FetchData`	This call retrieves the contents of the file specified by *Fid* from the file server to the client. The *Position* parameter specifies the first byte to be fetched by this call; 0 specifies the first byte. The *Length* parameter specifies the number of bytes desired; a value of 0xFFFFFFFF indicates the entire file. The *OutStatus* parameter returns the status of the file, and the *CallBack* parameter may grant a callback promise to the caller.
`FetchACL`	This call fetches the access control list associated with the directory *Fid*. The *AccessList* parameter is an INOUT parameter that specifies, on input, the largest size string the client is willing to receive as input, and returns, as output, the access control list. The *OutStatus* parameter returns the current status of the directory. Note that a new callback is not returned by this call; any existing callback promise remains in effect, however.
`FetchStatus`	This call retrieves the status information associated with the file or directory specified by *Fid*. On return, the *OutStatus* parameter contains the file's status information. The *CallBack* parameter may contain, on return, a new callback promise from the file server.
`StoreData`	This call updates the data and status portions of the file named by the *Fid* parameter. The *InStatus* parameter allows setting of the *ClientModTime* field; all other fields must remain unchanged. The *Position* and *Length* parameters specify a starting byte position and length for this transfer. The *OutStatus* parameter returns the updated directory status information.
`StoreACL`	This call updates the access control information associated with a file or directory. The *Fid* parameter specifies the directory whose ACL will change. The *InStatus* parameter may someday allow the updating of status information simultaneously; at present it must be set to all "noop" values. The *AccessList* parameter contains the new ACL, and the *OutStatus* parameter returns the updated directory status information.
`StoreStatus`	This call updates the status information associated with the file or directory *Fid*. The *InStatus* parameter specifies the new file status. The *OutStatus* parameter returns the updated directory status information, since other status fields (such as the *ServerModTime* field) may change as a side-effect of making any status changes.
`RemoveFile`	This call removes a file or symbolic link (but not a directory) from the file system. The *DirFid* parameter specifies the directory from which to remove the entry. The *Name* parameter specifies the name of the file or symbolic link to be removed. The *OutStatus* parameter returns the updated directory's status information. The cache manager is responsible for decrementing the link count in the file's associated cached status by 1.

`CreateFile` This call is used to create a file (but not a symbolic link or a directory). The *DirFid* parameter specifies the directory in which to create the file. The *Name* parameter specifies the name of the file to be created. The *InStatus* parameter specifies the initial status fields for the new file. After the call completes, the *OutFid* parameter contains the file's file ID, and the new file's status is returned in *OutFidStatus*. In addition, the updated directory's status is returned in *OutDirStatus* and a new callback may be returned in *CallBack*.

If the call succeeds, it is the cache manager's responsibliity to either create an entry locally in the directory specified by *DirFid*, or to invalidate this directory's cache entry.

`Rename` This call renames the file *OldName* in the directory specified by *OldDirFid* to be the file *NewName* in the directory *NewDirFid*. The updated directory status for both directories is returned in *OutOldDirStatus* and *OutNewDirStatus*. If the two directories are the same, the same status is returned twice.

The rename must not result in hard links existing to the same object from two different directories, or the error code EXDEV (18 decimal) will be returned.

If a directory is moved from one directory to another, the cache manager must either update the cached copy of the moved directory in order to update its ".." entry, or the cache manager must invalidate the cache entry for the moved directory. The directory link counts will be updated by the server in the returned directory status blocks, *OutNewDirStatus* and *OutOldDirStatus*.

`Symlink` This call creates a symbolic link in the directory specified by *DirFid* with the name *Name*. The contents of the new symbolic link are specified by *LinkContents*; this is the target of the new symbolic link. The *InStatus* parameter should specify the new link's *ClientModTime* and *UnixModeBits* fields. Note that a symbolic link with protection mode 0644 (octal) is treated by the Andrew file system as an Andrew file system mount point. The parameter *OutFid* returns the file ID of the newly-created link, and the *OutFidStatus* parameter gives the complete status information for this newly-created entity. The *OutDirStatus* parameter provides the updated directory status information.

Usage notes: note that no callback is returned on the new symbolic link, since symbolic links can never change, they can only be deleted. It is recommended that the cache manager make use of this fact.

`Link` This call creates a hard link to file specified by *ExistingFid*, with the name *Name* in the directory specified by *DirFid*. The file named by *ExistingFid* must not be a directory, and must furthermore be in the directory *DirFid*, that is, the same directory as the new link will be created. The parameter *OutFidStatus* returns the updated file status, and the *OutDirStatus* returns the updated directory status.

`MakeDir` This call creates a new directory named *Name* in the parent directory *DirFid*. The *InStatus* parameter provides the initial *UnixModeBits* and *ClientModTime* values; the others are ignored. The new directory's file ID is returned in *OutFid*, and the new directory's status is returned in *OutFidStatus*. The parent directory's updated status is returned in *OutDirStatus* and a callback for the newly-created directory may be returned in *CallBack*.

It is the cache manager's responsibliity to either create an entry locally in the directory specified by *DirFid*, or to invalidate this directory's cache entry.

`RemoveDir` This call removes a directory from the file system. The parameter *DirFid* specifies the parent directory of the directory to be removed. The parameter *Name* gives the name of the directory to be removed (relative to its parent). The parent directory's updated status is returned in *OutDirStatus*.

The directory must be empty (containing only entries for "." and ".."), otherwise this call will fail. It is the cache manager's responsibliity to either create an entry locally in the directory specified by *DirFid*, or to invalidate this directory's cache entry.

`SetLock` This call sets an advisory lock of type *Type* on the file specified by *Fid*. Advisory locks do not interfere with any other operations, except for other SetLock calls. If *Type* is equal to LockRead (0) a read (shared) lock is obtained; if *Type* is equal to LockWrite (1), an exclusive lock is granted. This call never blocks, rather, if the lock can not be obtained, the code EWOULDBLOCK (35) is returned and no lock is granted. In this case, the client is expected to periodically retry this operation.

Advisory locks normally timeout after 180 seconds. The ExtendLock call should be issued every 60 seconds or so in order to prevent a lock from timing out.

The caller must have "k" access to a file or directory in order to set advisory locks.

`ExtendLock` This call extends an advisory lock for the file specified by *Fid*. Advisory locks normally timeout after three minutes (180 seconds); this call should be issued periodically by the cache manager if locks of longer duration are required.

The caller must have "k" access to a file or directory in order to extend advisory locks.

`ReleaseLock` This call releases an advisory lock set on the file *Fid*.

The caller must have "k" access to a file or directory in order to remove an advisory lock.

`GetStatistics` This call returns statistics concerning file server throughput, resource usage and disk storage at the file server to which the call is directed. It is used for status monitoring purposes only.

`GiveUpCallBacks` This call returns an array of callback promises to a given file server. The callback promises must have been granted originally by the file server to which this call is directed. The cache manager should not attempt to give up more than 50 callback promises in any one call to a file server.

The file *CallBacks_Array* parameter specifies the new callback level desired for the corresponding element in the *Fids_Array*. In the typical case, the caller will be discarding the callback promises, and will specifiy a callback type of *DROPPED* (3).

This call always returns success, whether or not all of the callback promises did indeed exist at the specified file server.

`GetVolumeInfo` This call returns the location information associated with the volume *Volumeid* in the structure *VolumeInfo*. The *Volumeid* parameter is a character string and may be either the volume name, or an ASCII representation of a volume number, treated as an unsigned decimal integer. For instance, if volume number 0x88000000 has name "user.bozo", then the caller could pass in as the *Volumeid* either the string "user.bozo" or the string "2281701376".

`GetVolumeStatus` This call returns the status information associated with a volume. The volume ID is passed in as an integer. The *VolumeStatus* parameter is an INOUT parameter for historical reasons; none of the input values are examined. The resulting volume status block is returned in *VolumeStatus*, while the last three parameters are set to the volume name (the terminating null character is included), the offline message explaining why the volume is offline, and the volume's message of the day, explaining any other information of interest to the volume's user.

`SetVolumeStatus` This call sets the status information associated with a volume. The volume ID identifies the volume to be processed. The *VolumeStatus* structure should be set to the volume's new status. The last three parameters should indicate the new volume name, and the new messages associated with the volume. A 0-length for `ame.SeqLen`, `OfflineMsg.SeqLen` or `Motd.SeqLen` tells the server to not change that string. The only values that can be updated in the *VolumeStatus* structure are the *MinQuota* and *MaxQuota* fields; a value of -1 (0xFFFFFFFF) directs the server to leave the field unchanged.

This call returns the updated volume status to the caller in the specified INOUT parameters, just as in GetVolumeStatus.

`GetRootVolume` This call returns the name of the root volume for the appropriate cell. The name is returned in the space provided by the bounded byte string, along with its terminating null character.

`CheckToken` This call takes two input parameters, a ViceID representing the user whose token (in Kerberos terms, a ticket) is being checked for validity, and the ticket itself. The call returns 0 if and only if the ticket is still valid for user by the specified user within this cell.

`GetTime` This call returns the time of day in seconds and microseconds, in its two output parameters. The *Seconds* parameter represents the number of seconds since 1/1/70, as in the Unix time standard.

`CallBack` This routine is used to asynchronously alter the state of possibly existing callbacks at the workstation. The *fids* array lists a set of files, and the *callbacks* array lists a set of callback operations to perform on those files. As a special case, fids[i].vnode == 0 means break all callbacks on the specified volume; otherwise break the specified callback on the specified segment. In this case, there is also an implied file status change. The possible operations are: break a call back (level == DROPPED), or downgrade a callback to non-exclusive (level == SHARED). In all cases, the callback ID specified in the callback structure should be greater or equal to the callback id elected to be deleted.

It is important, when writing a cache manager, to realize that the file server is permitted to keep locked any number of data structures while making an *CallBack* call. To avoid deadlock, it is strongly advised that the cache manager process this call without ever blocking.

`InitCallbackState` Initialize callback state at the workstation, with respect to the calling server. All callbacks from the calling server should be discarded.

`Probe` This call is a noop used by the server to check that a cache manager is still running. By periodically polling apparently inactive cache managers, a file server can timeout down workstations before actually trying to break a callback promise at that workstation, allowing the operation eventually triggering a break callback operation to complete faster.

`ReceivedStore` This call will be obsolete shortly. It indicates that a call storing data into a file has broken all callbacks and set the appropriate locks, indicating that the client can proceed knowing that all future calls to the server will see the new file data, not the old.

AFP

Server Calls

Message	Description
`GetSrvrInfo`	Obtain a block of descriptive information from the server, without requiring a session to be opened.
`GetSrvrParms`	Retrieve server-level parameters.
`Login`	Establish a session with a server. A protocol version is agreed upon and the user is authenticated.
`LoginCont`	Continue the Login and authentication process with a server.
`Logout`	Terminate a session with a server.
`MapID`	Map a User ID to a User Name, or a Group ID to a Group Name.
`MapName`	Map a User Name to a User ID, or a Group Name to a Group ID.

Volume Calls

Message	Description
`OpenVol`	"Mount" a volume. It must be called before any other call can be made to access objects on the volume.
`CloseVol`	"Unmount" a volume.
`GetVolParms`	Retrieve parameters for a particular volume. The volume is specified by its Volume ID as returned from the OpenVol call.
`SetVolParms`	Set the parameters for a particular volume. The volume is specified by its VolumeID as returned from the OpenVol call.
`Flush`	Flush to disk any data relating to the specified volume that has been modified by the user.

Directory Calls

Message	Description
`SetDirParms`	Set parameters for a particular directory
`OpenDir`	Open a directory on a Variable-DirID volume and obtain its directory identifier.
`CloseDir`	Close a directory.
`Enumerate`	Enumerate the contents of a directory. The reply is composed of a number of file and/or directory parameter structures.
`CreateDir`	Create a new directory.

File Calls

Message	Description
`SetFileParms`	Set parameters for a particular file.
`CreateFile`	Create a file.
`CopyFile`	Optional. Copy a file residing on one of the server's volumes to another location on one of the server's volumes. The destination of the copy is specified by providing a VolID, DirID, and Pathname that indicate the copy's new Parent Directory.

Combined Directory-File Calls

Message	Description
`GetFileDirParms`	Retrieve parameters for an object that may be a file or directory.
`SetFileDirParms`	Set parameters for a particular object, either a file or directory.
`Rename`	Rename either a directory or a file.
`Delete`	Delete either a directory or file.
`Move`	Move (not just copy) a directory or file to another location on a single volume (source and destination must be on the same volume). An object cannot be moved from one volume to another with this call, even though both volumes may be managed by the server. The destination of the move is specified by providing a DirID and Pathname that indicate the object's new Parent Directory.

Fork Calls

Message	Description
`GetForkParms`	Retrieve parameters for a file associated with a particular open fork.
`SetForkParms`	Set parameters for a file associated with a particular open fork.
`OpenFork`	Open the data or resource fork of an existing file for the purpose of reading from it or writing to it. Each fork must be opened separately; a unique OpenForkRefnum will be returned for each.
`Read`	Read a block of data from a open fork.
`Write`	Write a block of data to an open fork.
`FlushFork`	Any Writes made to a particular file fork may be buffered by the server in order to optimize disk accesses. Within the constraints of performance, the server will try to flush (commit to disk) each file as soon as possible, yet clients can force the server to write to the disk any data buffered from previous Writes by issuing this call.
`ByteRangeLock`	Lock a range of an open fork to ensure exclusive access. Locks prevent all other users from reading or writing any bytes within the range.
`CloseFork`	Close a fork which was opened by OpenFork.

Desktop Calls

Message	Description
`OpenDT`	Open the Macintosh Dektop data base.
`FpCloseDT`	Disassociate a user from the volume's Macintosh Desktop.

Icon-Related Calls

Message	Description
`AddIcon`	Add an icon bitmap to the Desktop Database.
`GetIcon`	Retrieve an icon from the Desktop database from a FileCreator/FileType specification.
`GetIconInfo`	

Application-Related Calls

Message	Description
`AddAPPL`	Add an APPL to the Desktop Database.
`RemoveAPPL`	Remove an APPL mapping from the Desktop Database.
`GetAPPL`	Retrieve information about a particular application from the Desktop Database.

Comment-Related Calls

Message	Description
`AddComment`	Add a comment for a file or directory to the Desktop Database.
`RemoveComment`	Remove a comment from the Desktop Database.
`GetComment`	Retrieve a comment associated with a specified file or directory from the Desktop Database.

RFS

Message	Description
ACCESS	Check access permissions
SYSACCT	Do system accounting
CHDIR	Change directory
CHMOD	Change file mode
CHOWN	Change file owner
CHROOT	Change root directory
CLOSE	Close a file
CREAT	Create a file
EXEC	Exec a file
EXECE	Exec a file with an environment
FCNTL	File control
FSTAT	Stat a file (uses file descriptor)
FSTATFS	Stat a file system (uses file descriptor)
IOCTL	Ioctl
LINK	First half of link() operation
LINK1	Second half of link() operation
MKNOD	Make a device file
OPEN	Open a file
READ	Read from a file
SEEK	Seek on a file
STAT	Stat a file (uses pathname)
STATFS	Stat a file system (uses pathname)
UNLINK	Unlink a file
UTIME	Change times on file
UTSSYS	Return information about a mounted file system.
WRITE	Write a file
GETDENTS	Read directory entries in a file system independent format
MKDIR	Make a directory
RMDIR	Remove a directory
SRMOUNT	Server side of remote mount
SRUMOUNT	Server side of remote unmount
COREDUMP	Dump core
WRITEI	Internal form of write system call
READI	Internal form of read system call
RSIGNAL	Send remote signal
SYNCTIME	Synchronize time between machines
IPUT	Free a remote inode
IUPDATE	Update a remote inode
UPDATE	Write modified buffers back to disk.

DDM

Message	Description
CLOSE	Close file. This command terminates the logical connection, established by OPEN, between the requester and a file.
CLRFIL	Clear file. This command clears an existing file of all records and reinitializes it as if it had just been created.
CRTAIF	Create alternate index file. This command creates an alternate index file on the target system. The alternate index file provides a *key field* access sequence to the records in an existing base target system file. The base file can be a sequential or keyed file.
CRTDIRF	Create direct file. This command creates a direct file on the target system.
CRTKEYF	Create keyed file. This command creates a keyed file on the target system.
CRTSEQF	Create sequential file. This command creates a sequential file on the target system.
CRTSIMF	Create similar file. This command creates a file similar to an existing file. The new file is created with the same file attributes (except file name) as the existing file. Record data is *not* copied from the existing file to the new file.
DCLFIL	Declare file. The declare file command associates a declared name with a collection of file-oriented parameters.
DELDCL	Delete declare. This command deletes collection of file-oriented parameters that were created by a previous `DCLFIL` command. DELFIL This command deletes a file from the target system, releases all locks held on the file, and releases the space it occupied.
DELREC	Delete record. This command deletes the record which has an update intent placed on it.
EXCSAT	Exchange server attributes. This command is used to exchange information between servers.
GETREC	Get record. This command gets and returns the record indicated by the current cursor position.
INSRECEF	Insert at EOF. This command inserts a record at the end of the file.
INSRECKY	Insert by key value. This command inserts one or more records according to their key values wherever there is available space in the file.
INSRECNB	Insert by record number. This command inserts one or more records at the position specified by the record number parameter.
LCKFIL	Lock file. This command locks the file for subsequent use by the requester.
LODFIL	Load file. This command loads one or more records into a file.
LSTFAT	List file attributes. This command retrieves selected attributes of a file.
MODREC	Modify record. This command modifies the record which has an update intent placed on it.
OPEN	Open file. This command establishes a logical connection between the using program on the source system and the file on the target system.
RNMFIL	Rename file. This command renames an existing file.
SETBOF	Set cursor to beginning of file. This command sets the cursor to the position ahead of the first record on the file.
SETEOF	Set cursor to end of file. This command sets the cursor to the position following the last record of the file.
SETFRS	Set cursor to first record. This command sets the cursor to the first record of the file.
SETKEY	Set cursor by key. This command positions the cursor based on the key value supplied and the relational operator specified for the relational operator parameter.
SETKEYFR	Set cursor to first record in key sequence. This command sets the cursor to the first record in the key sequence.
SETKEYLM	Set key limits. This command sets the limits of the key values for subsequent SET-KEYNX (set cursor to next record in key sequence) commands.
SETKEYLS	Set cursor to last record in key sequence. This command sets the cursor to the last record of the file in key sequence order.

`SETKEYNX`	Set cursor to next record in key sequence. This command sets the cursor to the next record of the file in the key sequence order following the record currently indicated by the cursor.
`SETKEYPR`	Set cursor to previous record in key sequence. This command sets the cursor to the previous record of the file in the key sequence order preceding the record currently indicated by the cursor.
`SETLST`	Set cursor to last record. This command sets the cursor to the last record of the file.
`SETMNS`	Set cursor minus. This command sets the cursor to the record number of the file indicated by the cursor minus the number of record positions specified by the cursor displacement parameter.
`SETNBR`	Set cursor to record number. This command sets the cursor to the record of the file indicated by the record number specified by the record number parameter.
`SETNXT`	Set cursor to next record. This command sets the cursor to the next record of the file with a record number one greater than the current cursor position.
`SETNXTKE`	Set cursor to next record with equal key. This command sets the cursor to the next record in the key sequence if the key field of that record has a value equal to the value specified in the KEYVAL parameter.
`SETPLS`	Set cursor plus. This command sets the cursor to the record number of the file indicated by the cursor plus the integer number of records specified by the cursor displacement parameter (CSRDSP).
`SETPRV`	Set cursor to previous record. This command sets the cursor to the record of the file with a record number one less than the current cursor position.
`SETUPDKY`	Set update intent by key. This command places an update intent on the record that has a key value equal to the KEYNAL parameter. The cursor position is not changed.
`SETUPDNB`	Set update intent by record number. This command places an update intent on the record at the record position specified by the RECNBR parameter. The cursor position is not changed
`ULDFIL`	Unload file. This command unloads the records of a file.
`UNLFIL`	Unlock file. This command releases explicit file locks held by the requester on the file.
`UNLIMPLK`	Unlock implicit record lock. This command releases all implicit record locks on records.

Sprite

Message	Description
ECHO_1	Echo. Performed by server's interrupt handler (unused).
ECHO_2	Echo. Performed by Rpc_Server process.
SEND	Send. Like Echo, but data only transferred to server.
RECEIVE	Receive. Data only transferred back to client.
GETTIME	Broadcast RPC to get the current time.
FS_PREFIX	Broadcast RPC to find prefix server.
FS_OPEN	Open a file system object by name.
FS_READ	Read data from a file system object.
FS_WRITE	Write data to a file system object.
FS_CLOSE	Close an I/O stream to a file system object.
FS_UNLINK	Remove the name of an object.
FS_RENAME	Change the name of an object.
FS_MKDIR	Create a directory.
FS_RMDIR	Remove a directory.
FS_MKDEV	Make a special device file.
FS_LINK	Make a directory reference to an existing object.
FS_SYM_LINK	Make a symbolic link to an existing object.
FS_GET_ATTR	Get the attributes of the object behind an I/O stream.
FS_SET_ATTR	Set the attributes of the object behind an I/O stream.
FS_GET_ATTR_PATH	Get the attributes of a named object.
FS_SET_ATTR_PATH	Set the attributes of a named object.
FS_GET_IO_ATTR	Get the attributes kept by the I/O server.
FS_SET_IO_ATTR	Set the attributes kept by the I/O server.
FS_DEV_OPEN	Complete the open of a remote device or pseudo-device.
FS_SELECT	Query the status of a device or pseudo-device.
FS_IO_CONTROL	Perform an object-specific operation.
FS_CONSIST	Request that cache consistency action be performed.
FS_CONSIST_REPLY	Acknowledgement that consistency action completed.
FS_COPY_BLOCK	Copy a block of a swap file.
FS_MIGRATE	Tell I/O server that an I/O stream has migrated.
FS_RELEASE	Tell source of migration to release I/O stream.
FS_REOPEN	Recover the state about an I/O stream.
FS_RECOVERY	Signal that recovery actions have completed.
FS_DOMAIN_INFO	Return information about a file system domain.
PROC_MIG_COMMAND	Used to transfer process state during migration.
PROC_REMOTE_CALL	Used to forward system call to the home node.
PROC_REMOTE_WAIT	Used to synchonize exit of migrated process.
PROC_GETPCB	Return process table entry for migrated process.
REMOTE_WAKEUP	Wakeup a remote process.
SIG_SEND	Issue a signal to a remote process.

NFILE

Message	Description
ABORT	ABORT cleanly interrupts and prematurely terminates a single direct access mode data transfer initiated with READ.
CHANGE-PROPERTIES	CHANGE-PROPERTIES changes one or more properties of a file.
CLOSE	CLOSE terminates a data transfer, and frees a data channel.
COMPLETE	COMPLETE performs file pathname completion.
CONTINUE	CONTINUE resumes a data transfer that was temporarily suspended due to an asynchronous error.
CREATE-DIRECTORY	CREATE-DIRECTORY creates a directory on the remote file system.
CREATE-LINK	CREATE-LINK creates a link on the remote file system.
DATA-CONNECTION	DATA-CONNECTION enables the user side to initiate the establishment of a new data connection.
DELETE	DELETE deletes a file on the remote file system.
DIRECT-OUTPUT	DIRECT-OUTPUT starts and stops output data flow for a direct access file opening.
DIRECTORY	DIRECTORY returns a directory listing including the identities and attributes for logically related groups of files, directories, and links.
DISABLE-CAPABILITIES	DISABLE-CAPABILITIES causes an access capability to be disabled on the server machine.
ENABLE-CAPABILITIES	ENABLE-CAPABILITIES causes an access capability to be enabled on the server machine.
EXPUNGE	EXPUNGE causes the directory specified by pathname to be expunged.
FILEPOS	FILEPOS sets the file access pointer to a given position, relative to the beginning of the file.
FINISH	FINISH closes a file and reopens it immediately with the file position pointer saved, thus leaving it open for further I/O.
HOME-DIRECTORY	HOME-DIRECTORY returns the full pathname of the home directory on the server machine for the given user.
LOGIN	LOGIN logs the given user in to the server machine, using the password if necessary.
MULTIPLE-FILE-PLISTS	MULTIPLE-FILE-PLISTS returns file property information of one or more files.
OPEN	OPEN opens a file for reading, writing, or direct access at the server host.
PROPERTIES	PROPERTIES requests the property information about one file.
READ	READ requests input data flow for direct access openings.
RENAME	RENAME requests the server to give a file a new name.
RESYNCHRONIZE-DATA-CHANNEL	RESYNCHRONIZE-DATA-CHANNEL begins a prescribed procedure between user and server over the unsafe data channel specified by handle. The resynchronization procedure clears the data channel of any unwanted data, and restores the data channel to a safe state, ready to transfer data again.
UNDATA-CONNECTION	UNDATA-CONNECTION explicitly disestablishes a data connection from the user side.

Sources

IBM, Inc., ‘‘IBM Distributed Data Management Architecture: General Information,’’ GC21-9527 (June 1986).

Brent B. Welch, *Naming, State Management, and User-Level Extensions in the Sprite Distributed File System*, PhD Thesis, University of California, Berkeley (April 1990).

B. Greenberg and S. Keene, ‘‘NFILE — A File Access Protocol,’’ RFC 1037, USC/Information Sciences Institute (December, 1987).