Network File System (NFS) library

Generated by Doxygen 1.8.1.2

Thu Aug 1 2013 15:54:58

Contents

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

3 Data Structure Documentation

3.1 fattr_t Struct Reference

The "fattr" structure contains the attributes of a file.

Data Fields

- ftype_t type
  The type of the file.
- uint32_t mode
  The access mode encoded as a set of bits.
- uint32_t nlink
  The number of hard links to the file (the number of different names for the same file).
- uint32_t uid
  The user identification number of the owner of the file.
- uint32_t gid
  The group identification number of the group of the file.
- uint32_t size
  The size in bytes of the file.
- uint32_t block_size
  The size in bytes of a block of the file.
- uint32_t rdev
  The device number of the file if it is type NFCHR or NFBLK.
- uint32_t blocks
  The number of blocks the file takes up on disk.
- uint32_t fsid
  The file system identifier for the file system containing the file.
- uint32_t fileid
  A number that uniquely identifies the file within its file system.
- timeval_t atime
  The time when the file was last accessed for either read or write.
- timeval_t mtime
  The time when the file data was last modified (written).
- timeval_t ctime
  The time when the status of the file was last changed.

3.1.1 Detailed Description

The "fattr" structure contains the attributes of a file.

3.1.2 Field Documentation

3.1.2.1 uint32_t mode

The access mode encoded as a set of bits.

Notice that the file type is specified both in the mode bits and in the file type. The encoding of this field is the same as the mode bits returned by the stat(2) system call in UNIX.

3.1.2.2 uint32_t nlink

The number of hard links to the file (the number of different names for the same file).

3.1.2.3 timeval_t ctime

The time when the status of the file was last changed.

Writing to the file also changes "ctime" if the size of the file changes.

The documentation for this struct was generated from the following file:

- nfs.h

3.2 fhandle_t Struct Reference

The "fhandle" is the file handle passed between the server and the client.

3.2.1 Detailed Description

The "fhandle" is the file handle passed between the server and the client.

All file operations are done using file handles to refer to a file or directory. The file handle can contain whatever information the server needs to distinguish an individual file.

The documentation for this struct was generated from the following file:

- nfs.h

3.3 sattr_t Struct Reference

The "sattr" structure contains the file attributes which can be set from the client.

Data Fields

- uint32_t mode

  The access mode encoded as a set of bits.
- uint32_t uid

  The user identification number of the owner of the file.
- uint32_t gid

  The group identification number of the group of the file.
- uint32_t size

  The size in bytes of the file. Zero means that the file should be truncated.
- timeval_t atime

  The time when the file was last accessed for either read or write.
- timeval_t mtime

  The time when the file data was last modified (written).

3.3.1 Detailed Description

The "sattr" structure contains the file attributes which can be set from the client.

The fields are the same as for "fattr" above. A "size" of zero means the file should be truncated. A value of -1 indicates a field that should be ignored.

3.3.2 Field Documentation

3.3.2.1 uint32_t mode

The access mode encoded as a set of bits.

The encoding of this field is the same as the mode bits returned by the stat(2) system call in UNIX.

The documentation for this struct was generated from the following file:

- nfs.h

3.4 timeval_t Struct Reference

The "timeval" structure is the number of seconds and microseconds since midnight January 1, 1970, Greenwich Mean Time.

Data Fields

- uint32_t seconds

  The seconds portion of the time value.
- uint32_t useconds

  The micro seconds portion of the time value.

3.4.1 Detailed Description

The "timeval" structure is the number of seconds and microseconds since midnight January 1, 1970, Greenwich Mean Time.

It is used to pass time and date information.

The documentation for this struct was generated from the following file:

- nfs.h

4 File Documentation

4.1 nfs.h File Reference

Network File System (NFS) client.

Data Structures

- struct fhandle_t

  The "fhandle" is the file handle passed between the server and the client.
- struct timeval_t

The "timeval" structure is the number of seconds and microseconds since midnight January 1, 1970, Greenwich Mean Time.

- struct fattr_t

  The "fattr" structure contains the attributes of a file.

- struct sattr_t

  The "sattr" structure contains the file attributes which can be set from the client.

Macros

- #define FHSIZE 32

  The size in bytes of the opaque file handle.

- #define MAXNAMLEN 255

  The maximum number of bytes in a file name argument.

- #define MAXPATHLEN 1024

  The maximum number of bytes in a pathname argument.

Typedefs

- typedef uint32_t nfscookie_t

  A cookie provided by the server which can be used for subsequent calls.

- typedef void(∗ nfs_getattr_cb_t)(uintptr_t token, enum nfs_stat status, fattr_t ∗fattr)

  A call back function provided by the caller of nfs_getattr, executed once a response is received.

- typedef void(∗ nfs_lookup_cb_t)(uintptr_t token, enum nfs_stat status, fhandle_t ∗fh, fattr_t ∗fattr)

  A call back function provided by the caller of nfs_lookup, executed once a response is received.

- typedef void(∗ nfs_create_cb_t)(uintptr_t token, enum nfs_stat status, fhandle_t ∗fh, fattr_t ∗fattr)

  A call back function provided by the caller of nfs_create, executed once a response is received.

- typedef void(∗ nfs_remove_cb_t)(uintptr_t token, enum nfs_stat status)

  A call back function provided by the caller of nfs_remove, executed once a response is received.

- typedef void(∗ nfs_readdir_cb_t)(uintptr_t token, enum nfs_stat status, int num_files, char ∗file_names[], nfscookie_t nfscookie)

  A call back function provided by the caller of nfs_readdir, executed once a response is received.

- typedef void(∗ nfs_read_cb_t)(uintptr_t token, enum nfs_stat status, fattr_t ∗fattr, int count, void ∗data)

  A call back function provided by the caller of nfs_read, executed once a response is received.

- typedef void(∗ nfs_write_cb_t)(uintptr_t token, enum nfs_stat status, fattr_t ∗fattr, int count)

  A call back function provided by the caller of nfs_write, executed once a response is received.

Enumerations

- enum nfs_stat_t {
  NFS_OK = 0, NFSERR_PERM = 1, NFSERR_NOENT = 2, NFSERR_IO = 5,
  NFSERR_NXIO = 6, NFSERR_ACCES = 13, NFSERR_EXIST = 17, NFSERR_NODEV = 19,
  NFSERR_NOTDIR = 20, NFSERR_ISDIR = 21, NFSERR_FBIG = 27, NFSERR_NOSPC = 28,
  NFSERR_ROFS = 30, NFSERR_NAMETOOLONG = 63, NFSERR_NOTEMPTY = 66, NFSERR_DQUOT = 69,
  NFSERR_STALE = 70, NFSERR_WFLUSH = 99, NFSERR_COMM = 200 }

  The "nfs_stat" type is returned with every NFS procedure results.

- enum rpc_stat_t {
  RPC_OK = 0, RPCERR_NOMEM = 1, RPCERR_NOBUF = 2, RPCERR_COMM = 3,
  RPCERR_NOSUP = 4 }

  The "rpc_stat" type is returned when an asynchronous response is expected.

- enum ftype_t { ,
  NFREG = 1, NFDIR = 2, NFBLK = 3, NFCHR = 4,
  NFLNK = 5 }

  The enumeration "ftype" gives the type of a file.

Functions

- enum rpc_stat nfs_init (const struct ip_addr *server)

  Initialises the NFS subsystem.
- void nfs_timeout (void)

  Handles packet loss and retransmission.
- enum rpc_stat nfs_mount (const char *dir, fhandle_t *pfh)

  A synchronous function used to mount a file system over the network.
- enum rpc_stat nfs_print_exports (void)

  Synchronous function used to print the directories exported by the server.
- enum rpc_stat nfs_getattr (const fhandle_t *fh, nfs_getattr_cb_t callback, uintptr_t token)

  An asynchronous function used for retrieving the attributes of a file.
- enum rpc_stat nfs_lookup (const fhandle_t *pfh, const char *name, nfs_lookup_cb_t callback, uintptr_t token)

  Asynchronous function used for retrieving an NFS file handle (fhandle_t) of a file located on the server.
- enum rpc_stat nfs_create (const fhandle_t *pfh, const char *name, const sattr_t *sattr, nfs_create_cb_t callback, uintptr_t token)

  An asynchronous function used for creating a new file on the NFS file server.
- enum rpc_stat nfs_remove (const fhandle_t *pfh, const char *name, nfs_remove_cb_t callback, uintptr_t token)

  An asynchronous function used for removing an existing file from the NFS file server.
- enum rpc_stat nfs_readdir (const fhandle_t *pfh, nfscookie_t cookie, nfs_readdir_cb_t callback, uintptr_t token)

  An asynchronous function used for reading the names of the files that are stored within the given directory.
- enum rpc_stat nfs_read (const fhandle_t *fh, int offset, int count, nfs_read_cb_t callback, uintptr_t token)

  An asynchronous function used for reading data from a file.
- enum rpc_stat nfs_write (const fhandle_t *fh, int offset, int count, const void *data, nfs_write_cb_t callback, uintptr_t token)

  Asynchronous function used for writing data to a file.
- int nfs_test (char *mnt)

  Tests the NFS system using the provided path as a scratch directory The tests will not begin unless the scratch directory is empty but will clean up this directory if tests complete successfully.

4.1.1 Detailed Description

Network File System (NFS) client.

Date

Sun Jul 7 21:03:06 2013

This library implements a wrapper around the NFS version 2 RPC specification. The application is provided with calls to mount a file system on a remote host and manipulate the files contained within.

The UDP protocol stack provided by the LWIP library is used for all network traffic. Reliable transport is assured through unique transaction IDs (XIDs) and selective retransmission. Transactions that are sent to the server are held in a local transaction list until a response is received. Periodic calls to nfs_timeout will trigger retransmissions as necessary.

This library requires that the server be hosting the UDP time protocol. The time of day is used to encourage the generation of unique transaction IDs. NFS and the associated services of mountd and portmapper must also be hosted by the server.

Besides the initialisation process, all communication is asynchronous. It is the combined responsibility of LWIP and the application to monitor and process network traffic whilst waiting for a response to an NFS transaction. When the response arrives, the registered callback for the transaction will be called to complete the transaction.

Data provided to the call back functions are available only during the execution of the call back function. It is the applications responsibility to copy this data to an alternate location if this data is required beyond the scope of the callback.

This NFS client library will authenticate using UNIX_AUTH credentials. All transactions are will originate from the "root" user. For this reason, it is recommended that the server does not export the file system with the "no_root_squash" flag. The "all_squash" flag should be used with anonuid and anongid optionally set as required.

4.1.2 Typedef Documentation

4.1.2.1 typedef void(* nfs_getattr_cb_t)(uintptr_t token, enum nfs_stat status, fattr_t *fattr)

A call back function provided by the caller of nfs_getattr, executed once a response is received.

Parameters

in	*token*	The unmodified token provided to the nfs_getattr call.
in	*status*	The NFS call status.
in	*fattr*	If status is NFS_OK, fattr will contain the attributes of the file in question. The contents of "fattr" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.

4.1.2.2 typedef void(* nfs_lookup_cb_t)(uintptr_t token, enum nfs_stat status, fhandle_t *fh, fattr_t *fattr)

A call back function provided by the caller of nfs_lookup, executed once a response is received.

Parameters

in	*token*	The unmodified token provided to the nfs_lookup call.
in	*status*	The NFS call status.
in	*fh*	If status is NFS_OK, fh will contain a handle to the file in question. The contents of "fh" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.
in	*fattr*	If status is NFS_OK, fattr will contain the attributes of the file in question. The contents of "fattr" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.

4.1.2.3 typedef void(* nfs_create_cb_t)(uintptr_t token, enum nfs_stat status, fhandle_t *fh, fattr_t *fattr)

A call back function provided by the caller of nfs_create, executed once a response is received.

Parameters

in	*token*	The unmodified token provided to the nfs_create call.
in	*status*	The NFS call status.
in	*fh*	If status is NFS_OK, fh will contain a handle to the file created. The contents of "fh" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.
in	*fattr*	If status is NFS_OK, fattr will contain the attributes of the file created. The contents of "fattr" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.

4.1.2.4 typedef void(* nfs_remove_cb_t)(uintptr_t token, enum nfs_stat status)

A call back function provided by the caller of nfs_remove, executed once a response is received.

Parameters

in	*token*	The unmodified token provided to the nfs_remove call.
in	*status*	The NFS call status.

4.1.2.5 typedef void(* nfs_readdir_cb_t)(uintptr_t token, enum nfs_stat status, int num_files, char *file_names[], nfscookie_t nfscookie)

A call back function provided by the caller of nfs_readdir, executed once a response is received.

Parameters

in	*token*	The unmodified token provided to the nfs_readdir call.
in	*status*	The NFS call status.
in	*num_files*	The number of file names read.
in	*file_names*	An array of NULL terminated file names that were read from the directory in question. The contents of "file_names" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.
in	*nfscookie*	A cookie to be used for subsequent calls to nfs_readdir in order to read the remaining file names from the directory The value of nfscookie will be given as 0 when there are no more file entries to read.

4.1.2.6 typedef void(* nfs_read_cb_t)(uintptr_t token, enum nfs_stat status, fattr_t *fattr, int count, void *data)

A call back function provided by the caller of nfs_read, executed once a response is received.

Parameters

in	*token*	The unmodified token provided to the nfs_read call.
in	*status*	The NFS call status.
in	*fattr*	If status is NFS_OK, fattr will contain the attributes. of the file that was read from. The contents of "fattr" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.
in	*count*	If status is NFS_OK, provides the number of bytes that were read from the file.
in	*data*	The memory address of the "count" bytes that were read from the file. The contents of "data" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.

4.1.2.7 typedef void(* nfs_write_cb_t)(uintptr_t token, enum nfs_stat status, fattr_t *fattr, int count)

A call back function provided by the caller of nfs_write, executed once a response is received.

Parameters

in	*token*	The unmodified token provided to the nfs_write call.
in	*status*	The NFS call status.
in	*fattr*	If status is NFS_OK, fattr will contain the new attributes of the file that was written. The contents of "fattr" will be invalid once this call back returns. It is the applications responsibility to copy this data to a more permanent location if it is required after this call back completes.
in	*count*	If status is NFS_OK, provides the number of bytes that were written to the file.

4.1.3 Enumeration Type Documentation

4.1.3.1 enum nfs_stat_t

The "nfs_stat" type is returned with every NFS procedure results.

A value of NFS_OK indicates that the call completed successfully and the results are valid. The other values indicate some kind of error occurred on the server side during the servicing of the procedure. The error values are derived from UNIX error numbers.

Enumerator:

NFS_OK The call completed successfully and the results are valid.

NFSERR_PERM Not owner. The caller does not have correct ownership to perform the requested operation.

NFSERR_NOENT No such file or directory. The file or directory specified does not exist.

NFSERR_IO Some sort of hard error occurred when the operation was in progress. This could be a disk error, for example.

NFSERR_NXIO No such device or address.

NFSERR_ACCES Permission denied. The caller does not have the correct permission to perform the requested operation.

NFSERR_EXIST File exists. The file specified already exists.

NFSERR_NODEV No such device.

NFSERR_NOTDIR Not a directory. The caller specified a non-directory in a directory operation.

NFSERR_ISDIR Is a directory. The caller specified a directory in a non-directory operation.

NFSERR_FBIG File too large. The operation caused a file to grow beyond the servers limit.

NFSERR_NOSPC No space left on device. The operation caused the servers file system to reach its limit.

NFSERR_ROFS Read-only file system. Write attempted on a read-only file system.

NFSERR_NAMETOOLONG File name too long. The file name in an operation was too long.

NFSERR_NOTEMPTY Directory not empty. Attempted to remove a directory that was not empty.

NFSERR_DQUOT Disk quota exceeded. The clients disk quota on the server has been exceeded.

NFSERR_STALE The "fhandle" given in the arguments was invalid. That is, the file referred to by that file handle no longer exists, or access to it has been revoked.

NFSERR_WFLUSH The servers write cache used in the "WRITECACHE" call got flushed to disk.

NFSERR_COMM A communication error occurred at the RPC layer.

4.1.3.2 enum rpc_stat_t

The "rpc_stat" type is returned when an asynchronous response is expected.

A value of RPC_OK indicated that the transaction was successfully sent. Note that this does not always mean that the transaction was successfully delivered.

Enumerator:

RPC_OK The call completed successfully.

RPCERR_NOMEM Out of memory.

RPCERR_NOBUF No network buffers available for communication.

RPCERR_COMM Communication error in send phase.

RPCERR_NOSUP The host rejected the request.

4.1.3.3 enum ftype_t

The enumeration "ftype" gives the type of a file.

Enumerator:

NFREG indicates a non-file.

NFDIR a regular file.

NFBLK a directory.

NFCHR a block-special device.

NFLNK a character-special device.

4.1.4 Function Documentation

4.1.4.1 enum rpc_stat nfs_init (const struct ip_addr * *server*)

Initialises the NFS subsystem.

This function should be called once at startup with the address of your NFS server. The UDP time protocol will be used to obtain a seed for transaction ID numbers.

Parameters

in	*server*	The IP address of the NFS server that we should connect to

Returns

RPC_OK if the NFS subsystem was successfully initialised. Otherwise an appropriate error code will be returned.

4.1.4.2 void nfs_timeout (void)

Handles packet loss and retransmission.

Since this NFS library runs over the unreliable UDP protocol, it is possible that packets may be dropped. To allow NFS to retransmit packets that might have been dropped you must arrange for nfs_timeout to be called every 100ms. This could be achieved by using a timer.

4.1.4.3 enum rpc_stat nfs_mount (const char * *dir,* **fhandle_t** * *pfh*)

A synchronous function used to mount a file system over the network.

This function will mount a file system and return a cookie to it in pfh. The returned cookie should be used on subsequent NFS transactions.

Parameters

in	*dir*	The path that NFS should mount.
out	*pfh*	The returned file handle if the call was successful.

Returns

RPC_OK if the call was successful and pfh was updated. Otherwise, an appropriate error code will be returned.

4.1.4.4 enum rpc_stat nfs_print_exports (void)

Synchronous function used to print the directories exported by the server.

This function is primarily used for the purpose of debugging.

Returns

RPC_OK if the call was successful and the export list was printed. Otherwise, an appropriate error code is returned.

4.1.4.5 enum rpc_stat nfs_getattr (const fhandle_t * *fh*, nfs_getattr_cb_t *callback*, uintptr_t *token*)

An asynchronous function used for retrieving the attributes of a file.

This function is the equivalent of the UNIX "stat" function. It will find the current attributes on a given file handle. The attributes are passed back through the provided callback function (nfs_getattr_cb_t) with the provided token passed, unmodified, as an argument.

Parameters

in	*fh*	An NFS handle to the file in question.
in	*callback*	An nfs_getattr_cb_t callback function to call once a response arrives.
in	*token*	A token to pass, unmodified, to the callback function.

Returns

RPC_OK if the request was successfully sent. Otherwise an appropriate error code will be returned. "callback" will be called once the response to this request has been received.

4.1.4.6 enum rpc_stat nfs_lookup (const fhandle_t * *pfh*, const char * *name*, nfs_lookup_cb_t *callback*, uintptr_t *token*)

Asynchronous function used for retrieving an NFS file handle (fhandle_t) of a file located on the server.

Before you are able to complete any operation on a file you must obtain a handle to it. This function will find a file named "name" in the specified directory. The directory is given in the form of a handle which may have been provided by nfs_mount. When the transaction has completed, the provided callback (nfs_lookup_cb_t) will be executed with the provided token passed, unmodified, as an argument. as an argument

Parameters

in	*pfh*	An NFS file handle (fhandle_t) to the directory that contains the requested file.
in	*name*	The NULL terminated file name to look up.
in	*callback*	An nfs_lookup_cb_t callback function to call once a response arrives.
in	*token*	A token to pass, unmodified, to the callback function.

Returns

RPC_OK if the request was successfully sent. Otherwise an appropriate error code will be returned. "callback" will be called once the response to this request has been received.

4.1.4.7 enum rpc_stat nfs_create (const fhandle_t * *pfh*, const char * *name*, const sattr_t * *sattr*, nfs_create_cb_t *callback*, uintptr_t *token*)

An asynchronous function used for creating a new file on the NFS file server.

This function is used to create a new file named "name" with the attributes "sattr". On completion, the provided callback (nfs_create_cb_t) will be executed with "token" passed, unmodified, as an argument.

Parameters

in	*pfh*	An NFS file handle (fhandle_t) to the directory that should contain the newly created file.
in	*name*	The NULL terminated name of the file to create.
in	*sattr*	The attributes which the file should posses after creation.
in	*callback*	An nfs_create_cb_t callback function to call once a response arrives.
in	*token*	A token to pass, unmodified, to the callback function.

Returns

RPC_OK if the request was successfully sent. Otherwise an appropriate error code will be returned. "callback" will be called once the response to this request has been received.

4.1.4.8 enum rpc_stat nfs_remove (const fhandle_t * *pfh*, const char * *name*, nfs_remove_cb_t *callback*, uintptr_t *token*)

An asynchronous function used for removing an existing file from the NFS file server.

This function will remove a file named "name" from the provided directory. The directory takes the form of a handle that may be acquired through nfs_mount or nfs_lookup. When the transaction has completed, the provided callback function (nfs_remove_cb_t) will be executed with "token" passed, unmodified, as an argument.

Parameters

in	*pfh*	An NFS file handle (fhandle_t) to the directory that contains the file to remove.
in	*name*	The name of the file to remove.
in	*callback*	An nfs_remove_cb_t callback function to call once a response arrives.
in	*token*	A token to pass, unmodified, to the callback function.

Returns

RPC_OK if the request was successfully sent. Otherwise an appropriate error code will be returned. "callback" will be called once the response to this request has been received.

4.1.4.9 enum rpc_stat nfs_readdir (const fhandle_t * *pfh*, nfscookie_t *cookie*, nfs_readdir_cb_t *callback*, uintptr_t *token*)

An asynchronous function used for reading the names of the files that are stored within the given directory.

This function reads the contents, that is the filenames, from the directory provided by "pfh". When the transaction is complete, the provided callback function (nfs_readdir_cb_t) will be executed with the unmodified "token" passed as an argument. The number of file names that this transaction can return is of course limited by the Maximum Transmission Unit (MTU) of the network link. To compensate for this limitation, a "cookie" is passed as an argument to the transaction. The cookie should be initially be provided with the value zero. The callback function will be provided with a cookie value to use if subsequent calls are required.

Parameters

in	*pfh*	An NFS handle to the directory to read.
in	*cookie*	An NFS cookie to be used in the case of a continuation. When this is the first call to this function, "cookie" should be provided as 0. Otherwise, the submitted value should be the value that was returned from the previous call.
in	*callback*	An nfs_readdir_cb_t callback function to call once a response arrives.
in	*token*	A token to pass, unmodified, to the callback function.

Returns

RPC_OK if the request was successfully sent. Otherwise an appropriate error code will be returned. "callback" will be called once the response to this request has been received.

4.1.4.10 enum rpc_stat nfs_read (const fhandle_t * *fh*, int *offset*, int *count*, nfs_read_cb_t *callback*, uintptr_t *token*)

An asynchronous function used for reading data from a file.

nfs_read will start at "offset" bytes within the file provided as "fh" and read a maximum of "count" bytes of data. When the transaction has completed, the provided callback function (nfs_read_cb_t) will be called with "token" passed, unmodified, as an argument. The file data and the actual number of bytes read is passed to the callback

but the data will only be available for the scope of the callback. It is the applications responsibility to ensure that any data that is required outside of this scope is moved to a more permanent location before returning from the callback.

Parameters

in	*fh*	An NFS file handle (fhandle_t) to the file which should be read from.
in	*offset*	The position, in bytes, at which to begin reading data.
in	*count*	The number of bytes to read from the file.
in	*callback*	An nfs_read_cb_t callback function to call once a response arrives.
in	*token*	A token to pass, unmodified, to the callback function.

Returns

RPC_OK if the request was successfully sent. Otherwise an appropriate error code will be returned. "callback" will be called once the response to this request has been received.

4.1.4.11 enum rpc_stat nfs_write (const fhandle_t * *fh*, int *offset*, int *count*, const void * *data*, nfs_write_cb_t *callback*, uintptr_t *token*)

Asynchronous function used for writing data to a file.

nfs_write will start at "offset" bytes within the file provided as "fh" and write a maximum of "count" bytes of the provided "data". When the transaction has completed, the provided callback function (nfs_write_cb_t) will be called with "token" passed, unmodified, as an argument. The callback will also be provided with the actual number of bytes written.

Parameters

in	*fh*	An NFS file handle (fhandle_t) to the file which should be written to.
in	*offset*	The position, in bytes, at which to begin writing data.
in	*count*	The number of bytes to write to the file.
in	*data*	The start address of the data that is to be written.
in	*callback*	An nfs_write_cb_t callback function to call once a response arrives.
in	*token*	A token to pass, unmodified, to the callback function.

Returns

RPC_OK if the request was successfully sent. Otherwise an appropriate error code will be returned. "callback" will be called once the response to this request has been received.

4.1.4.12 int nfs_test (char * *mnt*)

Tests the NFS system using the provided path as a scratch directory The tests will not begin unless the scratch directory is empty but will clean up this directory if tests complete successfully.

Parameters

mnt	The mount point to use when generating test files.

Returns

0 if all tests completed successfully. Otherwise, returns the number of errors recorded.

Index