The VFS-FS protocol

This page provides the official documentation of the VFS-FS protocol of MINIX 3. It describes the protocol used between the VFS server and file servers responsible for mounted file systems. The current version documents the protocol used in Git commit **dca81e05**. If you update this document because of changes to MINIX 3, please mention the revision number of the change in the wiki comment.

Sommaire

General information

The following information is written mainly for people implementing a file server.

Request and replies

All requests made in the VFS-FS protocol come from VFS; the file server only sends replies. Every request must be answered with exactly one reply. The first

request from VFS is REQ_READSUPER; the last request is REQ_UNMOUNT. The file server must use sef_receive() or sef_receive_status() to receive messages, and should use send() to send replies.

The `<minix/vfsif.h>` header file provides the definitions for requests, replies and message fields. VFS puts the request code in the type field of the request message. Request codes are always in the range VFS_BASE to VFS_BASE+NREQS, exclusive. The file server puts reply code for a request in the type field of the reply message. This is always OK (0) or a negative error code. Any request not implemented by the file server must return EINVAL or, preferably, ENOSYS.

It is possible for the file server to send a request to VFS in response to a request from VFS. Such requests always have a positive nonzero request code, which allows them to be distinguished from replies. VFS will process requests from a FS in reply to its own request, repeatedly, until it receives an actual reply. Support for such nested calls is limited and may break arbitrarily. Currently, VFS allows only the getsysinfo() system call to be invoked this way, for use by ProcFS.

System interaction

The driver should use the System Event Framework (SEF). This framework automatically takes care of interaction with the Reincarnation Server (RS).

REQ_UNMOUNT is not a request for termination. Once the file system has been unmounted, the file server should terminate upon receipt of a SIGTERM signal. The file server must not terminate from a SIGTERM as long as the file system is still mounted (this situation can occur upon system shutdown), but should then terminate after processing a subsequent unmount request. In other words, the file server should terminate after it received a SIGTERM signal *and* it has been unmounted. The file system may, but need not support receipt of another REQ_READSUPER request after being unmounted.

Signals may be received by registering a signal handler callback function using the SEF API. Aside from the messages recognized by SEF, the file server will in principle not receive any messages besides requests from VFS.

General behavior

If a file server returns an error in response to a request, that request is expected not to have had any effect: the state of the file server is exactly the same before and after the request, and no "progress" has been made in any sense.

A notable exception to these are the EENTERMOUNT, ELEAVEMOUNT and ESYMLINK pseudo-errors returned by REQ_LOOKUP. These errors are interpreted by VFS as part of the path lookup process, and are never passed back to the application. No other requests must return these errors.

Inodes

VFS requests mainly revolve around inodes. In the VFS-FS protocol, inodes represent open files (of any type, including directories) in the system. Many requests operate on an inode, or on a directory entry (see below). An inode is uniquely identified by its inode number; the file server is responsible for picking inode numbers for its files. Inode number 0 is reserved and may mean "no inode"

in some contexts.

Upon success, certain requests (REQ_READSUPER, REQ_LOOKUP, REQ_CREATE, REQ_NEWNODE) open an inode. Each open request for an inode increases that inode's reference count: the number of times the inode is opened by VFS. The REQ_PUTNODE request decreases an open inode's reference count, effectively closing the inode when its reference count has reached zero again. A file server should only treat the reference count as an indicator whether the inode is open or not. The actual reference count is meaningless, as VFS may delay REQ_PUTNODE requests for as long as the file remains open, or drop all but one references at any time.

VFS caches open inodes including some metadata, in the form of vnodes. Cached metadata includes the file's mode, size, owner UID and GID, and special device number. VFS only refers to open inodes in requests. That is, if an inode number is passed as part of a VFS request, the associated inode will have a nonzero reference count in the file server.

Grants

Data transferred between VFS (or the user process) and the FS that is not put in the request/reply messages, is passed using memory grants. An FS can use sys_safecopyfrom (for READ grants) or sys_safecopyto (for WRITE grants) to retrieve or store data. These functions may not succeed; any errors thrown by these functions must be passed back to VFS.

For grants used to pass strings from VFS to the file server, the provided grant and string length will include the terminating '\0' character. A notable exception is REQ_SLINK, which provides an unterminated string for the link target; similarly, REQ_RDLINK is expected to return an unterminated string. In all other cases, file servers may check for a positive nonzero length and the presence of the terminating '\0' character as part of protection against VFS bugs.

Directory entries

Several calls manipulate directory entries. A directory entry is always provided in the form of a containing directory inode number, which is guaranteed to be an inode for a directory by VFS, and a last path component, which is a name string passed via a grant (usually in the REQ_GRANT field of the request message) and a string length (usually in the REQ_PATH_LENGTH field).

Such a name string will never contain a '/' slash character, or contain a '\0' character anywhere else but at the end. As indicated above, it is guaranteed to end with a '\0' character, and the passed string length will include that character. The maximum string length, including terminating '\0' character, is limited to PATH_MAX+1 bytes. A file server is free not to accept long names by returning an ENAMETOOLONG error.

VFS generally does not perform any other checks on the last path component of such directory entries, leaving those up to each file server. For example, when VFS requests the creation of a new directory entry, a file server will have to make sure that an entry with that name does not already exist in that directory.

Protection

VFS takes care of all checks to make sure that no write requests are sent to a read-only file system. In principle, the file system should never have to return an EROFS error. A file server may choose to perform this check itself anyway, to protect against VFS bugs.

VFS performs all access permission checks, with one exception. During lookups, when an inode is used as a directory in the process of resolving a path, the file server needs to check whether the caller has search access for that directory. A file server may choose to perform other checks itself as well to protect against VFS bugs, but may not always be able to, since not all requests provide the caller's user and group ID.

Device drivers

Every file server is responsible for raw transfers from (REQ_BREAD) and to (REQ_BWRITE) the device that it has been mounted on. This allows it to keep its cache synchronized. Before such raw transfers are initiated, VFS will send a REQ_NEW_DRIVER request to the file server to indicate which driver label to use. The root file server has more responsibilities as far as this is concerned; see the next section.

MINIX 3 supports restarting crashed device drivers. This includes the block device drivers that file systems are mounted on. File servers may assume a crash-only model: a block device driver either returns a valid response to a request, or communication results in an error. This may be an IPC-level error or an ERESTART response code. IPC-level errors may be transient, and the file server should retry the original request at least once to see if another IPC-level error is generated. If so, the FS may assume that the device driver has become fully unavailable. An ERESTART response code indicates the driver has been restarted and the file server should reissue a BDEV_OPEN to the driver before retrying the request. Other non-IPC error codes may be transient. For robustness, the FS may choose to retry an operation a few times before propagating up the error. If at all possible, the file system should use the *libbdev* library to talk to block device drivers, which transparently takes care of all these issues.

Root file server

The root file server (serving the "/" partition) has a number of additional responsibilities that other file servers do not have. In particular:

- It is responsible for all block devices that have not been mounted.
  - It is currently the only file server that has to support REQ_FLUSH, to flush its cache for certain unmounted devices.
  - It is also the only file server that will receive REQ_NEW_DRIVER requests for devices other than the one it is mounted on.

Pipes and sockets

While file systems may support files of type named pipe (S_IFIFO) and of type socket (S_IFSOCK), they need not implement support for pipe communication nor unix domain socket communication. This is all handled by the Pipe File Server (PFS). PFS is also responsible for cloned devices and is consequently the only file server that has to support REQ_NEWNODE, to create deleted open files.

Protocol messages

This specification reflects the protocol as it should be implemented, not how it is implemented by MFS. In particular, old and deprecated requests are not and should not be included.

The entire VFS-FS protocol is entirely POSIX-oriented. Any deviation from the requirements imposed by POSIX in this specification is unintentional except when mentioned explicitly. For convenience, links to the relevant Open Group function specifications and file access (ATIME), modification (MTIME) and change (CTIME) time-stamp update requirements are provided.

The reply codes in this document are advisory and mostly aimed at indicating additional restrictions needed for POSIX compliance. Not all of them may be applicable to every file server, and a file server may send other error codes where appropriate. Errors resulting from protocol validation checks (e.g. EROFS), and sys_safecopy.. errors, are not included.

The requests are ordered according to the following rough categorization:

- Mounting and unmounting
  - REQ_READSUPER
  - REQ_UNMOUNT
- Inode open and close functions
  - REQ_LOOKUP
  - REQ_CREATE
  - REQ_NEWNODE
  - REQ_PUTNODE
- Inode use functions
  - REQ_READ
  - REQ_WRITE
  - REQ_PEEK
  - REQ_BPEEK
  - REQ_GETDENTS
  - REQ_INHIBREAD
  - REQ_FTRUNC
- Inode metadata retrieval and manipulation
  - REQ_STAT
  - REQ_CHOWN
  - REQ_CHMOD
  - REQ_UTIME
- Directory entry manipulation
  - REQ_CREATE (see above)
  - REQ_MKDIR
  - REQ_MKNOD
  - REQ_LINK
  - REQ_UNLINK

REQ_RMDIR

REQ_RENAME

REQ_SLINK

REQ_RDLINK

- Miscellaneous file system operations

  REQ_MOUNTPOINT

  REQ_FSTATFS

  REQ_STATVFS

  REQ_SYNC

- Block I/O functions

  REQ_FLUSH

  REQ_NEW_DRIVER

  REQ_BREAD

  REQ_BWRITE

Mounting and unmounting

REQ_READSUPER

Mount the file system.

Request fields

REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for the label of the block device driver to use
REQ_PATH_LEN	m9_s2	unsigned short	length of the label
REQ_DEV	m9_l5	dev_t	device number of block device to mount
REQ_FLAGS	m9_s3	int	flag field containing a bitwise combination of the following possible flags: REQ_RDONLY (the file system is mounted read-only), REQ_ISROOT (the file system is the root file system)

Reply fields

RES_INODE_NR	m9_l1	ino_t	*upon success:* inode number of the root inode
RES_MODE	m9_s2	mode_t	*upon success:* mode of the root inode
RES_FILE_SIZE_HI	m9_l2	u32_t	*upon success:* file size of the root inode (upper 32 bits)
RES_FILE_SIZE_LO	m9_l3	u32_t	*upon success:* file size of the root inode (lower 32 bits)

RES_UID	m9_s4	uid_t	*upon success:* user ID of the root inode
RES_GID	m9_s1	gid_t	*upon success:* group ID of the root inode
RES_CONREQS	m9_s3	u16_t	*upon success:* number of concurrent requests that can be handled by FS

Reply codes

EINVAL	label too long
EINVAL	unable to retrieve endpoint from DS using label
EINVAL	opening device driver failed
EINVAL	reading superblock failed
OK	file system initialized and mounted

Description

This is the first request sent by VFS to any file server. Upon receiving this request, the file server is expected to initialize itself. This typically includes opening the given device and reading the file system's superblock from it; the file server can first obtain the driver endpoint by querying DS with the label name. If the call succeeds, the file server is expected to open the root inode (setting its reference count to 1), and return its details to VFS.

Notes

- The label is null-terminated, and the given length includes the terminating '\0' character. The maximum label size is always at most DS_MAX_KEYLEN, but VFS currently uses a hardcoded size of 16 (including '\0').

REQ_UNMOUNT

Unmount the file system.

Request fields

none

Reply fields

none

Reply codes

OK	file system unmounted

Description

This is the last request sent by VFS to any file server. It indicates that the

file system is to be unmounted, and all of its resources to be cleaned up. This typically includes writing out any dirty data and closing the associated device. Analogous to how REQ_READSUPER opens the root inode, REQ_UNMOUNT must close the root inode by decreasing its reference count with 1. VFS assumes that this function succeeds, and will ignore returned error codes.

Notes

- After this request is finished, all inodes will have a reference count of zero.

Inode open and close functions

REQ_LOOKUP

Resolve a path string to an inode.

Request fields

REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ|WRITE) of the buffer containing the pathname
REQ_PATH_LEN	m9_s2	int	length of the remaining part of the string to resolve
REQ_PATH_SIZE	m9_l5	size_t	total size of the buffer
REQ_DIR_INO	m9_l3	ino_t	inode number of the starting directory
REQ_ROOT_INO	m9_l4	ino_t	inode number of the root directory of the caller, or 0 if not on this file system
REQ_FLAGS	m9_s3	int	flag field containing a bitwise combination of the following possible flags: PATH_RET_SYMLINK (do not resolve a symlink as the last path component), PATH_GET_UCRED (copy credentials from VFS instead of using REQ_UID and REQ_GID)
REQ_UID	m9_s4	uid_t	user ID of the caller
REQ_GID	m9_s1	gid_t	group ID of the caller
REQ_GRANT2	m9_l1	cp_grant_id_t	memory grant (READ) of the *vfs_ucred_t* structure containing supplemental group data
REQ_UCRED_SIZE	m9_s4	size_t	total size of *vfs_ucred_t* structure

Reply fields

RES_INODE_NR	m9_l1	ino_t	*upon success:* resulting file inode number
RES_MODE	m9_s2	mode_t	*upon success:* resulting file mode
RES_FILE_SIZE_HI	m9_l2	u32_t	*upon success:* resulting file size (upper 32 bits)
RES_FILE_SIZE_LO	m9_l3	u32_t	*upon success:* resulting file size (lower 32 bits)
RES_DEV	m9_l4	dev_t	*upon success:* resulting file device number
RES_UID	m9_s4	uid_t	*upon success:* resulting file user ID
RES_GID	m9_s1	gid_t	*upon success:* resulting file group ID
RES_INODE_NR	m9_l1	ino_t	*upon EENTERMOUNT:* inode number of the mountpoint inode
RES_OFFSET	m9_s2	int	*upon EENTERMOUNT and ELEAVEMOUNT and ESYMLINK:* new starting offset of string within buffer
RES_SYMLOOP	m9_s3	unsigned short	*upon EENTERMOUNT and ELEAVEMOUNT and ESYMLINK:* number of symbolic links followed

Reply codes

ENAMETOOLONG	provided path length exceeds what file server can handle
ENAMETOOLONG	any of the path components is longer than the file system supports
ENOTDIR	any of the intermediate path components is not a directory
EACCES	the caller has no search access permission on any of the intermediate directories
ENFILE	no inodes are available in memory
ELOOP	more than SYMLOOP_MAX symlinks were encountered during the lookup
ENAMETOOLONG	resulting path to copy back (including terminating '\0') does not fit in provided buffer
EENTERMOUNT	a mountpoint was encountered
ELEAVEMOUNT	".." is followed from the file system root and the file system root is not the caller root inode
ESYMLINK	an absolute symlink was encountered
OK	inode successfully looked up and opened

Description

This request resolves a path string to an inode. The path is

null-terminated; the file server may use REQ_PATH_LEN to determine how many bytes it needs to copy in before starting resolution. This length includes the terminating '\0' character.

The path resolution starts at the inode given by REQ_DIR_INO. This is always a directory.

In the path, path components are separated by '/' slash characters. Multiple consecutive slash characters are to be treated like one single slash character. A single "." dot path component refers to the current directory resolved so far; two ".." dots refer to the parent directory of the directory resolved so far. Path names ending in a slash have an implied single dot as last path component: they refer to the directory resolved so far. If the given path is empty, the path resolves to the starting inode. A slash at the beginning of the path name has no special meaning, but if the entire path consists of only slashes, this case should be treated as being followed by a dot rather than being empty.

Whenever an inode is used as a directory in a lookup, the file server must check that the caller has search access permissions on that directory, using the directory's mode and the given caller credentials - see below. An inode is used as a directory when it is not the last path component. As indicated, a path component followed by a slash is never the last path component. If an empty path was given, the starting inode is not considered to be accessed as a directory. One exception to the access permission check requirements relates to leaving mountpoints; see below.

If the given root inode (REQ_ROOT_INO) is nonzero, this inode is a directory, and the file server must never allow a path lookup to go below this inode. A lookup on ".." from the root inode uses that root inode as a directory but should resolve to that root inode itself, rather than its parent. If the given root inode is zero, the caller's root directory is on another file system and need not be considered.

Lookups may cross mountpoints. If a lookup on ".." is performed from the file system root (and that inode is not also the caller's root directory), the file server must return the ELEAVEMOUNT pseudo-error to VFS, along with an offset pointing to the first character of the ".." path component or any slash character directly preceding it. If a lookup encounters an inode previously marked as mountpoint (using REQ_MOUNTPOINT), the file server must return the EENTERMOUNT pseudo-error to VFS, along with the inode number of the mountpoint and the offset pointing to the first character after the path component identifying the mountpoint (which is typically either a '/' slash character or the terminating '\0' character). It is possible that an inode marked as a mountpoint is encountered even when going up the tree - in this case, too, the inode should indeed be treated as a mountpoint and generate an EENTERMOUNT error.

The main exception to the behavior with respect to mountpoints, is if the given starting inode is a mountpoint itself. In this case, the first path component after the mountpoint is guaranteed by VFS to be "..". When leaving a mountpoint this way, the file server must skip the search access permissions check on the mountpoint directory, even though it is used as directory. In essence, the properties of the mountpoint directory are entirely hidden.

REQ_FLAGS is a bitwise OR'ed mask of flags. If it has the PATH_RET_SYMLINK flag set, and the last path component of the path is a

symbolic link, then the result of the lookup must be that symbolic link inode; otherwise, the lookup must continue with the contents of the symbolic link. All other path components that resolve to symbolic links must be followed, up to SYMLOOP_MAX times during the lookup. The count of symbolic link resolutions must be returned if the lookup results in EENTERMOUNT, ELEAVEMOUNT and ESYMLINK. If a symbolic link is resolved during the lookup process, then the symbolic link followed by any remaining part of the original path must be used as the new remaining path. If the symbolic link is relative (i.e. it does not start with a '/' slash), the file server is expected to continue resolving the path. If the symbolic link is absolute (i.e. it starts with a slash), then the file server must return the ESYMLINK pseudo-error to VFS. In general, when the file server resolves a symbolic link and then proceeds to return an EENTERMOUNT, ELEAVEMOUNT or ESYMLINK pseudo-error, it must copy back the remaining part of the changed path back into the given granted buffer. The file server must null-terminate the path, and be careful not to exceed the total REQ_PATH_SIZE size.

The caller credentials that are used for directory search access checks, may be passed in either of two ways. If the PATH_GET_UCRED flag is not set in the REQ_FLAGS field, then REQ_UID and REQ_GID are to be used as the caller's user and group ID, respectively. If PATH_GET_UCRED is set, the file system must instead copy in a *vfs_ucred_t* structure from VFS, from the grant identified by REQ_GRANT2. This supplies the caller's user ID, group ID, and a set of supplemental group IDs. The caller is considered to be part of all these groups. The REQ_UCRED_SIZE field should be used to make sure that VFS and the file server agree on the size of the *vfs_ucred_t* structure. The file server may choose to copy in only the given number of bytes if the size is smaller than it expects. A larger size should be considered to be an error.

Upon success, the resulting inode is opened (increasing its reference count with 1) and its details are returned to VFS. Upon entering a mountpoint (EENTERMOUNT), the inode number of the mountpoint is returned but its reference count remains unchanged.

Notes

- If the given root directory inode is nonzero, a file server may choose to continue resolving absolute symlinks itself.
- See this section for a possible approach to implementing most of the requirements above.

REQ_CREATE

Create a regular file.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number of the containing directory for the new file
REQ_MODE	m9_s3	mode_t	mode for the file
REQ_UID	m9_s4	uid_t	user ID for the file

REQ_GID	m9_s1	gid_t	group ID for the file
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for the last path component
REQ_PATH_LEN	m9_s2	unsigned short	length of the last path component

Reply fields

RES_INODE_NR	m9_l1	ino_t	*upon success:* inode number of created file
RES_MODE	m9_s2	mode_t	*upon success:* mode of created file
RES_FILE_SIZE_HI	m9_l2	u32_t	*upon success:* file size of created file (upper 32 bits)
RES_FILE_SIZE_LO	m9_l3	u32_t	*upon success:* file size of created file (lower 32 bits)
RES_UID	m9_s4	uid_t	*upon success:* user ID of created file
RES_GID	m9_s1	gid_t	*upon success:* group ID of created file

Reply codes

ENAMETOOLONG	the last path component is longer than the file system supports
EEXIST	an entry with that name already exists in the given directory
ENFILE	no inodes are available
ENOSPC	no space is left on the device
EFBIG	the containing directory can not handle any more entries
ENOENT	the containing directory has been removed
OK	regular file created and opened

Description

This request creates a new regular file with the given properties, opens it (setting its reference count to 1), and returns its actual details to VFS.

Notes

- The given mode includes the S_IFREG type.
- The resulting file size will typically be zero.
- This is a write request, and will not be sent to read-only file systems.
- Open Group links: open, creat
- POSIX notes: upon success, the new file's ATIME, CTIME and MTIME, and the containing directory's CTIME and MTIME are updated.

REQ_NEWNODE

Create an open, unlinked file.

Request fields

REQ_MODE	m9_s3	mode_t	mode for the inode
REQ_DEV	m9_l5	dev_t	device number for the inode
REQ_UID	m9_s4	uid_t	user ID for the inode
REQ_GID	m9_s1	gid_t	group ID for the inode

Reply fields

RES_INODE_NR	m9_l1	ino_t	*upon success:* inode number of the resulting inode
RES_MODE	m9_s2	mode_t	*upon success:* mode of the resulting inode
RES_FILE_SIZE_HI	m9_l2	u32_t	*upon success:* size of the resulting inode (upper 32 bits)
RES_FILE_SIZE_LO	m9_l3	u32_t	*upon success:* size of the resulting inode (lower 32 bits)
RES_DEV	m9_l4	dev_t	*upon success:* device number of the resulting inode
RES_UID	m9_s4	uid_t	*upon success:* user ID of the resulting inode
RES_GID	m9_s1	gid_t	*upon success:* group ID of the resulting inode

Reply codes

ENFILE	no inodes are available
OK	temporary inode created and opened

Description

This request creates an open but unlinked file. If successful, an inode with a reference count of 1 is returned; this inode exists until its reference count is decreased to zero again. Currently, only the pipe file server will get REQ_NEWNODE requests.

Notes

- The given device number is only relevant for block-special and character-special files.
- The resulting file size will typically be zero.
- Only PipeFS will get this request.
- This is a write request, and would in principle not be sent to read-only file systems.

- POSIX notes: upon success, the resulting file's ATIME, MTIME and CTIME are updated.

REQ_PUTNODE

Decrease an open file's reference count.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_COUNT	m9_l2	ino_t	number of references to drop

Reply fields

none

Reply codes

OK	reference count decreased

Description

This request tells the file server to decrease the reference count of an inode, closing the file if the reference count reaches zero. VFS assumes that this function succeeds, and will ignore returned error codes.

Notes

- VFS will always decrease the count by at least one, and at most the total count.
- VFS may delay putnode requests for as long as the file remains open.

Inode use functions

REQ_READ

Read from a file.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (WRITE) to store the resulting data in
REQ_SEEK_POS_HI	m9_l3	u32_t	seek position into the open file (upper 32 bits)
REQ_SEEK_POS_LO	m9_l4	u32_t	seek position into the open file (lower 32 bits)
REQ_NBYTES	m9_l5	size_t	number of bytes to read

Reply fields

RES_SEEK_POS_HI	m9_l3	u32_t	*upon success:* resulting file position (upper 32 bits)
RES_SEEK_POS_LO	m9_l4	u32_t	*upon success:* resulting file position (lower 32 bits)
RES_NBYTES	m9_l5	size_t	*upon success:* number of bytes read

Reply codes

OK	results successfully (partially) read, or EOF reached

Description

This call should store as many bytes as can fit and do not exceed the end of file into the given buffer, starting from the given seek position. The seek position should be advanced accordingly. If the given position points at or beyond the end of file, zero bytes (EOF) should be returned successfully, and the position must not be advanced further.

Notes

- This call is not used for block or character special files.
- The number of bytes to read is always positive and nonzero.
- Open Group link: read
- POSIX notes: upon success, file's ATIME is updated.

REQ_WRITE

Write to a file.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) containing the data to write
REQ_SEEK_POS_HI	m9_l3	u32_t	seek position into the open file (upper 32 bits)
REQ_SEEK_POS_LO	m9_l4	u32_t	seek position into the open file (lower 32 bits)
REQ_NBYTES	m9_l5	size_t	number of bytes to write

Reply fields

RES_SEEK_POS_HI	m9_l3	u32_t	*upon success:* resulting file position (upper 32 bits)
RES_SEEK_POS_LO	m9_l4	u32_t	*upon success:* resulting file position (lower 32 bits)

RES_NBYTES	m9_l5	size_t	*upon success:* number of bytes written

Reply codes

ENOSPC	no space is left on the device
EFBIG	the write would make the resulting file size too big
OK	results successfully written

Description

This call call should write all bytes to the given seek position of the file, possibly extending the file size. If the position is already beyond the end of the file, the gap between the current end of file and the new data should be filled with zeroes (using file holes if supported).

Notes

- This call is not used for block or character special files.
- The number of bytes to write is always positive and nonzero.
- This is a write request, and will not be sent to read-only file systems.
- Open Group link: write
- POSIX notes: upon success, the file's CTIME and MTIME are updated.

REQ_PEEK

Request filesystem to retrieve the requested inode data.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_SEEK_POS_HI	m9_l3	u32_t	seek position into the open file (upper 32 bits)
REQ_SEEK_POS_LO	m9_l4	u32_t	seek position into the open file (lower 32 bits)
REQ_NBYTES	m9_l5	size_t	number of bytes to read

Reply fields

RES_NBYTES	m9_l5	size_t	*upon success:* number of bytes were successfully put in the cache

Reply codes

OK	peek done, see REQ_NBYTES reply to see how many were available, filesystem cache metadata
ENOSYS	filesystem does not implement REQ_PEEK and mmap() call of files opened on this FS should fail

Description

This call operates just like REQ_READ, but the results are not copied anywhere except for to the in-filesystem cache. It is to support a to-be-merged mmap implementation in which filesystem cache blocks are annotated with inode number/offset information that VM can then use to share pages of with processes.

The (future) mmap implementation makes one important assumption based on the result of REQ_PEEK. If it returns ENOSYS, mmap() can not work as relies on REQ_PEEK to make the blocks visible to VM. mmap() will fail gracefully when it is called on FD's that represent open inodes on this FS. More significantly, if REQ_PEEK does not return ENOSYS, it is assumed that REQ_PEEK also informs the cache code of inode metadata.

Therefore, filesystems must do both (implement REQ_PEEK and inform the cache code of inode metadata), or make REQ_PEEK always return ENOSYS. Otherwise the mmap() may succeed but page accesses will fail, causing disasters in general and most of all when exec() uses mmap().

REQ_BPEEK

Request filesystem to retrieve the requested block data.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_SEEK_POS_HI	m9_l3	u32_t	seek position into the block device (upper 32 bits)
REQ_SEEK_POS_LO	m9_l4	u32_t	seek position into the block device (lower 32 bits)
REQ_NBYTES	m9_l5	size_t	number of bytes to read

Reply codes

OK	peek done. all requested blocks have been in the cache.
ENOSYS	filesystem does not implement REQ_BPEEK and mmap() call of files opened on this FS should fail

Description

This call operates just like REQ_PEEK, but the request (offset and size) are about the block device instead of any particular inode.

for filesystems using libminixfs, lmfs_do_bpeek() can implement this request for filesystems if desired.

REQ_GETDENTS

Retrieve directory entries.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number of the directory
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (WRITE) to store resulting struct dirent entries and names in
REQ_MEM_SIZE	m9_l5	size_t	size of given memory grant
REQ_SEEK_POS_HI	m9_l3	u32_t	seek position into the open file (upper 32 bits)
REQ_SEEK_POS_LO	m9_l4	u32_t	seek position into the open file (lower 32 bits)

Reply fields

RES_SEEK_POS_HI	m9_l3	u32_t	*upon success:* new seek position into the file (upper 32 bits)
RES_SEEK_POS_LO	m9_l4	u32_t	*upon success:* new seek position into the file (lower 32 bits)
RES_NBYTES	m9_l5	size_t	*upon success:* the amount of resulting bytes stored, with 0 for EOF

Reply codes

ENOENT	the given file position is not aligned to the internal data structures (file system specific)
EINVAL	the given buffer is too small to store even one entry (including padding)
OK	stored zero or more entries in the user's buffer

Description

This call should store as many entries as can fit and are still present into the given buffer, starting from the entry identified by the given seek position. The seek position should be advanced accordingly. If the given position points beyond the last entry, zero entries should be returned successfully, and the position must not be advanced further.

Notes

- What the position represents is entirely up to the FS. it could just represent a counter, in which case ENOENT will never have to be thrown. While seeks to unaligned positions are not allowed, seeks beyond the directory end should just be taken as EOF.
- Each stored entry is of type `struct dirent`, with member `d_name` containing the null-terminated name of the directory entry and extra padding to make the following `struct dirent` 32-bit aligned.
- Each dirent's `d_off` indicates the (file-system specific) position of that entry, and `d_reclen` represents the total size of the returned `struct dirent` including name and padding.

- Open Group link (closest match): readdir
- POSIX notes: upon success, the ATIME of the directory is updated.

REQ_FTRUNC

Set size, or free space, of an open file.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_TRC_START_HI	m9_l2	u32_t	new file size or starting position (inclusive) of region to free (upper 32 bits)
REQ_TRC_START_LO	m9_l3	u32_t	new file size or starting position (inclusive) of region to free (lower 32 bits)
REQ_TRC_END_HI	m9_l4	u32_t	zero or ending position (exclusive) of region to free (upper 32 bits)
REQ_TRC_END_LO	m9_l5	u32_t	zero or ending position (exclusive) of region to free (lower 32 bits)

Reply fields

none

Reply codes

EFBIG	the resulting file would be too big
OK	file size changed and/or holes created

Description

The meaning of this request depends on whether the REQ_TRC_END value is zero or not. If it is zero, the size of the given file has to be adjusted, truncating or expanding the file to the size passed in REQ_TRC_START. Expanding means filling with zeroes, using file holes if supported. If REQ_TRC_END is not zero, then the region within the file from REQ_TRC_START to REQ_TRC_END is to be freed, using file holes if possible. File servers that do not support file holes should zerofill the given region, providing the same net effect to the caller. In any case, freeing space alone must not change the file size.

Notes

- VFS guarantees that this call is issued only on regular files.
- VFS guarantees that if REQ_TRC_END is nonzero, it is greater than REQ_TRC_START, and no greater than the file size.
- Regions for freeing space are not aligned on block boundaries. On block-oriented file systems that do support file holes, the file server may still have to zero part of the first and last block that the region covers partially.

- This is a write request, and will not be sent to read-only file systems.
- Open Group links: ftruncate, truncate
- POSIX notes: upon success, the file's CTIME and MTIME are updated, even when the file size has not changed.

REQ_INHIBREAD

Mark file as target of seek operation.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number

Reply fields

none

Reply codes

OK	request processed successfully

Description

This request may help determine a file system's read-ahead behavior. After a seek, the file server may choose not to do aggressive read-ahead along with the next read call, because it is likely that another seek will follow after (for applications following a seek-read-seek-read pattern). The next read call may enable read-ahead for the read after that (seek-read-read) again.

Notes

- This scheme assumes only one reader per file at any time.

Inode metadata retrieval and manipulation

REQ_STAT

Retrieve file status.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (WRITE) to store resulting "struct stat" in

Reply fields

none

Reply codes

OK	result stored in buffer

Description

The file server is requested to fill a "struct stat" structure with details about the given file, and copy it out using the provided grant.

Notes

- The st_dev field of the structure should be set to the value provided with the initial REQ_READSUPER request.
- Open Group links: fstat, stat
- Open Group update (2008):

REQ_CHOWN

Change file ownership.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_UID	m9_s4	uid_t	new user ID for the file
REQ_GID	m9_s1	gid_t	new group ID for the file

Reply fields

RES_MODE	m9_s2	mode_t	*upon success:* resulting inode mode

Reply codes

OK	ownership changed

Description

This request changes ownership of the given file to a new user and group ID. The file server is expected to clear the setuid and setgid bits from the file, and send back the resulting mode to VFS for synchronization of the vnode's cached mode.

Notes

- This is a write request, and will not be sent to read-only file systems.
- Open Group links: fchown, chown

- POSIX notes: the CTIME of the file is updated.

REQ_CHMOD

Change file mode.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_MODE	m9_s3	mode_t	new mode for the file

Reply fields

RES_MODE	m9_s2	mode_t	*upon success:* resulting inode mode

Reply codes

OK	mode changed

Description

This request changes the access permissions part (but not the type part) of the mode of the given inode. The resulting mode is sent back to VFS.

Notes

- The access permissions mask to use on the given mode is 07777 octal, and defined as ALL_MODES in `<minix/const.h>`.
- This is a write request, and will not be sent to read-only file systems.
- Open Group links: fchmod, chmod
- POSIX notes: the CTIME of the file is updated.

REQ_UTIME

Set file times.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_ACTIME	m9_l2	time_t	new access time, in seconds since Epoch
REQ_MODTIME	m9_l3	time_t	new modification time, in seconds since Epoch
REQ_ACNSEC	m9_l4	long	new access nanoseconds, or special value UTIME_OMIT or UTIME_NOW
REQ_MODNSEC	m9_l5	long	new modification nanoseconds, or special value UTIME_OMIT or UTIME_NOW

Reply fields

none

Reply codes

OK	custom file times set

Description

The purpose of this request is to set a custom access time stamp (ATIME), and modification time stamp (MTIME) on a file. Time stamps are expressed in seconds and nanoseconds, like the two members of `struct timespec`. For each of the two time stamps, the special value UTIME_OMIT (defined in <sys/stat.h>) asks to avoid modification of the data stored in the file system; and the special value UTIME_NOW (also defined in <sys/stat.h>) asks to set the time stamp to the current time.

Notes

- This is a write request, and will not be sent to read-only file systems.
- If UTIME_NOW is passed, the count of seconds passed in the corresponding time_t value is set by VFS to the current time.
- POSIX requires rounding of time stamps to be done toward zero; so discard nanoseconds if the file system does not support them; this also works for UTIME_NOW.
- Open Group link: utime and

- Open Group update (2008): futimens and utimensat and

- POSIX notes: the CTIME of the file is also updated.

Directory entry manipulation

REQ_MKDIR

Create a directory.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number of the containing directory for the new file
REQ_MODE	m9_s3	mode_t	mode for the directory
REQ_UID	m9_s4	uid_t	user ID for the directory
REQ_GID	m9_s1	gid_t	group ID for the directory
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for the last path component
REQ_PATH_LEN	m9_s2	unsigned short	length of the last path component

Reply fields

none

Reply codes

ENAMETOOLONG	the last path component is longer than the file system supports
EEXIST	a directory entry with that name already exists
ENFILE	no inodes are available
ENOSPC	no space is left on the device
EFBIG	the containing directory can not handle any more entries
EMLINK	the containing directory has the maximum number of links already
ENOENT	the containing directory has been removed
OK	directory created

Description

This request creates a new directory with the given properties.

Notes

- The given mode includes the S_IFDIR type.
- This is a write request, and will not be sent to read-only file systems.
- Open Group link: mkdir
- POSIX notes: upon success, the new directory's ATIME, CTIME and MTIME, and the containing directory's CTIME and MTIME are updated.

REQ_MKNOD

Create a special file.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number of the containing directory for the new file
REQ_MODE	m9_s3	mode_t	mode for the file
REQ_DEV	m9_l5	dev_t	device number
REQ_UID	m9_s4	uid_t	user ID for the file
REQ_GID	m9_s1	gid_t	group ID for the file
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for the last path component
REQ_PATH_LEN	m9_s2	unsigned short	length of the last path component

Reply fields

none

Reply codes

ENAMETOOLONG	the last path component is longer than the file system supports
EEXIST	a directory entry with that name already exists
EINVAL	the given file type is invalid or not supported
ENFILE	no inodes are available
ENOSPC	no space is left on the device
EFBIG	the containing directory can not handle any more entries
ENOENT	the containing directory has been removed
OK	special file created

Description

This request creates a new file with the given properties. The type contained in the given mode is arbitrary, and may even be invalid. Interpretation is up to the file system.

Notes

- This is a write request, and will not be sent to read-only file systems.
- Open Group link: mknod
- POSIX notes: upon success, the new file's ATIME, CTIME and MTIME, and the containing directory's CTIME and MTIME are updated.

REQ_LINK

Create a hard link to a file.

Request fields

REQ_INODE_NR	m9_l1	ino_t	link file inode number
REQ_DIR_INO	m9_l3	ino_t	inode number of the containing directory for the new link
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for the last path component
REQ_PATH_LEN	m9_s2	unsigned short	length of the last path component

Reply fields

none

Reply codes

ENAMETOOLONG	the last path component is longer than the file system supports
EEXIST	a directory entry with that name already exists
EPERM	the linked file is a directory
EMLINK	the linked inode has the maximum number of links already
ENOSPC	no space is left on the device
EFBIG	the containing directory can not handle any more entries
ENOENT	the containing directory has been removed
OK	new link created

Description

This request creates a new directory entry that hardlinks to an existing inode (the file being linked).

Notes

- Linking to directories may, but need not be supported.
- This is a write request, and will not be sent to read-only file systems.
- Open Group link: link
- POSIX notes: upon success, the CTIME of the linked file and the CTIME and MTIME of the containing directory of the new link are updated.

REQ_UNLINK

Unlink a file.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number of the containing

			directory for the file
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for last path component
REQ_PATH_LEN	m9_s2	unsigned short	length of the last path component

Reply fields

none

Reply codes

ENAMETOOLONG	the last path component is longer than the file system supports
ENOENT	no directory entry with that name exists
EPERM	the given name refers to a directory
OK	unlinked file

Description

This request unlinks a file. The inode must not represent a directory.

Notes

- Unlinking of directories, if supported, is only allowed to be done by the super user; since this request does not provide caller information, unlinking of directories can not be supported.
- This is a write request, and will not be sent to read-only file systems.
- Open Group link: unlink
- POSIX notes: upon success, the CTIME of the unlinked file is updated (if still linked elsewhere or open), and the CTIME and MTIME of the containing directory are updated.

REQ_RMDIR

Remove an empty directory.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number of the containing directory for the file
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for last path component
REQ_PATH_LEN	m9_s2	unsigned short	length of the last path component

Reply fields

none

Reply codes

ENAMETOOLONG	the last path component is longer than the file system supports
ENOENT	no directory entry with that name exists
ENOTDIR	the given name does not refer to a directory.
ENOTEMPTY	the given directory is not empty
EINVAL	the given directory is "." or ".."
EBUSY	the given directory is the root directory of the file system
OK	removed directory

Description

This request removes a directory. The inode must represent a directory, and that directory must be empty.

Notes

- This is a write request, and will not be sent to read-only file systems.
- Open Group link: rmdir
- POSIX notes: upon success, the CTIME of the removed directory is updated (if still open), and the CTIME and MTIME of the containing directory are updated.

REQ_RENAME

Rename a file or directory.

Request fields

REQ_REN_OLD_DIR	m9_l3	ino_t	inode number of containing directory for the old file
REQ_REN_NEW_DIR	m9_l4	ino_t	inode number of containing directory for the new file
REQ_REN_GRANT_OLD	m9_l2	cp_grant_id_t	memory grant (READ) for the old last path component
REQ_REN_LEN_OLD	m9_s1	unsigned short	length of the old last path component
REQ_REN_GRANT_NEW	m9_l1	cp_grant_id_t	memory grant (READ) for the new last path component
REQ_REN_LEN_NEW	m9_s2	unsigned short	length of the new last path component

Reply fields

none

Reply codes

ENAMETOOLONG	the last path component of the old or new file is longer than the file system supports
ENOENT	the old file does not exist
OK	the old and new last path component and containing directory are the same
EBUSY	the old file is a mountpoint directory
EINVAL	an attempt is made to move a directory to within its own subtree
EINVAL	the old or new last path component is "." or ".."
EMLINK	the old file is a directory and the new file doesn't exist but the new containing directory has the maximum number of links
ENOTDIR	the old file is a directory and the new file exists but is not a directory
EISDIR	the old file is not a directory and the new file exists but is a directory
ENOTEMPTY	the new file is a directory but is not empty
EBUSY	the new file is the root directory of the file system
ENOSPC	no space is left on the device
EFBIG	the new containing directory can not handle any more entries
ENOENT	the new containing directory has been removed
OK	file renamed

Description

This request renames a file or directory to a new name (last path component) and/or a new containing directory. An attempt to rename a file to itself is a no-op and must return OK. Otherwise, if the new name already exists in the new containing directory, and is of the same directory type as the old file (i.e., directory or not directory), then the call implies an unlink or rmdir on the new last path component first, applying the same restrictions as those calls. Other restrictions apply, generally to maintain file system consistency.

Notes

- This is a write request, and will not be sent to read-only file systems.
- Open Group link: rename
- POSIX notes: upon success, the CTIME and MTIME of the old and new containing directories are updated.

REQ_SLINK

Create a symbolic link.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number of the containing directory for the new file
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for the link name's last path component
REQ_PATH_LEN	m9_s2	unsigned short	length of the link name's last path component
REQ_GRANT3	m9_l3	cp_grant_id_t	memory grant (READ) for the link target (not including a trailing '\0')
REQ_MEM_SIZE	m9_l5	size_t	length of the link target (not including a trailing '\0')
REQ_UID	m9_s4	uid_t	user ID for the new symlink
REQ_GID	m9_s1	gid_t	group ID for the new symlink

Reply fields

none

Reply codes

ENAMETOOLONG	the last path component is longer than the file system supports
EEXIST	a directory entry with that name already exists
ENFILE	no inodes are available
ENOSPC	no space is left on the device
EFBIG	the containing directory can not handle any more entries
ENOENT	the containing directory has been removed
ENAMETOOLONG	the link target contains '\0' bytes
OK	symbolic link created

Description

This request instructs the file server to create a symbolic link entry. The resulting symlink's mode is expected to be S_IFLNK|0777 (octal). The symlink should point to the given link target, which is, as far as this request is concerned, a block of arbitrary data.

Notes

- The link target is guaranteed to be at least 1 byte and less than SYMLINK_MAX bytes.
- This is a write request, and will not be sent to read-only file systems.
- Open Group link: symlink
- POSIX notes: upon success, the new symlink's ATIME, CTIME and MTIME,

and the containing directory's CTIME and MTIME are updated.

REQ_RDLINK

Retrieve symbolic link target.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (WRITE) for buffer to write result to
REQ_MEM_SIZE	m9_l5	size_t	size of buffer to write to

Reply fields

RES_NBYTES	m9_l5	size_t	*upon success:* number of bytes written

Reply codes

OK	result stored in buffer

Description

This call asks the file server to copy the link target of a symbolic link into the given grant. No terminating '\0' character need to be written. If the given buffer is smaller than the link target, a partial result is to be written.

Notes

- VFS does not check the permissions of the symbolic link. A symbolic link is always expected to be readable.
- Open Group link: readlink

Miscellaneous file system operations

REQ_MOUNTPOINT

Mark an inode as mountpoint.

Request fields

REQ_INODE_NR	m9_l1	ino_t	inode number of file to use as mountpoint

Reply fields

none

Reply codes

EBUSY	inode already in use as mountpoint
ENOTDIR	given inode is not a directory
OK	inode marked as mountpoint

Description

This request marks the given inode as being in use as a mountpoint for another file system. VFS will keep the inode open for as long as it is a mountpoint, and send a REQ_PUTNODE request for it when the other file system is unmounted.

Notes

- The mountpoint will not be accessed by inode number for any other requests, besides REQ_LOOKUP requests continuing up from that mountpoint. In essence, an inode that acts as a mountpoint is entirely hidden from the user's point of view.
- VFS guarantees that the reference count of the given inode is 1 at the time of this request.
- It may be that inodes that are part of the directory hierarchy below the given inode, are still open. The file system must keep this in mind. As far as parent modifications go (such as file times), the ultimate ancestor of such inodes is the mountpoint inode, not the mounted inode that is on another file system, so the file system need not make any exceptions there. However, the mountpoint inode must be hidden during lookups going up from such inodes (see the REQ_LOOKUP description).
- In principle, not only directories can be mounted; there is support for mounting regular files in both VFS and MFS as well. A file server may choose to support this by only returning ENOTDIR for inodes that it does not support being a mountpoint.

REQ_FSTATFS

Retrieve file system status.

Request fields

REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (WRITE) to store resulting "struct statfs" in

Reply fields

none

Reply codes

OK	result stored in buffer

Description

The file server is requested to fill a "struct statfs" structure with details about the file system, and copy it out using the provided grant.

Notes

- The only field currently present in this structure is the 'f_bsize' field - the file system block size. This field is used by for example du(1).

REQ_STATVFS

Retrieve file system statistics (The POSIX equivalent of REQ_FSTATFS)

Request fields

REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (WRITE) to store resulting "struct statvfs" in

Reply fields

none

Reply codes

OK	result stored in buffer

Description

The file server is requested to fill a "struct statvfs" structure with details about the file system, and copy it out using the provided grant.

Notes

- By convention, fields that are unknown or not applicable to the file system, should be set to zero.
- Open Group links: fstatvfs, statvfs

REQ_SYNC

Write any unwritten data to disk.

Request fields

none

Reply fields

none

Reply codes

OK	request processed successfully

Description

This request tells the file server to flush its cache to the backing device.

Notes

- VFS currently translates fsync() to a sync on the entire file system.
- A root file server must also invalidate any cached data associated with devices other than the one it is mounted on.
- Open Group links: fsync, sync

Block I/O functions

REQ_FLUSH

Flush cached data for an unmounted device.

Request fields

REQ_DEV	m9_l5	dev_t	device number

Reply fields

none

Reply codes

EBUSY	the device is mounted
OK	cache flushed and invalidated for this device

Description

This request is a lightweight version of REQ_SYNC, telling the root file server to give up the part of its cache used for the given device, provided that the device is not the device that the root file system is mounted on.

Notes

- Only the file server providing the root file system will get this request.
- VFS may in fact send requests for mounted devices, but will currently ignore returned error codes.

REQ_NEW_DRIVER

Set a new driver label for a major device.

Request fields

REQ_DEV	m9_l5	dev_t	device number
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) for the label of the block device driver to use
REQ_PATH_LEN	m9_s2	unsigned short	length of the label

Reply fields

none

Reply codes

OK	request processed successfully

Description

This request tells a file server that that the endpoint of a driver may have changed. The request will be sent in two cases: to inform any file server that its underlying block device driver has restarted, and to inform the root file server that the particular driver has to be used for raw block I/O on the given major number from that point on. The request provides the driver's major number, and the label of the driver. This label will always be the same across driver restarts, but may not be the same for block I/O drivers given to the root file server.

The file server must use DS to look up the corresponding endpoint, and use that endpoint to communicate with the driver from then on. In the case of driver restarts, the file server may have to invoke recovery code (for example, to reopen the underlying block device), if it has not found out about the restart through IPC errors already. In general, the file server must only perform recovery upon getting this request if the resulting endpoint is different from the one it knew about.

Notes

- The file server must ignore the minor part of the given device number.
- Only the root file server will receive these requests for devices that it is not mounted on.
- The label is null-terminated, and the given length includes the terminating '\0' character. The maximum label size is always at most DS_MAX_KEYLEN, but VFS currently uses a hardcoded size of 16 (including '\0').

REQ_BREAD

Read from a block device directly.

Request fields

REQ_DEV2	m9_l1	dev_t	device number
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (WRITE) to store the resulting data in
REQ_SEEK_POS_HI	m9_l3	u32_t	read position (upper 32 bits)
REQ_SEEK_POS_LO	m9_l4	u32_t	read position (lower 32 bits)
REQ_NBYTES	m9_l5	size_t	number of bytes to read

Reply fields

RES_SEEK_POS_HI	m9_l3	u32_t	*upon success:* resulting position

			(upper 32 bits)
RES_SEEK_POS_LO	m9_l4	u32_t	*upon success:* resulting position (lower 32 bits)
RES_NBYTES	m9_l5	size_t	*upon success:* total number of bytes read

Reply codes

EIO	I/O error reported by the device driver
OK	results successfully (partially) read, or EOF reached

Description

This request reads from a block device directly, as a result of an application reading from a block-special file. File servers will receive such requests for the block device that they are mounted on. This allows them to keep these transfers in sync with their internal cache.

Notes

- Requests partially beyond the end of the device may be truncated. Requests fully beyond the end of the device should be treated like EOF (i.e. zero bytes are read).
- The file server providing the root file system will also receive these requests for all block devices that have not been mounted.

REQ_BWRITE

Write to a block device directly.

Request fields

REQ_DEV2	m9_l1	dev_t	device number
REQ_GRANT	m9_l2	cp_grant_id_t	memory grant (READ) containing the data to write
REQ_SEEK_POS_HI	m9_l3	u32_t	write position (upper 32 bits)
REQ_SEEK_POS_LO	m9_l4	u32_t	write position (lower 32 bits)
REQ_NBYTES	m9_l5	size_t	number of bytes to write

Reply fields

RES_SEEK_POS_HI	m9_l3	u32_t	*upon success:* resulting position (upper 32 bits)
RES_SEEK_POS_LO	m9_l4	u32_t	*upon success:* resulting position (lower 32 bits)
RES_NBYTES	m9_l5	size_t	*upon success:* total number of bytes written

Reply codes

EIO	I/O error reported by the device driver
OK	results successfully (partially) written, or EOF reached

Description

This request writes from a block device directly, as a result of an application writing to a block-special file. File servers will receive such requests for the block device that they are mounted on. This allows them to keep these transfers in sync with their internal cache. The semantics of writing to a mounted block device are undefined; it is up to the file server to determine how to deal with such requests.

Notes

- Requests partially beyond the end of the device may be truncated. Requests fully beyond the end of the device should be treated like EOF (i.e. zero bytes are written).
- The file server providing the root file system will also receive these requests for all block devices that have not been mounted.
- While this is a write request, it will also be sent to read-only file systems. This is by design.

Additional information

Implementing the lookup request

The following pseudocode presents one possible approach to implementing the heart of the REQ_LOOKUP request:

- retrieve the path to resolve [ENAMETOOLONG]
- take the given start inode as current inode
- while the remaining part of the path is not empty,
    - skip over zero or more '/' characters in the path
    - if remaining path is empty, use "." as component
    - else, get the next '/'-delimited component [ENAMETOOLONG]
    - if the current inode is a mountpoint,
        - if the component is not "..": [EINVAL]
    - else,
        - if the current inode is not a directory: [ENOTDIR]
        - if the caller has no search access permissions on the current inode: [EACCES]
    - if the component is ".", or the component is ".." and the current directory is the given root inode,
        - continue
    - else, if the component is "..",
        - if the current inode is the file system root directory: [ELEAVEMOUNT]

        - set the current inode to the parent inode of the current inode
        - if the now current inode is a mountpoint: [EENTERMOUNT]
    - else, if the component identifies a symlink, and either the component is not the last in the path or PATH_RET_SYMLINK is not set,
        - if the number of symlink resolutions increased by one exceeds SYMLOOP_MAX: [ELOOP]
        - set the new path to the contents of the symlink plus the remaining part of the current path [ENAMETOOLONG]
        - if the now current path starts with a '/': [ESYMLINK]
    - else,
        - set the current inode to the inode identified by the component [ENOENT]
        - if the now current inode is a mountpoint: [EENTERMOUNT]
- open the resulting inode [OK]

For file servers that do not support symbolic links and/or mountpoints, the lookup can be simplified accordingly.

If at least one symlink was resolved, and the returned error is EENTERMOUNT, ELEAVEMOUNT or ESYMLINK, then the unresolved part of the new path must be copied back to VFS.

Regardless,

- In the case of ELEAVEMOUNT, the returned offset must point to the start of the ".." component used in that loop iteration.
- In the case of EENTERMOUNT, the returned offset must point to the first character after the component that resolved to a mountpoint.
- In the case of ESYMLINK, the returned offset is typically zero.

References

An extension of this protocol is under development, to add support for failure resilience and dynamic updates.

This document is *not* based on the original VFS-FS protocol documentation by Balazs Gerofi. However, that document may still provide additional insights.

Design and implementation of the MINIX Virtual File system by Balazs Gerofi, August, 2006