fuse_operations Struct Reference

`#include <fuse.h>`

Data Fields

- int(* **getattr**)(const char *, struct stat *, struct **fuse_file_info** *fi)
- int(* **readlink**)(const char *, char *, size_t)
- int(* **mknod**)(const char *, mode_t, dev_t)
- int(* **mkdir**)(const char *, mode_t)
- int(* **unlink**)(const char *)
- int(* **rmdir**)(const char *)
- int(* **symlink**)(const char *, const char *)
- int(* **rename**)(const char *, const char *, unsigned int flags)
- int(* **link**)(const char *, const char *)
- int(* **chmod**)(const char *, mode_t, struct **fuse_file_info** *fi)
- int(* **chown**)(const char *, uid_t, gid_t, struct **fuse_file_info** *fi)
- int(* **truncate**)(const char *, off_t, struct **fuse_file_info** *fi)
- int(* **open**)(const char *, struct **fuse_file_info** *)
- int(* **read**)(const char *, char *, size_t, off_t, struct **fuse_file_info** *)
- int(* **write**)(const char *, const char *, size_t, off_t, struct **fuse_file_info** *)
- int(* **statfs**)(const char *, struct statvfs *)
- int(* **flush**)(const char *, struct **fuse_file_info** *)
- int(* **release**)(const char *, struct **fuse_file_info** *)
- int(* **fsync**)(const char *, int, struct **fuse_file_info** *)
- int(* **setxattr**)(const char *, const char *, const char *, size_t, int)
- int(* **getxattr**)(const char *, const char *, char *, size_t)
- int(* **listxattr**)(const char *, char *, size_t)
- int(* **removexattr**)(const char *, const char *)

int(*	**opendir**)(const char *, struct **fuse_file_info** *)
int(*	**readdir**)(const char *, void *, **fuse_fill_dir_t**, off_t, struct **fuse_file_info** *, enum **fuse_readdir_flags**)
int(*	**releasedir**)(const char *, struct **fuse_file_info** *)
int(*	**fsyncdir**)(const char *, int, struct **fuse_file_info** *)
void *(*	**init**)(struct **fuse_conn_info** *conn, struct **fuse_config** *cfg)
void(*	**destroy**)(void *private_data)
int(*	**access**)(const char *, int)
int(*	**create**)(const char *, mode_t, struct **fuse_file_info** *)
int(*	**lock**)(const char *, struct **fuse_file_info** *, int cmd, struct **flock** *)
int(*	**utimens**)(const char *, const struct timespec tv[2], struct **fuse_file_info** *fi)
int(*	**bmap**)(const char *, size_t blocksize, uint64_t *idx)
int(*	**ioctl**)(const char *, unsigned int cmd, void *arg, struct **fuse_file_info** *, unsigned int flags, void *data)
int(*	**poll**)(const char *, struct **fuse_file_info** *, struct fuse_pollhandle *ph, unsigned *reventsp)
int(*	**write_buf**)(const char *, struct **fuse_bufvec** *buf, off_t off, struct **fuse_file_info** *)
int(*	**read_buf**)(const char *, struct **fuse_bufvec** **bufp, size_t size, off_t off, struct **fuse_file_info** *)
int(*	**flock**)(const char *, struct **fuse_file_info** *, int op)
int(*	**fallocate**)(const char *, int, off_t, off_t, struct **fuse_file_info** *)
ssize_t(*	**copy_file_range**)(const char *path_in, struct **fuse_file_info** *fi_in, off_t offset_in, const char *path_out, struct **fuse_file_info** *fi_out, off_t offset_out, size_t size, int flags)

Detailed Description

The file system operations:

Most of these should work very similarly to the well known UNIX file system operations. A major exception is that instead of returning an error in 'errno', the operation should return the negated error value (-errno) directly.

All methods are optional, but some are essential for a useful filesystem (e.g. getattr). Open, flush, release, fsync, opendir, releasedir, fsyncdir, access, create, truncate, lock, init and destroy are special purpose methods, without which a full featured filesystem can still be implemented.

In general, all methods are expected to perform any necessary permission checking. However, a filesystem may delegate this task to the kernel by passing the `default_permissions` mount option to `fuse_new()`. In this case, methods will only be called if the kernel's permission check has succeeded.

Almost all operations take a path which can be of any length.

Definition at line **299** of file **fuse.h**.

Field Documentation

◆ access

int(* fuse_operations::access) (const char *, int)

Check file access permissions

This will be called for the **access()** system call. If the 'default_permissions' mount option is given, this method is not called.

This method is not called under Linux kernel versions 2.4.x

Definition at line **597** of file **fuse.h**.

◆ bmap

int(* fuse_operations::bmap) (const char *, size_t blocksize, uint64_t *idx)

Map block index within file to block index within device

Note: This makes sense only for block device backed filesystems mounted with the 'blkdev' option

Definition at line **665** of file **fuse.h**.

◆ chmod

int(* fuse_operations::chmod) (const char *, mode_t, struct **fuse_file_info** *fi)

Change the permission bits of a file

`fi` will always be NULL if the file is not currenlty open, but may also be NULL if the file is open.

Definition at line **367** of file **fuse.h**.

◆ chown

int(* fuse_operations::chown) (const char *, uid_t, gid_t, struct **fuse_file_info** *fi)

Change the owner and group of a file

`fi` will always be NULL if the file is not currenlty open, but may also be NULL if the file is open.

Unless FUSE_CAP_HANDLE_KILLPRIV is disabled, this method is expected to reset the setuid and setgid bits.

Definition at line **377** of file **fuse.h**.

◆ copy_file_range

ssize_t(* fuse_operations::copy_file_range) (const char *path_in, struct **fuse_file_info** *fi_in, off_t offset_in, const char *path_out, struct **fuse_file_info** *fi_out, off_t offset_out, size_t size, int flags)

Copy a range of data from one file to another

Performs an optimized copy between two file descriptors without the additional cost of transferring data through the FUSE kernel module to user space (glibc) and then back into the FUSE filesystem again.

In case this method is not implemented, glibc falls back to reading data from the source and writing to the destination. Effectively doing an inefficient copy of the data.

Definition at line **774** of file **fuse.h**.

◆ create

int(* fuse_operations::create) (const char *, mode_t, struct **fuse_file_info** *)

Create and open a file

If the file does not exist, first create it with the specified mode, and then open it.

If this method is not implemented or under Linux kernel versions earlier than 2.6.15, the **mknod()** and **open()** methods will be called instead.

Definition at line **609** of file **fuse.h**.

◆ destroy

void(* fuse_operations::destroy) (void *private_data)

Clean up filesystem

Called on filesystem exit.

Definition at line **586** of file **fuse.h**.

◆ fallocate

int(* fuse_operations::fallocate) (const char *, int, off_t, off_t, struct **fuse_file_info** *)

Allocates space for an open file

This function ensures that required space is allocated for specified file. If this function returns success then any subsequent write request to specified range is guaranteed not to fail because of lack of space on the file system media.

Definition at line **760** of file **fuse.h**.

◆ flock

int(* fuse_operations::flock) (const char *, struct **fuse_file_info** *, int op)

Perform BSD file locking operation

The op argument will be either LOCK_SH, LOCK_EX or LOCK_UN

Nonblocking requests will be indicated by ORing LOCK_NB to the above operations

For more information see the flock(2) manual page.

Additionally fi->owner will be set to a value unique to this open file. This same value will be supplied to ->**release()** when the file is released.

Note: if this method is not implemented, the kernel will still allow file locking to work locally. Hence it is only interesting for network filesystems and similar.

Definition at line **750** of file **fuse.h**.

◆ flush

int(* fuse_operations::flush) (const char *, struct **fuse_file_info** *)

Possibly flush cached data

BIG NOTE: This is not equivalent to **fsync()**. It's not a request to sync dirty data.

Flush is called on each close() of a file descriptor, as opposed to release which is called on the close of the last file descriptor for a file. Under Linux, errors returned by **flush()** will be passed to userspace as errors from close(), so **flush()** is a good place to write back any cached dirty data. However, many applications ignore errors on close(), and on non-Linux systems, close() may succeed even if **flush()** returns an error. For these reasons, filesystems should not assume that errors returned by flush will ever be noticed or even delivered.

NOTE: The **flush()** method may be called more than once for each **open()**. This happens if more than one file descriptor refers to an open file handle, e.g. due to dup(), dup2() or fork() calls. It is not possible to determine if a flush is final, so each flush should be treated equally. Multiple write-flush sequences are relatively rare, so this shouldn't be a problem.

Filesystems shouldn't assume that flush will be called at any particular point. It may be called more times than expected, or not at all.

Definition at line **496** of file **fuse.h**.

◆ fsync

int(* fuse_operations::fsync) (const char *, int, struct **fuse_file_info** *)

Synchronize file contents

If the datasync parameter is non-zero, then only the user data should be flushed, not the meta data.

Definition at line **517** of file **fuse.h**.

◆ fsyncdir

int(* fuse_operations::fsyncdir) (const char *, int, struct **fuse_file_info** *)

Synchronize directory contents

If the datasync parameter is non-zero, then only the user data should be flushed, not the meta data

Definition at line **568** of file **fuse.h**.

◆ getattr

int(* fuse_operations::getattr) (const char *, struct stat *, struct **fuse_file_info** *fi)

Get file attributes.

Similar to stat(). The 'st_dev' and 'st_blksize' fields are ignored. The 'st_ino' field is ignored except if the 'use_ino' mount option is given. In that case it is passed to userspace, but libfuse and the kernel will still assign a different inode for internal use (called the "nodeid").

`fi` will always be NULL if the file is not currently open, but may also be NULL if the file is open.

Definition at line **311** of file **fuse.h**.

◆ getxattr

int(* fuse_operations::getxattr) (const char *, const char *, char *, size_t)

Get extended attributes

Definition at line **523** of file **fuse.h**.

◆ init

void*(* fuse_operations::init) (struct **fuse_conn_info** *conn, struct **fuse_config** *cfg)

Initialize filesystem

The return value will passed in the `private_data` field of `struct fuse_context` to all file operations, and as a parameter to the **destroy()** method. It overrides the initial value provided to **fuse_main()** / **fuse_new()**.

Definition at line **578** of file **fuse.h**.

◆ ioctl

int(* fuse_operations::ioctl) (const char *, unsigned int cmd, void *arg, struct **fuse_file_info** *, unsigned int flags, void *data)

Ioctl

flags will have FUSE_IOCTL_COMPAT set for 32bit ioctls in 64bit environment. The size and direction of data is determined by *IOC**() decoding of cmd. For _IOC_NONE, data will be NULL, for _IOC_WRITE data is out area, for _IOC_READ in area and if both are set in/out area. In all non-NULL cases, the area is of _IOC_SIZE(cmd) bytes.

If flags has FUSE_IOCTL_DIR then the **fuse_file_info** refers to a directory file handle.

Note : the unsigned long request submitted by the application is truncated to 32 bits.

Definition at line **683** of file **fuse.h**.

◆ link

int(* fuse_operations::link) (const char *, const char *)

Create a hard link to a file

Definition at line **360** of file **fuse.h**.

◆ listxattr

int(* fuse_operations::listxattr) (const char *, char *, size_t)

List extended attributes

Definition at line **526** of file **fuse.h**.

◆ lock

int(* fuse_operations::lock) (const char *, struct **fuse_file_info** *, int cmd, struct **flock** *)

Perform POSIX file locking operation

The cmd argument will be either F_GETLK, F_SETLK or F_SETLKW.

For the meaning of fields in 'struct flock' see the man page for fcntl(2). The l_whence field will always be set to SEEK_SET.

For checking lock ownership, the 'fuse_file_info->owner' argument must be used.

For F_GETLK operation, the library will first check currently held locks, and if a conflicting lock is found it will return information without calling this method. This ensures, that for local locks the l_pid field is correctly filled in. The results may not be accurate in case of race conditions and in the presence of hard links, but it's unlikely that an application would rely on accurate GETLK results in these cases. If a conflicting lock is not found, this method will be called, and the filesystem may fill out l_pid by a meaningful value, or it may leave this field zero.

For F_SETLK and F_SETLKW the l_pid field will be set to the pid of the process performing the locking operation.

Note: if this method is not implemented, the kernel will still allow file locking to work locally. Hence it is only interesting for network filesystems and similar.

Definition at line **641** of file **fuse.h**.

◆ mkdir

int(* fuse_operations::mkdir) (const char *, mode_t)

Create a directory

Note that the mode argument may not have the type specification bits set, i.e. S_ISDIR(mode) can be false. To obtain the correct directory type bits use mode|S_IFDIR

Definition at line **337** of file **fuse.h**.

◆ mknod

int(* fuse_operations::mknod) (const char *, mode_t, dev_t)

Create a file node

This is called for creation of all non-directory, non-symlink nodes. If the filesystem defines a **create()** method, then for regular files that will be called instead.

Definition at line **329** of file **fuse.h**.

◆ open

int(* fuse_operations::open) (const char *, struct **fuse_file_info** *)

Open a file

Open flags are available in fi->flags. The following rules apply.

- Creation (O_CREAT, O_EXCL, O_NOCTTY) flags will be filtered out / handled by the kernel.
- Access modes (O_RDONLY, O_WRONLY, O_RDWR, O_EXEC, O_SEARCH) should be used by the filesystem to check if the operation is permitted. If the `-o default_permissions` mount option is given, this check is already done by the kernel before calling **open()** and may thus be omitted by the filesystem.
- When writeback caching is enabled, the kernel may send read requests even for files opened with O_WRONLY. The filesystem should be prepared to handle this.
- When writeback caching is disabled, the filesystem is expected to properly handle the O_APPEND flag and ensure that each write is appending to the end of the file.
- When writeback caching is enabled, the kernel will handle O_APPEND. However, unless all changes to the file come through the kernel this will not work reliably. The filesystem should thus either ignore the O_APPEND flag (and let the kernel handle it), or return an error (indicating that reliably O_APPEND is not available).

Filesystem may store an arbitrary file handle (pointer, index, etc) in fi->fh, and use this in other all other file operations (read, write, flush, release, fsync).

Filesystem may also implement stateless file I/O and not store anything in fi->fh.

There are also some flags (direct_io, keep_cache) which the filesystem may set in fi, to change the way the file is opened. See **fuse_file_info** structure in <**fuse_common.h**> for more details.

If this request is answered with an error code of ENOSYS and FUSE_CAP_NO_OPEN_SUPPORT is set in `fuse_conn_info.capable`, this is treated as success and future calls to open will also succeed without being send to the filesystem process.

Definition at line **436** of file **fuse.h**.

◆ opendir

int(* fuse_operations::opendir) (const char *, struct **fuse_file_info** *)

Open directory

Unless the 'default_permissions' mount option is given, this method should check if opendir is permitted for this directory. Optionally opendir may also return an arbitrary filehandle in the **fuse_file_info** structure, which will be passed to readdir, releasedir and fsyncdir.

Definition at line **539** of file **fuse.h**.

◆ poll

int(* fuse_operations::poll) (const char *, struct **fuse_file_info** *, struct fuse_pollhandle *ph, unsigned *reventsp)

Poll for IO readiness events

Note: If ph is non-NULL, the client should notify when IO readiness events occur by calling fuse_notify_poll() with the specified ph.

Regardless of the number of times poll with a non-NULL ph is received, single notification is enough to clear all. Notifying more times incurs overhead but doesn't harm correctness.

The callee is responsible for destroying ph with **fuse_pollhandle_destroy()** when no longer in use.

Definition at line **701** of file **fuse.h**.

◆ read

int(* fuse_operations::read) (const char *, char *, size_t, off_t, struct **fuse_file_info** *)

Read data from an open file

Read should return exactly the number of bytes requested except on EOF or error, otherwise the rest of the data will be substituted with zeroes. An exception to this is when the 'direct_io' mount option is specified, in which case the return value of the read system call will reflect the return value of this operation.

Definition at line **447** of file **fuse.h**.

◆ read_buf

int(* fuse_operations::read_buf) (const char *, struct **fuse_bufvec** **bufp, size_t size, off_t off, struct **fuse_file_info** *)

Store data from an open file in a buffer

Similar to the **read()** method, but data is stored and returned in a generic buffer.

No actual copying of data has to take place, the source file descriptor may simply be stored in the buffer for later data transfer.

The buffer must be allocated dynamically and stored at the location pointed to by bufp. If the buffer contains memory regions, they too must be allocated using malloc(). The allocated memory will be freed by the caller.

Definition at line **730** of file **fuse.h**.

◆ readdir

int(* fuse_operations::readdir) (const char *, void *, **fuse_fill_dir_t**, off_t, struct **fuse_file_info** *, enum **fuse_readdir_flags**)

Read directory

The filesystem may choose between two modes of operation:

1) The readdir implementation ignores the offset parameter, and passes zero to the filler function's offset. The filler function will not return '1' (unless an error happens), so the whole directory is read in a single readdir operation.

2) The readdir implementation keeps track of the offsets of the directory entries. It uses the offset parameter and always passes non-zero offset to the filler function. When the buffer is full (or an error happens) the filler function will return '1'.

Definition at line **556** of file **fuse.h**.

◆ readlink

int(* fuse_operations::readlink) (const char *, char *, size_t)

Read the target of a symbolic link

The buffer should be filled with a null terminated string. The buffer size argument includes the space for the terminating null character. If the linkname is too long to fit in the buffer, it should be truncated. The return value should be 0 for success.

Definition at line **321** of file **fuse.h**.

◆ release

int(* fuse_operations::release) (const char *, struct **fuse_file_info** *)

Release an open file

Release is called when there are no more references to an open file: all file descriptors are closed and all memory mappings are unmapped.

For every **open()** call there will be exactly one **release()** call with the same flags and file handle. It is possible to have a file opened more than once, in which case only the last release will mean, that no more reads/writes will happen on the file. The return value of release is ignored.

Definition at line **510** of file **fuse.h**.

◆ releasedir

int(* fuse_operations::releasedir) (const char *, struct **fuse_file_info** *)

Release directory

Definition at line **561** of file **fuse.h**.

◆ removexattr

int(* fuse_operations::removexattr) (const char *, const char *)

Remove extended attributes

Definition at line **529** of file **fuse.h**.

◆ rename

int(* fuse_operations::rename) (const char *, const char *, unsigned int flags)

Rename a file

flags may be `RENAME_EXCHANGE` or `RENAME_NOREPLACE`. If RENAME_NOREPLACE is specified, the filesystem must not overwrite *newname* if it exists and return an error instead. If `RENAME_EXCHANGE` is specified, the filesystem must atomically exchange the two files, i.e. both must exist and neither may be deleted.

Definition at line **357** of file **fuse.h**.

◆ rmdir

int(* fuse_operations::rmdir) (const char *)

Remove a directory

Definition at line **343** of file **fuse.h**.

◆ setxattr

int(* fuse_operations::setxattr) (const char *, const char *, const char *, size_t, int)

Set extended attributes

Definition at line **520** of file **fuse.h**.

◆ statfs

int(* fuse_operations::statfs) (const char *, struct statvfs *)

Get file system statistics

The 'f_favail', 'f_fsid' and 'f_flag' fields are ignored

Definition at line **466** of file **fuse.h**.

◆ symlink

int(* fuse_operations::symlink) (const char *, const char *)

Create a symbolic link

Definition at line **346** of file **fuse.h**.

◆ truncate

int(* fuse_operations::truncate) (const char *, off_t, struct **fuse_file_info** *fi)

Change the size of a file

`fi` will always be NULL if the file is not currenlty open, but may also be NULL if the file is open.

Unless FUSE_CAP_HANDLE_KILLPRIV is disabled, this method is expected to reset the setuid and setgid bits.

Definition at line **387** of file **fuse.h**.

◆ unlink

int(* fuse_operations::unlink) (const char *)

Remove a file

Definition at line **340** of file **fuse.h**.

◆ utimens

int(* fuse_operations::utimens) (const char *, const struct timespec tv[2], struct **fuse_file_info** *fi)

Change the access and modification times of a file with nanosecond resolution

This supersedes the old utime() interface. New applications should use this.

`fi` will always be NULL if the file is not currenlty open, but may also be NULL if the file is open.

See the utimensat(2) man page for details.

Definition at line **656** of file **fuse.h**.

◆ write

int(* fuse_operations::write) (const char *, const char *, size_t, off_t, struct **fuse_file_info** *)

Write data to an open file

Write should return exactly the number of bytes requested except on error. An exception to this is when the 'direct_io' mount option is specified (see read operation).

Unless FUSE_CAP_HANDLE_KILLPRIV is disabled, this method is expected to reset the setuid and setgid bits.

Definition at line **459** of file **fuse.h**.

◆ write_buf

int(* fuse_operations::write_buf) (const char *, struct **fuse_bufvec** *buf, off_t off, struct **fuse_file_info** *)

Write contents of buffer to an open file

Similar to the **write()** method, but data is supplied in a generic buffer. Use **fuse_buf_copy()** to transfer data to the destination.

Unless FUSE_CAP_HANDLE_KILLPRIV is disabled, this method is expected to reset the setuid and setgid bits.

Definition at line **713** of file **fuse.h**.

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

- include/**fuse.h**