

NAME

fanotify_mark - add, remove, or modify an fanotify mark on a filesystem object

SYNOPSIS

```
#include <sys/fanotify.h>
```

```
int fanotify_mark(int fanotify_fd, unsigned int flags,  
uint64_t mask, int dirfd, const char *pathname);
```

DESCRIPTION

For an overview of the fanotify API, see [fanotify\(7\)](#).

[fanotify_mark\(2\)](#) adds, removes, or modifies an fanotify mark on a filesystem object. The caller must have read permission on the filesystem object that is to be marked.

The *fanotify_fd* argument is a file descriptor returned by [fanotify_init\(2\)](#).

flags is a bit mask describing the modification to perform. It must include exactly one of the following values:

FAN_MARK_ADD

The events in *mask* will be added to the mark mask (or to the ignore mask). *mask* must be nonempty or the error **EINVAL** will occur.

FAN_MARK_REMOVE

The events in argument *mask* will be removed from the mark mask (or from the ignore mask). *mask* must be nonempty or the error **EINVAL** will occur.

FAN_MARK_FLUSH

Remove either all mount or all non-mount marks from the fanotify group. If *flag* contains **FAN_MARK_MOUNT**, all marks for mounts are removed from the group. Otherwise, all marks for directories and files are removed. No flag other than **FAN_MARK_MOUNT** can be used in conjunction with **FAN_MARK_FLUSH**. *mask* is ignored.

If none of the values above is specified, or more than one is specified, the call fails with the error **EINVAL**.

In addition, zero or more of the following values may be ORed into *flags*:

FAN_MARK_DONT_FOLLOW

If *pathname* is a symbolic link, mark the link itself, rather than the file to which it refers. (By default, [fanotify_mark\(\)](#) dereferences *pathname* if it is a symbolic link.)

FAN_MARK_ONLYDIR

If the filesystem object to be marked is not a directory, the error **ENOTDIR** shall be raised.

FAN_MARK_MOUNT

Mark the mount point specified by *pathname*. If *pathname* is not itself a mount point, the mount point containing *pathname* will be marked. All directories, subdirectories, and the contained files of the mount point will be monitored.

FAN_MARK_IGNORED_MASK

The events in *mask* shall be added to or removed from the ignore mask.

FAN_MARK_IGNORED_SURV_MODIFY

The ignore mask shall survive modify events. If this flag is not set, the ignore mask is cleared when a modify event occurs for the ignored file or directory.

mask defines which events shall be listened for (or which shall be ignored). It is a bit mask composed of the following values:

FAN_ACCESS

Create an event when a file or directory (but see BUGS) is accessed (read).

FAN_MODIFY

Create an event when a file is modified (write).

FAN_CLOSE_WRITE

Create an event when a writable file is closed.

FAN_CLOSE_NOWRITE

Create an event when a read-only file or directory is closed.

FAN_OPEN

Create an event when a file or directory is opened.

FAN_OPEN_PERM

Create an event when a permission to open a file or directory is requested. An fanotify file descriptor created with **FAN_CLASS_PRE_CONTENT** or **FAN_CLASS_CONTENT** is required.

FAN_ACCESS_PERM

Create an event when a permission to read a file or directory is requested. An fanotify file descriptor created with **FAN_CLASS_PRE_CONTENT** or **FAN_CLASS_CONTENT** is required.

FAN_ONDIR

Create events for directories—for example, when **opendir(2)**, **readdir(2)** (but see BUGS), and **closedir(2)** are called. Without this flag, only events for files are created.

FAN_EVENT_ON_CHILD

Events for the immediate children of marked directories shall be created. The flag has no effect when marking mounts. Note that events are not generated for children of the sub-directories of marked directories. To monitor complete directory trees it is necessary to mark the relevant mount.

The following composed value is defined:

FAN_CLOSE

A file is closed (**FAN_CLOSE_WRITE|FAN_CLOSE_NOWRITE**).

The filesystem object to be marked is determined by the file descriptor *dirfd* and the pathname specified in *pathname*:

- * If *pathname* is NULL, *dirfd* defines the filesystem object to be marked.
- * If *pathname* is NULL, and *dirfd* takes the special value **AT_FDCWD**, the current working directory is to be marked.
- * If *pathname* is absolute, it defines the filesystem object to be marked, and *dirfd* is ignored.
- * If *pathname* is relative, and *dirfd* does not have the value **AT_FDCWD**, then the filesystem object to be marked is determined by interpreting *pathname* relative the directory referred to by *dirfd*.
- * If *pathname* is relative, and *dirfd* has the value **AT_FDCWD**, then the filesystem object to be marked is determined by interpreting *pathname* relative the current working directory.

RETURN VALUE

On success, **fanotify_mark()** returns 0. On error, -1 is returned, and *errno* is set to indicate the error.

ERRORS**EBADF**

An invalid file descriptor was passed in *fanotify_fd*.

EINVAL

An invalid value was passed in *flags* or *mask*, or *fanotify_fd* was not an fanotify file descriptor.

EINVAL

The fanotify file descriptor was opened with **FAN_CLASS_NOTIF** and *mask* contains a flag for permission events (**FAN_OPEN_PERM** or **FAN_ACCESS_PERM**).

ENOENT

The filesystem object indicated by *dirfd* and *pathname* does not exist. This error also occurs when trying to remove a mark from an object which is not marked.

ENOMEM

The necessary memory could not be allocated.

ENOSPC

The number of marks exceeds the limit of 8192 and the **FAN_UNLIMITED_MARKS** flag was not specified when the fanotify file descriptor was created with [fanotify_init\(2\)](#).

ENOSYS

This kernel does not implement **fanotify_mark()**. The fanotify API is available only if the kernel was configured with **CONFIG_FANOTIFY**.

ENOTDIR

flags contains **FAN_MARK_ONLYDIR**, and *dirfd* and *pathname* do not specify a directory.

VERSIONS

fanotify_mark() was introduced in version 2.6.36 of the Linux kernel and enabled in version 2.6.37.

CONFORMING TO

This system call is Linux-specific.

BUGS

The following bugs were present in Linux kernels before version 3.16:

- * If *flags* contains **FAN_MARK_FLUSH**, *dirfd* and *pathname* must specify a valid filesystem object, even though this object is not used.
- * [readdir\(2\)](#) does not generate a **FAN_ACCESS** event.
- * If [fanotify_mark\(2\)](#) is called with **FAN_MARK_FLUSH**, *flags* is not checked for invalid values.

SEE ALSO

[fanotify_init\(2\)](#), [fanotify\(7\)](#)

COLOPHON

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <http://www.kernel.org/doc/man-pages/>.