NAME

fanotify_init − create and initialize fanotify group

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <fcntl.h> /* Definition of **O_*** constants */
#include <sys/fanotify.h>

int fanotify_init(unsigned int *fla gs*, **unsigned int** *event_f_flags***);**

DESCRIPTION

For an overview of the fanotify API, see **fanotify**(7).

fanotify_init() initializes a new fanotify group and returns a file descriptor for the event queue associated with the group.

The file descriptor is used in calls to **fanotify_mark**(2) to specify the files, directories, mounts, or filesystems for which fanotify events shall be created. These events are received by reading from the file descriptor. Some events are only informative, indicating that a file has been accessed. Other events can be used to determine whether another application is permitted to access a file or directory. Permission to access filesystem objects is granted by writing to the file descriptor.

Multiple programs may be using the fanotify interface at the same time to monitor the same files.

The number of fanotify groups per user is limited. See **fanotify**(7) for details about this limit.

The *flags* argument contains a multi-bit field defining the notification class of the listening application and further single bit fields specifying the behavior of the file descriptor.

If multiple listeners for permission events exist, the notification class is used to establish the sequence in which the listeners receive the events.

Only one of the following notification classes may be specified in *fla gs*:

FAN_CLASS_PRE_CONTENT
This value allows the receipt of events notifying that a file has been accessed and events for permission decisions if a file may be accessed. It is intended for event listeners that need to access files before they contain their final data. This notification class might be used by hierarchical storage managers, for example. Use of this flag requires the**CAP_SYS_ADMIN** capability .

FAN_CLASS_CONTENT
This value allows the receipt of events notifying that a file has been accessed and events for permission decisions if a file may be accessed. It is intended for event listeners that need to access files when they already contain their final content. This notification class might be used by malware detection programs, for example. Use of this flag requires the**CAP_SYS_ADMIN** capability.

FAN_CLASS_NOTIF
This is the default value. It does not need to be specified. This value only allows the receipt of events notifying that a file has been accessed. Permission decisions before the file is accessed are not possible.

Listeners with different notification classes will receive events in the order **FAN_CLASS_PRE_CONTENT**, **FAN_CLASS_CONTENT**, **FAN_CLASS_NOTIF**. The order of notification for listeners in the same notification class is undefined.

The following bits can additionally be set in *fla gs*:

FAN_CLOEXEC
Set the close-on-exec flag (**FD_CLOEXEC**) on the new file descriptor. See the description of the **O_CLOEXEC** flag in **open**(2).

FAN_NONBLOCK
Enable the nonblocking flag (**O_NONBLOCK**) for the file descriptor. Reading from the file descriptor will not block. Instead, if no data is available, **read**(2) fails with the error **EAGAIN**.

FAN_UNLIMITED_QUEUE
Remove the limit on the number of events in the event queue. See **fanotify**(7) for details about this limit. Use of this flag requires the **CAP_SYS_ADMIN** capability.

FAN_UNLIMITED_MARKS
Remove the limit on the number of fanotify marks per user. See**fanotify**(7) for details about this limit. Use of this flag requires the**CAP_SYS_ADMIN** capability .

FAN_REPORT_TID (since Linux 4.20)
Report thread ID (TID) instead of process ID (PID) in the *pid* field of the *struct fanotify_event_metadata* supplied to **read**(2) (see **fanotify**(7)). Use of this flag requires the **CAP_SYS_ADMIN** capability.

FAN_ENABLE_AUDIT (since Linux 4.15)
Enable generation of audit log records about access mediation performed by permission events. The permission event response has to be marked with the **FAN_AUDIT** flag for an audit log record to be generated. Use of this flag requires the **CAP_AUDIT_WRITE** capability.

FAN_REPORT_FID (since Linux 5.1)
This value allows the receipt of events which contain additional information about the underlying filesystem object correlated to an event. An additional record of type **FAN_EVENT_INFO_TYPE_FID** encapsulates the information about the object and is included alongside the generic event metadata structure. The file descriptor that is used to represent the object correlated to an event is instead substituted with a file handle. It is intended for applications that may find the use of a file handle to identify an object more suitable than a file descriptor. Additionally, it may be used for applications monitoring a directory or a filesystem that are interested in the directory entry modification events **FAN_CREATE**, **FAN_DELETE**, **FAN_MOVE**, and **FAN_RENAME**, or in events such as **FAN_ATTRIB**, **FAN_DELETE_SELF**, and **FAN_MOVE_SELF**. All the events above require an fanotify group that identifies filesystem objects by file handles. Note that without the flag **FAN_REPORT_TARGET_FID**, for the directory entry modification events, there is an information record that identifies the modified directory and not the created/deleted/moved child object. The use of **FAN_CLASS_CONTENT** or **FAN_CLASS_PRE_CONTENT** is not permitted with this flag and will result in the error **EINVAL**. See**fanotify**(7) for additional details.

FAN_REPORT_DIR_FID (since Linux 5.9)
Events for fanotify groups initialized with this flag will contain (see exceptions below) additional information about a directory object correlated to an event. An additional record of type **FAN_EVENT_INFO_TYPE_DFID** encapsulates the information about the directory object and is included alongside the generic event metadata structure. For events that occur on a non-directory object, the additional structure includes a file handle that identifies the parent directory filesystem object. Note that there is no guarantee that the directory filesystem object will be found at the location described by the file handle information at the time the event is received. When combined with the flag **FAN_REPORT_FID**, two records may be reported with events that occur on a non-directory object, one to identify the non-directory object itself and one to identify the parent directory object. Note that in some cases, a filesystem object does not have a parent, for example, when an event occurs on an unlinked but open file. In that case, with the **FAN_REPORT_FID** flag, the event will be reported with only one record to identify the non-directory object itself, because there is no directory associated with the event. Without the **FAN_REPORT_FID** flag, no event will be reported. See **fanotify**(7) for additional details.

FAN_REPORT_NAME (since Linux 5.9)
Events for fanotify groups initialized with this flag will contain additional information about the name of the directory entry correlated to an event. This flag must be provided in conjunction with the flag **FAN_REPORT_DIR_FID**. Providing this flag value without **FAN_REPORT_DIR_FID**

will result in the error **EINVAL**. This flag may be combined with the flag**F AN_REPORT_FID**. An additional record of type **FAN_EVENT_INFO_TYPE_DFID_NAME**, which encapsulates the information about the directory entry, is included alongside the generic event metadata structure and substitutes the additional information record of type **FAN_EVENT_INFO_TYPE_DFID**. The additional record includes a file handle that identifies a directory filesystem object followed by a name that identifies an entry in that directory. For the directory entry modification events **FAN_CREATE**, **FAN_DELETE**, and **FAN_MOVE**, the reported name is that of the created/deleted/moved directory entry. The event **FAN_RENAME** may contain two information records. One of type **FAN_EVENT_INFO_TYPE_OLD_DFID_NAME** identifying the old directory entry, and another of type **FAN_EVENT_INFO_TYPE_NEW_DFID_NAME** identifying the new directory entry. For other events that occur on a directory object, the reported file handle is that of the directory object itself and the reported name is '.'. For other events that occur on a non-directory object, the reported file handle is that of the parent directory object and the reported name is the name of a directory entry where the object was located at the time of the event. The rationale behind this logic is that the reported directory file handle can be passed to **open_by_handle_at**(2) to get an open directory file descriptor and that file descriptor along with the reported name can be used to call **fstatat**(2). The same rule that applies to record type **FAN_EVENT_INFO_TYPE_DFID** also applies to record type **FAN_EVENT_INFO_TYPE_DFID_NAME**: if a non-directory object has no parent, either the event will not be reported or it will be reported without the directory entry information. Note that there is no guarantee that the filesystem object will be found at the location described by the directory entry information at the time the event is received. See**fanotify**(7) for additional details.

FAN_REPORT_DFID_NAME
This is a synonym for (**FAN_REPORT_DIR_FID|FAN_REPORT_NAME**).

FAN_REPORT_TARGET_FID (since Linux 5.17)
Events for fanotify groups initialized with this flag will contain additional information about the child correlated with directory entry modification events. This flag must be provided in conjunction with the flags **FAN_REPORT_FID**, **FAN_REPORT_DIR_FID** and **FAN_REPORT_NAME**. or else the error**EINV AL** will be returned. For the directory entry modification events **FAN_CREATE**, **FAN_DELETE**, **FAN_MOVE**, and **FAN_RENAME**, an additional record of type **FAN_EVENT_INFO_TYPE_FID**, is reported in addition to the information records of type **FAN_EVENT_INFO_TYPE_DFID**, **FAN_EVENT_INFO_TYPE_DFID_NAME**, **FAN_EVENT_INFO_TYPE_OLD_DFID_NAME**, and **FAN_EVENT_INFO_TYPE_NEW_DFID_NAME**. The additional record includes a file handle that identifies the filesystem child object that the directory entry is referring to.

FAN_REPORT_DFID_NAME_TARGET
This is a synonym for (**FAN_REPORT_DFID_NAME|FAN_REPORT_FID|FAN_REPORT_TARGET_FID**).

FAN_REPORT_PIDFD (since Linux 5.15)
Events for fanotify groups initialized with this flag will contain an additional information record alongside the generic *fanotify_e vent_metadata* structure. This information record will be of type **FAN_EVENT_INFO_TYPE_PIDFD** and will contain a pidfd for the process that was responsible for generating an event. A pidfd returned in this information record object is no different to the pidfd that is returned when calling **pidfd_open**(2). Usage of this information record are for applications that may be interested in reliably determining whether the process responsible for generating an event has been recycled or terminated. The use of the **FAN_REPORT_TID** flag along with **FAN_REPORT_PIDFD** is currently not supported and attempting to do so will result in the error **EINVAL** being returned. This limitation is currently imposed by the pidfd API as it currently only supports the creation of pidfds for thread-group leaders. Creating pidfds for non-thread-group leaders may be supported at some point in the future, so this restriction may eventually be lifted. For more details on information records, see **fanotify**(7).

The *event_f_flags* argument defines the file status flags that will be set on the open file descriptions that are created for fanotify events. For details of these flags, see the description of the *fla gs* values in **open**(2). *event_f_flags* includes a multi-bit field for the access mode. This field can take the following values:

O_RDONLY
: This value allows only read access.

O_WRONLY
: This value allows only write access.

O_RDWR
: This value allows read and write access.

Additional bits can be set in *event_f_flags*. The most useful values are:

O_LARGEFILE
: Enable support for files exceeding 2 GB. Failing to set this flag will result in an **EOVERFLOW** error when trying to open a large file which is monitored by an fanotify group on a 32-bit system.

O_CLOEXEC (since Linux 3.18)
: Enable the close-on-exec flag for the file descriptor. See the description of the**O_CLOEXEC** flag in **open**(2) for reasons why this may be useful.

The following are also allowable: **O_APPEND**, **O_DSYNC**, **O_NOATIME**, **O_NONBLOCK**, and **O_SYNC**. Specifying any other flag in *event_f_flags* yields the error **EINVAL** (but see BUGS).

RETURN VALUE

On success, **fanotify_init**() returns a new file descriptor. On error, −1 is returned, and *errno* is set to indicate the error.

ERRORS

EINVAL
: An invalid value was passed in *fla gs* or *event_f_flags*. **FAN_ALL_INIT_FLAGS** (deprecated since Linux 4.20) defines all allowable bits for *fla gs*.

EMFILE
: The number of fanotify groups for this user exceeds the limit. See **fanotify**(7) for details about this limit.

EMFILE
: The per-process limit on the number of open file descriptors has been reached.

ENOMEM
: The allocation of memory for the notification group failed.

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

EPERM
: The operation is not permitted because the caller lacks a required capability.

VERSIONS

fanotify_init() was introduced in Linux 2.6.36 and enabled in Linux 2.6.37.

Prior to Linux 5.13, calling **fanotify_init**() required the **CAP_SYS_ADMIN** capability. Since Linux 5.13, users may call **fanotify_init**() without the **CAP_SYS_ADMIN** capability to create and initialize an fanotify group with limited functionality.

The limitations imposed on an event listener created by a user without the
CAP_SYS_ADMIN capability are as follows:

- The user cannot request for an unlimited event queue by using **FAN_UNLIMITED_QUEUE**.
- The user cannot request for an unlimited number of marks by using **FAN_UNLIMITED_MARKS**.

- The user cannot request to use either notification classes **FAN_CLASS_CONTENT** or **FAN_CLASS_PRE_CONTENT**. This means that user cannot request permission events.
- The user is required to create a group that identifies filesystem objects by file handles, for example, by providing the **FAN_REPORT_FID** flag.
- The user is limited to only mark inodes. The ability to mark a mount or filesystem via **fanotify_mark**() through the use of **FAN_MARK_MOUNT** or **FAN_MARK_FILESYSTEM** is not permitted.
- The event object in the event queue is limited in terms of the information that is made available to the unprivileged user. A user will also not receive the pid that generated the event, unless the listening process itself generated the event.

STANDARDS

This system call is Linux-specific.

BUGS

The following bug was present before Linux 3.18:

- The **O_CLOEXEC** is ignored when passed in *event_f_flags*.

The following bug was present before Linux 3.14:

- The *event_f_flags* argument is not checked for invalid flags. Flags that are intended only for internal use, such as **FMODE_EXEC**, can be set, and will consequently be set for the file descriptors returned when reading from the fanotify file descriptor.

SEE ALSO

fanotify_mark(2), **fanotify**(7)