Rocky Enterprise Linux 9.2 Manual Pages on command 'fanotify_init.2'

$ man fanotify_init.2

FANOTIFY_INIT(2)   Linux Programmer's Manual   FANOTIFY_INIT(2)

NAME

fanotify_init - create and initialize fanotify group

SYNOPSIS

```
#include <fcntl.h>

#include <sys/fanotify.h>

int fanotify_init(unsigned int flags, 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, directo?ries, 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, indicat?ing 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.

In the current implementation, the number of fanotify groups per user is limited to 128. This limit cannot be overridden.

Calling fanotify_init() requires the CAP_SYS_ADMIN capability. This constraint might be relaxed in future versions of the API. Therefore, certain additional capability checks have been implemented as indicated below.

The flags argument contains a multi-bit field defining the notification class of the lis?tening application and further single bit fields specifying the behavior of the file de?scriptor.

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

Only one of the following notification classes may be specified in flags:

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 exam?ple.

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.

FAN_CLASS_NOTIF

This is the default value. It does not need to be specified. This value only al?lows the receipt of events notifying that a file has been accessed. Permission de?cisions 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 flags:

FAN_CLOEXEC

Set the close-on-exec flag (FD_CLOEXEC) on the new file descriptor. See the de?scription 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 of 16384 events for the event queue. Use of this flag requires the CAP_SYS_ADMIN capability.

FAN_UNLIMITED_MARKS

Remove the limit of 8192 marks. Use of this flag requires the CAP_SYS_ADMIN capa? bility.

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)).

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 in? terested in the directory entry modification events FAN_CREATE, FAN_DELETE, and FAN_MOVE, 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 for the directory entry modification events the reported file handle 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 permit? ted with this flag and will result in the error EINVAL. See fanotify(7) for addi? tional 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 struc? ture 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 oc? curs 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 de? tails.

FAN_REPORT_NAME (since Linux 5.9)

Events for fanotify groups initialized with this flag will contain additional in? formation 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 FAN_REPORT_FID. An additional record of type FAN_EVENT_INFO_TYPE_DFID_NAME, which encapsulates the information about the direc? tory entry, is included alongside the generic event metadata structure and substi? tutes the additional information record of type FAN_EVENT_INFO_TYPE_DFID. The ad? ditional record includes a file handle that identifies a directory filesystem ob? ject followed by a name that identifies an entry in that directory. For the direc? tory entry modification events FAN_CREATE, FAN_DELETE, and FAN_MOVE, the reported name is that of the created/deleted/moved directory entry. For other events that occur on a directory object, the reported file handle is that of the directory ob? ject itself and the reported name is '.'. For other events that occur on a non-di? rectory 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 direc? tory 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).

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 flags 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 fan? otify 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 flags or event_f_flags. FAN_ALL_INIT_FLAGS (depre?

cated since Linux kernel version 4.20) defines all allowable bits for flags.

EMFILE The number of fanotify groups for this user exceeds 128.

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 the CAP_SYS_ADMIN capabil?ity.

VERSIONS

fanotify_init() was introduced in version 2.6.36 of the Linux kernel and enabled in ver?sion 2.6.37.

CONFORMING TO

This system call is Linux-specific.

BUGS

The following bug was present in Linux kernels before version 3.18:

* The O_CLOEXEC is ignored when passed in event_f_flags.

The following bug was present in Linux kernels before version 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)

COLOPHON

This page is part of release 5.10 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 https://www.kernel.org/doc/man-pages/.