NAME

inotify − monitoring file system events

DESCRIPTION

The *inotify* API provides a mechanism for monitoring file system events. Inotify can be used to monitor individual files, or to monitor directories. When a directory is monitored, inotify will return events for the directory itself, and for files inside the directory.

The following system calls are used with this API: **inotify_init**(2) (or **inotify_init1**(2)), **inotify_add_watch**(2), **inotify_rm_watch**(2), **read**(2), and **close**(2).

inotify_init(2) creates an inotify instance and returns a file descriptor referring to the inotify instance. The more recent **inotify_init1**(2) is like **inotify_init**(2), but provides some extra functionality.

inotify_add_watch(2) manipulates the "watch list" associated with an inotify instance. Each item ("watch") in the watch list specifies the pathname of a file or directory, along with some set of events that the kernel should monitor for the file referred to by that pathname. **inotify_add_watch**(2) either creates a new watch item, or modifies an existing watch. Each watch has a unique "watch descriptor", an integer returned by **inotify_add_watch**(2) when the watch is created.

inotify_rm_watch(2) removes an item from an inotify watch list.

When all file descriptors referring to an inotify instance have been closed, the underlying object and its resources are freed for re-use by the kernel; all associated watches are automatically freed.

To determine what events have occurred, an application **read**(2)s from the inotify file descriptor. If no events have so far occurred, then, assuming a blocking file descriptor, **read**(2) will block until at least one event occurs (unless interrupted by a signal, in which case the call fails with the error **EINTR**; see **signal**(7)).

Each successful **read**(2) returns a buffer containing one or more of the following structures:

```
struct inotify_event {
    int      wd;       /* Watch descriptor */
    uint32_t mask;     /* Mask of events */
    uint32_t cookie;   /* Unique cookie associating related
                          events (for rename(2)) */
    uint32_t len;      /* Size of name field */
    char     name[];   /* Optional null-terminated name */
};
```

wd identifies the watch for which this event occurs. It is one of the watch descriptors returned by a previous call to **inotify_add_watch**(2).

mask contains bits that describe the event that occurred (see below).

cookie is a unique integer that connects related events. Currently this is only used for rename events, and allows the resulting pair of **IN_MOVE_FROM** and **IN_MOVE_TO** events to be connected by the application.

The *name* field is only present when an event is returned for a file inside a watched directory; it identifies the file pathname relative to the watched directory. This pathname is null-terminated, and may include further null bytes to align subsequent reads to a suitable address boundary.

The *len* field counts all of the bytes in *name*, including the null bytes; the length of each *inotify_event*

structure is thus *sizeof(inotify_event)+len*.

The behavior when the buffer given to **read**(2) is too small to return information about the next event depends on the kernel version: in kernels before 2.6.21, **read**(2) returns 0; since kernel 2.6.21, **read**(2) fails with the error **EINVAL**.

inotify events

The **inotify_add_watch**(2) *mask* argument and the *mask* field of the *inotify_event* structure returned when **read**(2)ing an inotify file descriptor are both bit masks identifying inotify events. The following bits can be specified in *mask* when calling **inotify_add_watch**(2) and may be returned in the *mask* field returned by **read**(2):

IN_ACCESS File was accessed (read) (*).
IN_ATTRIB Metadata changed, e.g., permissions, timestamps, extended attributes, link count (since Linux 2.6.25), UID, GID, etc. (*).
IN_CLOSE_WRITE
File opened for writing was closed (*).
IN_CLOSE_NOWRITE
File not opened for writing was closed (*).
IN_CREATE File/directory created in watched directory (*).
IN_DELETE File/directory deleted from watched directory (*).
IN_DELETE_SELF
Watched file/directory was itself deleted.
IN_MODIFY File was modified (*).
IN_MOVE_SELF Watched file/directory was itself moved.
IN_MOVED_FROM
File moved out of watched directory (*).
IN_MOVED_TO File moved into watched directory (*).
IN_OPEN File was opened (*).

When monitoring a directory, the events marked with an asterisk (*) above can occur for files in the directory, in which case the *name* field in the returned *inotify_event* structure identifies the name of the file within the directory.

The **IN_ALL_EVENTS** macro is defined as a bit mask of all of the above events. This macro can be used as the *mask* argument when calling **inotify_add_watch**(2).

Two additional convenience macros are **IN_MOVE**, which equates to IN_MOVED_FROM|IN_MOVED_TO, and **IN_CLOSE** which equates to IN_CLOSE_WRITE|IN_CLOSE_NOWRITE.

The following further bits can be specified in *mask* when calling **inotify_add_watch**(2):

IN_DONT_FOLLOW (since Linux 2.6.15)
Don't dereference *pathname* if it is a symbolic link.
IN_MASK_ADD Add (OR) events to watch mask for this pathname if it already exists (instead of replacing mask).
IN_ONESHOT Monitor *pathname* for one event, then remove from watch list.
IN_ONLYDIR (since Linux 2.6.15)
Only watch *pathname* if it is a directory.

The following bits may be set in the *mask* field returned by **read**(2):

IN_IGNORED Watch was removed explicitly (**inotify_rm_watch**(2)) or automatically (file was deleted, or file system was unmounted).
IN_ISDIR Subject of this event is a directory.

IN_Q_OVERFLOW
Event queue overflowed (*wd* is −1 for this event).

IN_UNMOUNT File system containing watched object was unmounted.

/proc interfaces

The following interfaces can be used to limit the amount of kernel memory consumed by inotify:

/proc/sys/fs/inotify/max_queued_events
The value in this file is used when an application calls **inotify_init**(2) to set an upper limit on the number of events that can be queued to the corresponding inotify instance. Events in excess of this limit are dropped, but an **IN_Q_OVERFLOW** event is always generated.

/proc/sys/fs/inotify/max_user_instances
This specifies an upper limit on the number of inotify instances that can be created per real user ID.

/proc/sys/fs/inotify/max_user_watches
This specifies an upper limit on the number of watches that can be created per real user ID.

VERSIONS

Inotify was merged into the 2.6.13 Linux kernel. The required library interfaces were added to glibc in version 2.4. (**IN_DONT_FOLLOW**, **IN_MASK_ADD**, and **IN_ONLYDIR** were only added in version 2.5.)

CONFORMING TO

The inotify API is Linux-specific.

NOTES

Inotify file descriptors can be monitored using **select**(2), **poll**(2), and **epoll**(7). When an event is available, the file descriptor indicates as readable.

Since Linux 2.6.25, signal-driven I/O notification is available for inotify file descriptors; see the discussion of **F_SETFL** (for setting the **O_ASYNC** flag), **F_SETOWN**, and **F_SETSIG** in **fcntl**(2). The *siginfo_t* structure (described in **sigaction**(2)) that is passed to the signal handler has the following fields set: *si_fd* is set to the inotify file descriptor number; *si_signo* is set to the signal number; *si_code* is set to **POLL_IN**; and **POLLIN** is set in *si_band*.

If successive output inotify events produced on the inotify file descriptor are identical (same *wd*, *mask*, *cookie*, and *name*) then they are coalesced into a single event if the older event has not yet been read (but see BUGS).

The events returned by reading from an inotify file descriptor form an ordered queue. Thus, for example, it is guaranteed that when renaming from one directory to another, events will be produced in the correct order on the inotify file descriptor.

The **FIONREAD ioctl**(2) returns the number of bytes available to read from an inotify file descriptor.

Inotify monitoring of directories is not recursive: to monitor subdirectories under a directory, additional watches must be created.

BUGS

In kernels before 2.6.16, the **IN_ONESHOT** *mask* flag does not work.

Before kernel 2.6.25, the kernel code that was intended to coalesce successive identical events (i.e., the two most recent events could potentially be coalesced if the older had not yet been read) instead checked if the most recent event could be coalesced with the *oldest* unread event.

SEE ALSO

inotify_add_watch(2), **inotify_init**(2), **inotify_init1**(2), **inotify_rm_watch**(2), **read**(2), **stat**(2), *Documentation/filesystems/inotify.txt*.

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.