NAME

Linux::Inotify2 - scalable directory/file change notification

SYNOPSIS

Callback Interface

```
use Linux::Inotify2;

# create a new object
my $inotify = new Linux::Inotify2
or die "unable to create new inotify object: $!";

# add watchers
$inotify->watch ("/etc/passwd", IN_ACCESS, sub {
my $e = shift;
my $name = $e->fullname;
print "$name was accessed\n" if $e->IN_ACCESS;
print "$name is no longer mounted\n" if $e->IN_UNMOUNT;
print "$name is gone\n" if $e->IN_IGNORED;
print "events for $name have been lost\n" if $e->IN_Q_OVERFLOW;

# cancel this watcher: remove no further events
$e->w->cancel;
});

# integration into AnyEvent (works with EV, Glib, Tk, POE...)
my $inotify_w = AnyEvent->io (
fh => $inofity->fileno, poll => 'r', cb => sub { $inotify->poll }
);

# manual event loop
1 while $inotify->poll;
```

Streaming Interface

```
use Linux::Inotify2 ;

# create a new object
my $inotify = new Linux::Inotify2
or die "Unable to create new inotify object: $!" ;

# create watch
$inotify->watch ("/etc/passwd", IN_ACCESS)
or die "watch creation failed" ;

while () {
my @events = $inotify->read;
unless (@events > 0) {
print "read error: $!";
last ;
}
printf "mask\t%d\n", $_->mask foreach @events ;
}
```

DESCRIPTION

This module implements an interface to the Linux 2.6.13 and later Inotify file/directory change notification system.

It has a number of advantages over the Linux::Inotify module:

```
- it is portable (Linux::Inotify only works on x86)
- the equivalent of fullname works correctly
- it is better documented
- it has callback-style interface, which is better suited for
integration.
```

The Linux::Inotify2 Class

Linux::Inotify2 Class

my $inotify = new Linux::Inotify2

Create a new notify object and return it. A notify object is kind of a container that stores watches on file system names and is responsible for handling event data.

On error, `undef` is returned and `$!` will be set accordingly. The following errors are documented:

```
ENFILE The system limit on the total number of file descriptors has been reached.
EMFILE The user limit on the total number of inotify instances has been reached.
ENOMEM Insufficient kernel memory is available.
```

Example:

```
my $inotify = new Linux::Inotify2
or die "Unable to create new inotify object: $!";
```

$watch = $inotify->watch ($name, $mask[, $cb])

Add a new watcher to the given notifier. The watcher will create events on the pathname $name as given in $mask, which can be any of the following constants (all exported by default) ORed together.

"file" refers to any file system object in the watched object (always a directory), that is files, directories, symlinks, device nodes etc., while "object" refers to the object the watcher has been set on itself:

```
IN_ACCESS object was accessed
IN_MODIFY object was modified
IN_ATTRIB object metadata changed
IN_CLOSE_WRITE writable fd to file / to object was closed
IN_CLOSE_NOWRITE readonly fd to file / to object closed
IN_OPEN object was opened
IN_MOVED_FROM file was moved from this object (directory)
IN_MOVED_TO file was moved to this object (directory)
IN_CREATE file was created in this object (directory)
IN_DELETE file was deleted from this object (directory)
IN_DELETE_SELF object itself was deleted
IN_MOVE_SELF object itself was moved
IN_ALL_EVENTS all of the above events

IN_ONESHOT only send event once
IN_ONLYDIR only watch the path if it is a directory
IN_DONT_FOLLOW don't follow a sym link
IN_MASK_ADD not supported with the current version of this module

IN_CLOSE same as IN_CLOSE_WRITE | IN_CLOSE_NOWRITE
IN_MOVE same as IN_MOVED_FROM | IN_MOVED_TO
```

$cb is a perl code reference that, if given, is called for each event. It receives a Linux::Inotify2::Event object.

The returned $watch object is of class Linux::Inotify2::Watch

On error, `undef` is returned and `$!` will be set accordingly. The following errors are documented:

```
EBADF The given file descriptor is not valid.
EINVAL The given event mask contains no legal events.
ENOMEM Insufficient kernel memory was available.
ENOSPC The user limit on the total number of inotify watches was reached or the kernel f
EACCESS Read access to the given file is not permitted.
```

Example, show when `/etc/passwd` gets accessed and/or modified once:

```
$inotify->watch ("/etc/passwd", IN_ACCESS | IN_MODIFY, sub {
my $e = shift;
print "$e->{w}{name} was accessed\n" if $e->IN_ACCESS;
print "$e->{w}{name} was modified\n" if $e->IN_MODIFY;
print "$e->{w}{name} is no longer mounted\n" if $e->IN_UNMOUNT;
print "events for $e->{w}{name} have been lost\n" if $e->IN_Q_OVERFLOW;

$e->w->cancel;
});
```

`$inotify`->fileno
: Returns the file descriptor for this notify object. When in non-blocking mode, you are responsible for calling the `poll` method when this file descriptor becomes ready for reading.

`$inotify`->blocking ($blocking)
: Clears ($blocking true) or sets ($blocking false) the `O_NONBLOCK` flag on the file descriptor.

`$count` = `$inotify`->poll
: Reads events from the kernel and handles them. If the notify file descriptor is blocking (the default), then this method waits for at least one event (and thus returns true unless an error occurs). Otherwise it returns immediately when no pending events could be read.

  Returns the count of events that have been handled.

`@events` = `$inotify`->read
: Reads events from the kernel. Blocks when the file descriptor is in blocking mode (default) until any event arrives. Returns list of `Linux::Inotify2::Event` objects or empty list if none (non-blocking mode) or error occurred ($! should be checked).

  Normally you shouldn't use this function, but instead use watcher callbacks and call `->poll`.

The Linux::Inotify2::Event Class

Objects of this class are handed as first argument to the watcher callback. It has the following members and methods:

`$event`->w
`$event`->{w}
: The watcher object for this event.

`$event`->name
`$event`->{name}
: The path of the file system object, relative to the watched name.

`$event`->fullname
: Returns the ''full'' name of the relevant object, i.e. including the `name` member of the watcher (if the watch object is on a directory and a directory entry is affected), or simply the `name` member itself when the object is the watch object itself.

`$event`->mask

`$event->{mask}`
: The received event mask. In addition to the events described for `$inotify->watch`, the following flags (exported by default) can be set:

```
IN_ISDIR event object is a directory
IN_Q_OVERFLOW event queue overflowed

# when any of the following flags are set,
# then watchers for this event are automatically canceled
IN_UNMOUNT filesystem for watched object was unmounted
IN_IGNORED file was ignored/is gone (no more events are delivered)
IN_ONESHOT only one event was generated
```

`$event->IN_xxx`
: Returns a boolean that returns true if the event mask contains any events specified by the mask. All of the `IN_xxx` constants can be used as methods.

`$event->cookie`
`$event->{cookie}`
: The event cookie to ''synchronize two events''. Normally zero, this value is set when two events relating to the same file are generated. As far as I know, this only happens for `IN_MOVED_FROM` and `IN_MOVED_TO` events, to identify the old and new name of a file.

The Linux::Inotify2::Watch Class

Watcher objects are created by calling the `watch` method of a notifier.

It has the following members and methods:

`$watch->name`
`$watch->{name}`
: The name as specified in the `watch` call. For the object itself, this is the empty string. For directory watches, this is the name of the entry without leading path elements.

`$watch->mask`
`$watch->{mask}`
: The mask as specified in the `watch` call.

`$watch->cb ([new callback])`
`$watch->{cb}`
: The callback as specified in the `watch` call. Can optionally be changed.

`$watch->cancel`
: Cancels/removes this watcher. Future events, even if already queued queued, will not be handled and resources will be freed.

SEE ALSO

AnyEvent, Linux::Inotify.

AUTHOR

Marc Lehmann <schmorp@schmorp.de>
http://home.schmorp.de/