NAME

Linux::Epoll – O(1) multiplexing for Linux

VERSION

version 0.016

SYNOPSIS

```
use Linux::Epoll;

my $epoll = Linux::Epoll->new();
$epoll->add($fh, 'in', sub {
    my $events = shift;
    do_something($fh) if $events->{in};
});
$epoll->wait while 1;
```

DESCRIPTION

Epoll is a multiplexing mechanism that scales up O(1) with number of watched files. Linux::Epoll is a callback style epoll module, unlike other epoll modules available on CPAN.

Types of events

- in

  The associated filehandle is available for reading.

- out

  The associated filehandle is available for writing.

- err

  An error condition has happened on the associated filehandle. `wait` will always wait on this event, it is not necessary to set this with `add` or `modify`.

- prio

  There is urgent data available for reading.

- et

  Set edge triggered behavior for the associated filehandle. The default behavior is level triggered. See you **epoll** (7) documentation for more information on what this means.

  Take note that when using edge triggered events, exceptions out of the wait can easily cause events to be lost (when consuming to more than one event at a time).

- hup

  A hang-up has happened on the associated filehandle. `wait` will always wait on this event, it is not necessary to set this with `add` or `modify`.

- rdhup

  Stream socket peer closed the connection, or shut down the writing half of connection. This flag is especially useful for writing simple code to detect peer shutdown when using Edge Triggered monitoring.

- oneshot

  Sets the one-shot behavior for the associated file descriptor. This means that after an event is pulled out with `wait` the associated file descriptor is internally disabled and no other events will be reported by the epoll interface. The user must call `modify` to rearm the file descriptor with a new event mask.

- wakeup

  If `oneshot` and `et` are clear and the process has the `CAP_BLOCK_SUSPEND` capability, ensure that the system does not enter ‘‘suspend’’ or ‘‘hibernate’’ while this event is pending or being processed.

The event is considered as being "processed" from the time when it is returned by a call to **epoll_wait** (2) until the next call to **epoll_wait** (2) on the same **epoll** (7) file descriptor, the closure of that file descriptor, the removal of the event file descriptor with EPOLL_CTL_DEL, or the clearing of EPOLLWAKEUP for the event file descriptor with EPOLL_CTL_MOD.

- exclusive

  Sets an exclusive wakeup mode for the epoll file descriptor that is being attached to the target file descriptor, fd. When a wakeup event occurs and multiple epoll file descriptors are attached to the same target file using `exclusive`, one or more of the epoll file descriptors will receive an event with `wait()`. The default in this scenario (when `exclusive` is not set) is for all epoll file descriptors to receive an event. `exclusive` is thus useful for avoiding thundering herd problems in certain scenarios.

  If the same file descriptor is in multiple epoll instances, some with the `exclusive` flag, and others without, then events will be provided to all epoll instances that did not specify `exclusive`, and at least one of the epoll instances that did specify `exclusive`.

  The following values may be specified in conjunction with `exclusive`: `in`, `out`, `wakeup`, and `et`. `hup` and `err` can also be specified, but this is not required: as usual, these events are always reported if they occur, regardless of whether they are specified in events. Attempts to specify other values in events yield an error. `exclusive` may be used only in an `add()` operation; attempts to employ it with `modify` yield an error. If `exclusive` has been set using `add()`, then a subsequent `modify()` on the same epfd, fd pair yields an error. A call to `add(` that specifies `exclusive` in events and specifies the target file descriptor fd as an epoll instance will likewise fail. The error in all of these cases is `EINVAL`.

METHODS

new()
: Create a new epoll instance.

add($fh, `$events`**,** `$callback`**)**
: Register the filehandle with the epoll instance and associate events `$events` and callback `$callback` with it. `$events` may be either a string (e.g. `'in'`) or an arrayref (e.g. `[qw/in out hup/]`). If a filehandle already exists in the set and `add` is called in non-void context, it returns undef and sets `$!` to `EEXIST`; if the file can't be waited upon it sets `$!` to `EPERM` instead. On all other error conditions an exception is thrown. The callback gets a single argument, a hashref whose keys are the triggered events.

modify($fh, `$events`**,** `$callback`**)**
: Change the events and callback associated on this epoll instance with filehandle `$fh`. The arguments work the same as with `add`. If a filehandle doesn't exist in the set and `modify` is called in non-void context, it returns undef and sets `$!` to `ENOENT`. On all other error conditions an exception is thrown.

delete($fh)
: Remove a filehandle from the epoll instance. If a filehandle doesn't exist in the set and `delete` is called in non-void context, it returns undef and sets `$!` to `ENOENT`. On all other error conditions an exception is thrown.

wait($number = 1, `$timeout` **= undef,** `$sigmask` **= undef)**
: Wait for up to `$number` events, where `$number` must be greater than zero. `$timeout` is the maximal time `wait` will wait for events in fractional seconds. If it is undefined it may wait indefinitely. `$sigmask` is the signal mask during the call. If it is not defined the signal mask will be untouched. If interrupted by a signal it returns undef/an empty list and sets `$!` to `EINTR`. On all other error conditions an exception is thrown.

REQUIREMENTS

This module requires at least Perl 5.10 and Linux 2.6.19 to function correctly.

SEE ALSO

- IO::Epoll

- Sys::Syscall
- IO::Poll

AUTHOR

Leon Timmermans <leont@cpan.org>

COPYRIGHT AND LICENSE