NAME

"IO::Async::Loop::Epoll" – use "IO::Async" with "epoll" on Linux

SYNOPSIS

```
use IO::Async::Loop::Epoll;

use IO::Async::Stream;
use IO::Async::Signal;

my $loop = IO::Async::Loop::Epoll->new();

$loop->add( IO::Async::Stream->new(
      read_handle => \*STDIN,
      on_read => sub {
         my ( $self, $buffref ) = @_;
         while( $$buffref =~ s/^(.*)\r?\n// ) {
            print "You said: $1\n";
         }
      },
) );

$loop->add( IO::Async::Signal->new(
      name => 'INT',
      on_receipt => sub {
         print "SIGINT, will now quit\n";
         $loop->loop_stop;
      },
) );

$loop->loop_forever();
```

DESCRIPTION

This subclass of IO::Async::Loop uses `epoll(7)` on Linux to perform read-ready and write-ready tests so that the O(1) high-performance multiplexing of Linux's `epoll_pwait(2)` syscall can be used.

The `epoll` Linux subsystem uses a persistent registration system, meaning that better performance can be achieved in programs using a large number of filehandles. Each `epoll_pwait(2)` syscall only has an overhead proportional to the number of ready filehandles, rather than the total number being watched. For more detail, see the `epoll(7)` manpage.

This class uses the `epoll_pwait(2)` system call, which atomically switches the process's signal mask, performs a wait exactly as `epoll_wait(2)` would, then switches it back. This allows a process to block the signals it cares about, but switch in an empty signal mask during the poll, allowing it to handle file IO and signals concurrently.

CONSTRUCTOR

new

```
$loop = IO::Async::Loop::Epoll->new()
```

This function returns a new instance of a `IO::Async::Loop::Epoll` object.

METHODS

As this is a subclass of IO::Async::Loop, all of its methods are inherited. Expect where noted below, all of the class's methods behave identically to `IO::Async::Loop`.

loop_once

```
$count = $loop->loop_once( $timeout )
```

This method calls `epoll_pwait(2)`, and processes the results of that call. It returns the total number of `IO::Async::Notifier` callbacks invoked, or `undef` if the underlying `epoll_pwait()` method returned an error. If the `epoll_pwait()` was interrupted by a signal, then 0 is returned instead.

SEE ALSO

- Linux::Epoll − O(1) multiplexing for Linux
- IO::Async::Loop::Poll − use IO::Async with **poll** (2)

AUTHOR

Paul Evans <leonerd@leonerd.org.uk>