## NAME

AnyEvent::Loop – AnyEvent's Pure–Perl event loop

## SYNOPSIS

```
use AnyEvent;
# use AnyEvent::Loop;

# this module gets loaded automatically when no other loop can be found

# Explicit use:
use AnyEvent::Loop;
use AnyEvent;

...

AnyEvent::Loop::run; # run the event loop
```

## DESCRIPTION

This module provides an event loop for AnyEvent in case no other event loop could be found or loaded. You don't have to do anything to make it work with AnyEvent except by possibly loading it before creating the first AnyEvent watcher.

This module is *not* some loop abstracion used by AnyEvent, but just another event loop like EV or Glib, just written in pure perl and delivered with AnyEvent, so AnyEvent always works, even in the absence of any other backend.

If you want to use this module instead of autoloading a potentially better event loop you can simply load it (and no other event loops) before creating the first watcher.

As for performance, this module is on par with (and usually faster than) most select/poll–based C event modules such as Event or Glib (it does not even come close to EV, though), with respect to I/O watchers. Timers are handled less optimally, but for many common tasks, it is still on par with event loops written in C.

This event loop has been optimised for the following use cases:

monotonic clock is available
: This module will use the POSIX monotonic clock option (if it can be detected at runtime) or the POSIX `times` function (if the resolution is at least 100Hz), in which case it will not suffer adversely from time jumps.

  If no monotonic clock is available, this module will not attempt to correct for time jumps in any way.

  The clock chosen will be reported if the environment variable `$PERL_ANYEVENT_VERBOSE` is set to 8 or higher.

any number of watchers on one fd
: Supporting a large number of watchers per fd is purely a dirty benchmark optimisation not relevant in practise. The more common case of having one watcher per fd/poll combo is special-cased, however, and therefore fast, too.

relatively few active fds per `select` call
: This module expects that only a tiny amount of fds is active at any one time. This is relatively typical of larger servers (but not the case where `select` traditionally is fast), at the expense of the ''dense activity case'' where most of the fds are active (which suits `select`).

  The optimal implementation of the ''dense'' case is not much faster, though, so the module should behave very well in most cases, subject to the bad scalability of `select` in the presence of a large number of inactive file descriptors.

lots of timer changes/iteration, or none at all

This module sorts the timer list using perl's `sort`, even though a total ordering is not required for timers internally.

This sorting is expensive, but means sorting can be avoided unless the timer list has changed in a way that requires a new sort.

This means that adding lots of timers is very efficient, as well as not changing the timers. Advancing timers (e.g. recreating a timeout watcher on activity) is also relatively efficient, for example, if you have a large number of timeout watchers that time out after 10 seconds, then the timer list will be sorted only once every 10 seconds.

This should not have much of an impact unless you have hundreds or thousands of timers, though, or your timers have very small timeouts.

## FUNCTIONS

The only user-visible functions provided by this module loop related − watchers are created via the normal AnyEvent mechanisms.

AnyEvent::Loop::run

Run the event loop, usually the last thing done in the main program when you want to use the pure-perl backend.

AnyEvent::Loop::one_event

Blocks until at least one new event has been received by the operating system, whether or not it was AnyEvent-related.

## SEE ALSO

AnyEvent.

## AUTHOR

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