NAME

AnyEvent::Impl::IOAsync − AnyEvent adaptor for IO::Async

SYNOPSIS

```
use AnyEvent;
use IO::Async::Loop;

# optionally set another event loop
use AnyEvent::Impl::IOAsync;
my $loop = new IO::Async::Loop;
AnyEvent::Impl::IOAsync::set_loop $loop;
```

DESCRIPTION

This module provides support for IO::Async as AnyEvent backend. It supports I/O, timers, signals and child process watchers. Idle watchers are emulated. I/O watchers need to dup their fh because IO::Async only supports IO handles, not plain file descriptors.

FUNCTIONS AND VARIABLES

The only user-servicible part in this module is the `set_loop` function and `$LOOP` variable:

AnyEvent::Impl::IOAsync::set_loop `$new_loop`

Unfortunately, IO::Async has no concept of a default loop. Modules using IO::Async must be told by their caller which loop to use, which makes it impossible to transparently use IO::Async from a module.

This module is no exception. It creates a new IO::Async::Loop object when it is loaded. This might not be the right loop object, though, and thus you can replace it by a call to this function with the loop object of your choice.

Note that switching loops while watchers are already initialised can have unexpected effects, and is not supported unless you can live with the consequences.

`$AnyEvent::Impl::IOAsync::LOOP`

This variable always contains the IO::Async::Loop object used by this AnyEvent backend. See above for more info.

Storing the "default" loop makes this module a possible arbiter for other modules that want to use IO::Async transparently. It's advised to directly refer to this variable each time you want to use it, without making a local copy.

PROBLEMS WITH IO::Async

This section had a long list of problems and shortcomings that made it almost impossible to support IO::Async. With version 0.33 of IO::Async, however, most of these have been fixed, so IO::Async can now be used as easily as many other loops.

There are a few remaining problems that require emulation or workarounds:

No support for multiple watchers per event

In most (all? documentation?) cases you cannot have multiple watchers for the same event (what's the point of having all these fancy notifier classes when you cannot have multiple notifiers for the same event? That's like only allowing one timer per second or so...).

For I/O watchers, AnyEvent has to **dup()** every file handle, as IO::Async fails to support the same or different file handles pointing to the same fd (the good thing is that it is documented, but why not fix it instead?).

Apart from these fatal flaws, there are a number of unpleasent properties that just need some mentioning:

Confusing and misleading names

Another rather negative point about this module family is its name, which is deeply confusing: Despite the "async" in the name, IO::Async only does *synchronous* I/O, there is nothing "asynchronous" about it whatsoever (when I first heard about it, I thought, "wow, a second async I/O module, what does it do compared to IO::AIO“, and was somehow set back when I learned that the only ”async" aspect of it is the name).

Inconsistent, incomplete and convoluted API

Implementing AnyEvent's rather simple timers on top of IO::Async's timers was a nightmare (try implementing a timer with configurable interval and delay value...).

The method naming is chaotic: `watch_child` creates a child watcher, but `watch_io` is an internal method; `detach_signal` removes a signal watcher, but `detach_child` forks a subprocess and so on).

Unpleasant surprises on GNU/Linux

When you develop your program on FreeBSD and run it on GNU/Linux, you might have unpleasant surprises, as IO::Async::Loop will by default use IO::Async::Loop::Epoll, which is incompatible with `fork`, so your network server will run into spurious and very hard to debug problems under heavy load, as IO::Async forks a lot of processes, e.g. for DNS resolution. It would be better if IO::Async would only load "safe" backends by default (or fix the epoll backend to work in the presence of fork, which admittedly is hard – EV does it for you, and also does not use unsafe backends by default).

On the positive side, performance with IO::Async is quite good even in my very demanding eyes.

SEE ALSO

AnyEvent, IO::Async.

AUTHOR

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

 Paul Evans <leonerd@leonerd.org.uk>
 Rewrote the backend for IO::Async version 0.33.
```