NAME

AnyEvent::FAQ – frequently asked questions

FAQs

The newest version of this document can be found at <http://pod.tst.eu/http://cvs.schmorp.de/AnyEvent/lib/AnyEvent/FAQ.pod>.

My program exits before doing anything, what's going on?

Programmers new to event-based programming often forget that you can actually do other stuff while ''waiting'' for an event to occur and therefore forget to actually wait when they do not, in fact, have anything else to do.

Here is an example:

```
use AnyEvent;

my $timer = AnyEvent->timer (after => 5, cb => sub { say "hi" });
```

The expectation might be for the program to print ''hi'' after 5 seconds and then probably to exit. However, if you run this, your program will exit almost instantly: Creating the timer does not wait for it, instead the `timer` method returns immediately and perl executes the rest of the program. But there is nothing left to execute, so perl exits.

To force AnyEvent to wait for something, use a condvar:

```
use AnyEvent;

my $quit_program = AnyEvent->condvar;
my $timer = AnyEvent->timer (after => 5, cb => sub { $quit_program->send });

$quit_program->recv;
```

Here the program doesn't immediately exit, because it first waits for the ''quit_program'' condition.

In most cases, your main program should call the event library ''loop'' function directly:

```
use EV;
use AnyEvent;

...

EV::loop;
```

Why is my `tcp_connect` callback never called?

Tricky: `tcp_connect` (and a few other functions in AnyEvent::Socket) is critically sensitive to the caller context.

In void context, it will just do its thing and eventually call the callback. In any other context, however, it will return a special ''guard'' object – when it is destroyed (e.g. when you don't store it but throw it away), tcp_connect will no longer try to connect or call any callbacks.

Often this happens when the `tcp_connect` call is at the end of a function:

```
sub do_connect {
   tcp_connect "www.example.com", 80, sub {
      ... lengthy code
   };
}
```

Then the caller decides whether there is a void context or not. One can avoid these cases by explicitly returning nothing:

```
sub do_connect {
   tcp_connect "www.example.com", 80, sub {
      ... lengthy code
   };

   () # return nothing
}
```

Why do some backends use a lot of CPU in `AE::cv->recv`?

Many people try out this simple program, or its equivalent:

```
use AnyEvent;
AnyEvent->condvar->recv;
```

They are then shocked to see that this basically idles with the Perl backend, but uses 100% CPU with the EV backend, which is supposed to be sooo efficient.

The key to understand this is to understand that the above program is actually *buggy*: Nothing calls `->send` on the condvar, ever. Worse, there are no event watchers whatsoever. Basically, it creates a deadlock: there is no way to make progress, this program doesn't do anything useful, and this will not change in the future: it is already an ex-parrot.

Some backends react to this by freezing, some by idling, and some do a 100% CPU loop.

Since this program is not useful (and behaves as documented with all backends, as AnyEvent makes no CPU time guarantees), this shouldn't be a big deal: as soon as your program actually implements *something*, the CPU usage will be normal.

Why does this FAQ not deal with AnyEvent::Handle questions?

Because AnyEvent::Handle has a NONFAQ on its own that already deals with common issues.

How can I combine Win32::GUI applications with AnyEvent?

Well, not in the same OS thread, that's for sure :) What you can do is create another ithread (or fork) and run AnyEvent inside that thread, or better yet, run all your GUI code in a second ithread.

For example, you could load Win32::GUI and AnyEvent::Util, then create a portable socketpair for GUI->AnyEvent communication.

Then fork/create a new ithread, in there, create a Window and send the `$WINDOW->{-Handle}` to the AnyEvent ithread so it can `PostMessage`.

GUI to AnyEvent communication could work by pushing some data into a Thread::Queue and writing a byte into the socket. The AnyEvent watcher on the other side will then look at the queue.

AnyEvent to GUI communications can also use a Thread::Queue, but to wake up the GUI thread, it would instead use `Win32::GUI::PostMessage $WINDOW, 1030, 0, ""`, and the GUI thread would listen for these messages by using `$WINDOW->Hook (1030 (), sub { ... })`.

My callback dies and...

It must not – part of the contract betwene AnyEvent and user code is that callbacks do not throw exceptions (and don't do even more evil things, such as using `last` outside a loop :). If your callback might die sometimes, you need to use `eval`.

If you want to track down such a case and you can reproduce it, you can enable wrapping (by calling `AnyEvent::Debug::wrap` or by setting `PERL_ANYEVENT_DEBUG_WRAP=1` before starting your program). This will wrap every callback into an eval and will report any exception complete with a backtrace and some information about which watcher died, where it was created and so on.

Author

Marc Lehmann <schmorp@schmorp.de>.