Lightweight Threads in Perl

Andrew Whatson

Thursday 23rd Aug 2012

What they're not

- NOT pre-emptive threads
- NOT kernel-level threads (eg. *pthreads*)
- NOT perl threads (*ithreads*)
- DEFINITELY NOT forks/processes

What they are

- co-operative threads (explicit yielding)
- share a single instance of the interpreter
- one thread active at a time
- similar to python/ruby “green threads”

Why they're useful

- easy to separate concurrent tasks
- simple solution for IO-bound concurrency
- minimal overhead - large *@threads* is fine
- same reasons you'd use *select*, *poll*, *epoll*, *kqueue*, etc.

Lightweight Threads in Perl

- AnyEvent
- Coro
- ... and friends

AnyEvent

- gives primitives for event callbacks
- async sockets, async file handles, timers
- support for many different event-loop backends
- uses *EV* (*libev* as used in *node.js*) by default if available
- includes a pure-perl event loop (useful for debugging)

Example 1: AnyEvent IO

```perl
#!/usr/bin/env perl
use Modern::Perl;
use AnyEvent;

$| = 1; print "enter your name> ";

my $name;
my $ready = AnyEvent->condvar;

my $wait_for_input = AnyEvent->io(
    fh   => \*STDIN,           # the file handle to watch
    poll => 'r',               # watch for read events
    cb   => sub {              # Callback:
      $name = <STDIN>;         #  retrieve a line of input
      chomp $name;             #  clean up that pesky newline
      $ready->send;            #  send the "ready" signal
   }
);

# DO OTHER STUFF

$ready->recv;                  # wait for the "ready" signal
undef $wait_for_input;         # clean up the IO watcher
say "your name is $name";
```

Coro

- gives primitives for lightweight threads
- create and run threads, cede, join
- also includes useful utilities for writing threaded code

Example 2: Simple Coro

```
#!/usr/bin/env perl
use Modern::Perl;
use Coro;

# The "async" thread:
async {
   say 'async 1';
   cede;
   say 'async 2';
};

# The "main" thread:
say 'main 1';
cede;
say 'main 2';
cede;
```

Coro + AnyEvent

- use AnyEvent to generate events
- use *rouse_cb* and *rouse_wait* to make them “blocking”
- write in a procedural style, but with effortless concurrency

Example 3: The Holy Grail?

```
#!/usr/bin/env perl
use Modern::Perl;
use AnyEvent;
use AnyEvent::HTTP   qw(http_get);
use Time::HiRes      qw(time);
use Coro             qw(async rouse_cb rouse_wait);

my $global_start = time;

my @urls = qw(
    http://xkcd.com      http://perlsphere.net    http://news.ycombinator.com
    http://slashdot.org  http://planet6.perl.org  http://reddit.com/r/cyberpunk
);

my @threads = ();
for my $url (@urls) {
    push @threads, async {              # create a new thread for each URL
        my $start = time;
        my $page = get_url($url);
        printf "got %-30s (%fs)\n", $url, time - $start;
    };
}

$_->join for @threads;                  # wait until all threads have finished

printf "got %-30s (%fs)\n", 'everything!', time - $global_start;

sub get_url {
    http_get(shift, rouse_cb);          # cede until ready
    my ($data, $headers) = rouse_wait;  # wait and retrieve results
    return $data;
}
```

AnyEvent + Coro + Moose

- state machines!
- resource managers!
- application controllers!
- lots of room for nice architecture here...

Debugging

- *Coro::Debug* sets up a debug socket
- *socat* the socket to get a REPL in your running code
- trace threads, see "process" trees, etc.

Where to go from here...

- need decent AnyEvent/Coro wrappers for stuff
  - many, many AnyEvent::Foo modules on the CPAN :)
  - eg. *AnyEvent::HTTP* and *AnyEvent::IRC*
- one big want is a decent AnyEvent-enabled DBI
  - options do exist...
  - they're not great for compatibility
  - DBI is one of the "Great Old Ones", difficult to extend
  - probably requires patches to *DBD::Pg*, *DBD::Mysql*, *DBD::Sqlite*, etc.