Scalable Socket I/O

- PG Consultants
- Peter Gordon
- peter@pg-consultants.com

Objective

- To download some http pages
  - Quickly

Start Small – One socket

```
use IO::Socket::INET
my $socket = new IO::Socket::INET(PeerAddr => 'www.yahoo.com',
                                  PeerPort => '80') ;

my $data = "GET / HTTP/1.1\r\nHost: www.yahoo.com\r\n\r\n" ;
print $socket $data ;
while(<$socket>) {
    print $_ ;
}
```

Result

```
HTTP/1.1 302 Found
Date: Wed, 03 Feb 2010 17:15:12 GMT
Location: http://m.www.yahoo.com/
Cache-Control: private
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8

ae
<html><body>The document has moved <a
href='http://www.yahoo.com/'>here</a>.</body></html><!-- f51.us.www.ird.yahoo.com
uncompressed/chunked Wed Feb  3 17:15:12 GMT 2010 -->

0
```

Protocol

- Give a URL
- Open a socket
- Collect the information
- If Document has moved (302)
  - Open a socket
  - Collect the information

Now let's try a few URLs

```
use IO::Socket::INET ;
my @url = qw[www.yahoo.com www.google.com www.perl.org] ;
for my $url (@url) {
    my $socket = new IO::Socket::INET(PeerAddr => $url,
                                      PeerPort => '80') ;
    my $data = "GET / HTTP/1.1\r\nHost: $url\r\n\r\n" ;
    print $socket $data ;
    while(<$socket>) {
        print $_ ;
    }
    }
```

A bit smarter – but not much

```
my $socket = new IO::Socket::INET(PeerAddr => 'www.yahoo.com',
                                  PeerPort => '80',
                                  Blocking => 0) ;
my $select = new IO::Select ;
$select->add($socket) ;
$select->can_write ;
my $data = "GET / HTTP/1.1\r\nHost: www.yahoo.com\r\n\r\n" ;
print $socket $data ;
while(1) {
    my @fh = $select->can_read ;
    if (@fh == 0) { # timeout
        exit(0) ;
    }
    for my $fh (@fh) {
        my $line = <$fh> ;
        if (length($line)) {  #eof
            last ;
        }
        print $line ;
        $lines->{$fh} .= $line ;
    }
}
```

Two sockets

```
my $socket1 = new IO::Socket::INET(PeerAddr => 'www.yahoo.com',PeerPort => '80',Blocking => 0) ;

my $socket2 = new IO::Socket::INET(PeerAddr => 'www.google.com',PeerPort => '80',Blocking => 0) ;

my $select = new IO::Select ;

$select->add($socket1) ;

$select->add($socket2) ;

# Event loop

while(1) {

        @fh = $select->can_write ;

        for (@fh) { write the data ..... }

         @fh = $select->can_read ;

        for (@fh) { read the data ....}

         … intertwined logic for each socket...

}
```

IO::Lambda

DESCRIPTION

This module is another attempt to fight the horrors of non-blocking I/O. It tries to bring back the simplicity of the declarative programming style, that is only available when one employs threads, coroutines, or co-processes. **Usually coding non-blocking I/O for single process, single thread programs requires construction of state machines,** often fairly complex, which fact doesn't help the code clarity, and is the reason why the asynchronous I/O programming is often considered 'messy'. IO::Lambda allows writing I/O callbacks in a style that resembles the good old sequential, declarative programming.

IO::Lambda Classes

IO::Lambda	non-blocking I/O as lambda calculus
IO::Lambda::Backtrace	backtrace chains of events
IO::Lambda::DBI	asynchronous DBI
IO::Lambda::DNS	DNS queries lambda style
IO::Lambda::Flock	lambda-style file locking
IO::Lambda::Fork	wait for blocking code in children processes
IO::Lambda::HTTP	http requests lambda style
IO::Lambda::HTTP::Authen::NTLM	library for enabling NTLM authentication in IO::Lambda::HTTP
IO::Lambda::HTTP::Authen::Negotiate	
IO::Lambda::HTTP::HTTPS	https requests lambda style
IO::Lambda::Loop::AnyEvent	AnyEvent event loop for IO::Lambda
IO::Lambda::Loop::Prima	Prima-based event loop for IO::Lambda
IO::Lambda::Loop::Select	select(2)-based event loop for IO::Lambda
IO::Lambda::Message	message passing queue
IO::Lambda::Mutex	wait for a shared resource
IO::Lambda::Poll	emulate asynchronous behavior by polling
IO::Lambda::SNMP	snmp requests lambda style
IO::Lambda::Signal	wait for pids and signals
IO::Lambda::Socket	wrapper condition for socket functions
IO::Lambda::Thread	wait for blocking code using threads

DNS

```
use IO::Lambda qw(:all);
use IO::Lambda::DNS qw(:all);
sub http {
    my $host = shift ;
    lambda {
        context $host, timeout => 10 ;
        dns {
            my $ip = shift ;
            print "$host $ip\n" ;
        } ;  } }
http('www.google.com')->wait ;
```

DNS - Multiple

```
use IO::Lambda qw(:all);
use IO::Lambda::DNS qw(:all);
sub http {
    my $host = shift ;
    lambda {
       context $host, timeout => 10 ;
       dns {
           my $ip = shift ;
           return $ip ;
       } ;
    }
}

my @hosts = ('www.perl.com', 'www.google.com');

lambda {
    my @funcs ;
    push @funcs, http($_) for (@hosts) ;
    context @funcs ;
    tails {
       print "$_\n" for (@_) ;
    } ;
}-> wait;
```

HTTP

```
sub http {
    my $host = shift ;
    lambda {
        print "Search for $host\n" ;
        context $host, timeout => 2 ;
        dns {
            my $ip = shift ;

            my $socket = IO::Socket::INET-> new(PeerAddr  => $ip,
                                               PeerPort  => 80,
                                               Blocking  => 0) ;
            context $socket, 2  ;
            writable {
                my $data = "GET / HTTP/1.1\r\nHost: $host\r\n\r\n" ;
                print $socket $data ;
                context $socket, 2  ;
                my $buff ;
                readable {
                    my $tempBuff ;
                    my $n = sysread($socket, $tempBuff, 500);
                    $buff .= $tempBuff ;
                    if ($n == 0) {
                        return $buff ;
                    }
                    return again ;
                }}}}
}
```

```
my @hosts = ('www.perl.com',
'www.google.com');

lambda {
    my @funcs ;
    for my $host (@hosts) {
        push @funcs, http($host) ;
    }
    context @funcs ;
    tails {
        my @result = @_ ;
        for (@result) {
            print "$_\n" ;
        }
    } ;
}-> wait;
```

SMTP

```
sub smtp {
   my $host = shift ;
   lambda {
        print "Search for $host\n" ;
        context $host, timeout => 2 ;
        dns {
           my $ip = shift ;

           my $socket = IO::Socket::INET-> new(PeerAddr  => $ip,  PeerPort  => 25, Blocking  => 0) ;
           my $buf ;
           my $count = 0 ;
           context getline, $socket, \$buf;
           tail {
                my $line = shift ;
                $count ++ ;
                return again if $count != 3 ;
                print $socket "HELO Peter\r\n" ;
                context getline, $socket, \$buf;
                tail {
                   print $socket "MAIL FROM: peter\@aaa.com\r\n" ;
                   context getline, $socket, \$buf;
                   tail {
                        print $socket "RCPT TO: peter\@pg-consultants.com\r\n" ;
                        context getline, $socket, \$buf;
                        tail {
                           print $socket "DATA\r\n" ;
                           tail {
                                print $socket "From: Peter\r\nSubject TEST\r\n\r\nHELLO WORLD\r\n" ;
                                print $socket ".\r\n" ;
                                context getline, $socket, \$buf;
                                tail {
                                   my $line = shift ;
                                   print "$line\n" ;
}}}}}}}}

my @hosts = ('mail.pg-consultants.com') ;
```