### When sending a large request to an unresponsive server, Net::HTTP can hang pretty much forever.

```ruby
# server.rb
require 'socket'
server = TCPServer.new('localhost', 2345)
loop do
  socket = server.accept
end

# client.rb
require 'net/http'
connection = Net::HTTP.new('localhost', 2345)
connection.open_timeout = 1
connection.read_timeout = 3
connection.start

post = Net::HTTP::Post.new('/', body = ('a' * 1023) + '
') * 5_000
post.body = body
puts "Sending #{(body.bytesize) bytes}"
connection.request(post)
```

The above code will hang forever on all systems I tested it on (OSX / Linux 3.19).

The issue only triggers once the request body is above a certain threshold. That threshold depends on the system, I assume it's due to the system's TCP settings, but a request over 4MB will trigger the issue consistently.

I assume it happens when the request is bigger than the socket buffer.

It's stuck on the following path:

```
/net/protocol.rb:211:in `write': Interrupt
/net/protocol.rb:185:in `block in write'
/net/protocol.rb:202:in `writing'
/net/protocol.rb:184:in `write'
/net/http/generic_request.rb:188:in `send_request_with_body'
/net/http/generic_request.rb:121:in `exec'
/net/http.rb:1435:in `block in transport_request'
/net/http.rb:1434:in `catch'
/net/http.rb:1434:in `transport_request'
/net/http.rb:1407:in `request'
```

I tried setting setsockopt(Socket::SOL_SOCKET, Socket::SO_SNDTIMEO, ...) on the client socket, but without success. However, adding a Timeout.timeout call around req.exec did work.
Revision 63587 - 06/06/2018 08:03 AM - naruse (Yui NARUSE)
Introduce write_timeout to Net::HTTP [Feature #13396]

Revision 63587 - 06/06/2018 08:03 AM - naruse (Yui NARUSE)
Introduce write_timeout to Net::HTTP [Feature #13396]

History
#1 - 04/04/2017 09:44 AM - byroot (Jean Boussier)
I submitted a pull request to solve this issue: https://github.com/ruby/ruby/pull/1575

#2 - 04/04/2017 06:08 PM - normalperson (Eric Wong)
jean.boussier@gmail.com wrote:

Issue #13396 has been updated by byroot (Jean Boussier).

I submitted a pull request to solve this issue: https://github.com/ruby/ruby/pull/1575

Thanks, I think having a write timeout for Net::HTTP is long overdue.

I'm not approved to make public API changes but would like it accepted.

I would also like to add native timeout support to IO.copy_stream (or improve stdlib timeout support for safety by implementing natively).

Note: I checked your commit c789ab8df5c8d6e0643c481f748014f0066345 by having a "fetch = +refs/pull/..." line in a "remote" section of my .git/config. I did not use any proprietary API or JavaScript to view your changes.

I tried setting setsockopt(Socket::SOL_SOCKET, Socket::SO_SNDTIMEO, ...) on the client socket, but without success. However adding a Timeout.timeout call around req.exec did work.

Under Linux, SO_SNDTIMEO won't work if the the socket was set to nonblocking which would cause Ruby to wait with ppoll (select on other platforms).

#3 - 04/13/2017 05:42 AM - shyouhei (Shyouhei Urabe)
- Tracker changed from Bug to Feature

Let me turn it to be a feature request. Will ask matz about it.

#4 - 05/17/2017 10:45 AM - byroot (Jean Boussier)

I would also like to add native timeout support to IO.copy_stream

That would be the best indeed.

#5 - 05/20/2017 06:48 PM - naruse (Yui NARUSE)
- Assignee set to naruse (Yui NARUSE)
- Status changed from Open to Assigned

The concept Net::HTTP#write_timeout sounds fine.

However adding a Timeout.timeout call around req.exec did work.

Timeout.timeout should be avoided because it sometimes caused trouble and now eliminated.
(see also https://github.com/ruby/ruby/pull/899)
I would also like to add native timeout support to IO.copy_stream

Agree.

Note: I checked your commit c789ab8df5c8d6e0643ccd481f74801400066345

You may know, https://github.com/ruby/ruby/pull/1575.patch and https://github.com/ruby/ruby/pull/1575.diff are also available. They are written as LINK rel="alternate" in the HTML.

I tried setting setsockopt(Socket::SOL_SOCKET, Socket::SO_SNDTIMEO, ...) on the client socket, but without success. However adding a
Timeout.timeout call around req.exec did work.

Under Linux, SO_SNDTIMEO won't work if the the socket was set to
nonblocking which would cause Ruby to wait with ppoll (select on other platforms).

Like this?
Maybe though IO#write seems to be kept as is.

```ruby
@@ -1340,6 +1348,17 @@
   n -= r;
   errno = EAGAIN;
 }
+ifdef SO_SNDTIMEO
 + if (errno == EAGAIN && is_socket(fptr->fd, fptr->pathv)) { +/* Even if EAGAIN, it may be caused by timeout */+
 + struct timeval timeout;
 + socklen_t sz = sizeof(timeout);
 + if (getsockopt(fd, SOL_SOCKET, SO_SNDTIMEO, &timeout, &sz) == 0 &&
 +    timeout.tv_sec != 0 || timeout.tv_usec != 0) { +    return r;
 + }
 + }
+endif
 if (r == -2L)
   return -1L;
 if (rb_io_wait_writable(fptr->fd)) {
```

#6 - 05/20/2017 10:01 PM - naruse (Yui NARUSE)

Only supports Linux:

```ruby
@@ -10372,11 +10383,23 @@
 static int
 nogvl_wait_for_single_fd(int fd, short events)
 {
-    return poll(&fds, 1, -1);
+    if (is_socket(fd, Qnil)) { +      struct timeval timeout;
+      socklen_t sz = sizeof(timeout);
+      if (getsockopt(fd, SOL_SOCKET, SO_SNDTIMEO, &timeout, &sz) == 0 &&
+          timeout.tv_sec != 0 || timeout.tv_usec != 0) { +          ts.tv_sec = timeout.tv_sec;
+          ts.tv_nsec = timeout.tv_usec * 1000;
+          tsp = &ts;
+      }
+    }
+    return ppoll(&fds, 1, tsp, NULL);
 }

 static int
@@ -10445,6 +10468,9 @@
 nogvl_copy_stream_wait_write(struct copy_stream_struct *stp)
 #endif
 } while (ret == -1 && maygvl_copy_stream_continue_p(0, stp));
+    if (ret == 0) {
```
Only supports Linux:

...And only with sockets where O_NONBLOCK is not set, because
the O_NONBLOCK flag will ignore SO_*TIMEO.

Anyways, I don't think using SO_*TIMEO for Ruby is worth it.

I think Ruby can gain an internal timeout mechanism which can
hook into all rb_wait APIs and raise Timeout::TimeoutError as
appropriate.

I will consider that for my work-in-progress on auto-fibers;
all APIs will have a timeout arg: [ruby-core:81244]

I just noticed that just use write_nonblock can solve this ticket:

```ruby
diff --git a/lib/net/protocol.rb b/lib/net/protocol.rb
index 7ec636b384..2a806caeb6 100644
--- a/lib/net/protocol.rb
+++ b/lib/net/protocol.rb
@@ -77,6 +77,12 @@ class OpenTimeout
           < Timeout::Error; end
     class ReadTimeout
           < Timeout::Error; end
+    class WriteTimeout
           < Timeout::Error; end
+    class BufferedIO #:nodoc: internal use only
+      def initialize(io, read_timeout: 60, continue_timeout: nil, debug_output: nil)
@@ -237,9 +243,32 @@ def writing
           len = @io.write_nonblock(*strs, exception: false)
           case len
           when Integer
+             orig_len = len
+             strs.each_with_index do |str, i|
+               len -= str.bytesize
+               if len == 0
+                 if strs.size == i+1
+                   @written_bytes += orig_len
+                   return orig_len
+                 else
+                   strs = strs[i+1..] # rest
+                   break
+                 end
+               elsif len < 0
+                 strs = strs[i..] # str and rest
+                 strs[0] = str[len, -len]
+                 break
+               else # len > 0
+                 # next
+               end
+           end
+           # continue looping
```
when :wait_writable
  @io.to_io.wait_writable(@write_timeout) or raise Net::ReadTimeout
  # continue looping
end

apply in changeset trunk/r63587.

Introduce write_timeout to Net::HTTP [Feature #13396]

#10 - 06/06/2018 09:04 AM - normalperson (Eric Wong)
naruse@ruby-lang.org wrote:

For OpenSSL, I think you need to expect :wait_readable on
write_nonblock, too.

Anyways, the code for handling partial write_nonblock case is verbose.
One day, I would like to:
1) integrate Timeout into core
2) make all SOCK_STREAM sockets non-blocking by default
3) Make rb_wait_for_single_fd aware of Timeouts

So we can use:

```
Timeout.timeout(@write_timeout) { @io.write(strs) }
```

And no new background threads get spawned.

P.S.: If Ruby were LGPL-2.1+, I would steal the ccan/timer module which
is optimized for frequently-expiring timers and be done with 1),
already.

#11 - 06/18/2018 11:13 PM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

Anyways, the code for handling partial write_nonblock case is verbose.
One day, I would like to:
1) integrate Timeout into core
2) make all SOCK_STREAM sockets non-blocking by default
3) Make rb_wait_for_single_fd aware of Timeouts

FYI, I’m close to having a patch ready for 1) and 3);
but maybe 3 is optional, even.

So we can use:

```
Timeout.timeout(@write_timeout) { @io.write(strs) }
```

And no new background threads get spawned.

P.S.: If Ruby were LGPL-2.1+, I would steal the ccan/timer module which
is optimized for frequently-expiring timers and be done with 1),
already.

ccan/timer may not be the right tool for the job (more on this
later).

#12 - 06/17/2021 01:33 PM - miguelteixeira (Miguel Teixeira)
There's an open issue on Net::HTTP#write_timeout not working when body_stream is used: https://bugs.ruby-lang.org/issues/17933

I'm just sharing this here for visibility.