Ruby master - Feature #5138

Add nonblocking IO that does not use exceptions for EOF and EWOULDBLOCK

08/02/2011 07:35 AM - wycats (Yehuda Katz)

Status:	Closed
Priority:	Normal
Assignee:	tenderlovemaking (Aaron Patterson)
Target version:	2.6

Description

The current Ruby I/O classes have non-blocking methods (read_nonblock and write_nonblock). These methods will never block, and if they would block, they raise an exception instead (IO::WaitReadable or IO::WaitWritable). In addition, if the IO is at EOF, they raise an EOFError.

These exceptions are raised repeatedly in virtually every use of the non-blocking methods. This patch adds a pair of methods (try_read_nonblock and try_write_nonblock) that have the same semantics as the existing methods, but they return Symbols instead of raising exceptions for these routine cases:

- :read_would_block
- :write_would_block
- :eof

The patch contains updates for IO, StringIO, and OpenSSL. The updates are fully documented and tested.

Related issues:

Related to Ruby master - Feature #4560: [PATCH] lib/net/protocol.rb: avoid ex...	**Assigned**

Associated revisions

Revision 988ca605 - 08/26/2013 10:41 PM - tenderlove

- io.c (io_read_nonblock): support non-blocking reads without raising exceptions. As in: io.read_nonblock(size, exception: false) [ruby-core:38666] [Feature #5138]
- ext/openssl/ossl_ssl.c (ossl_ssl_read_internal): ditto
- ext/stringio/stringio.c (strio_sysread): ditto
- io.c (rb_io_write_nonblock): support non-blocking writes without raising an exception.
- ext/openssl/ossl_ssl.c (ossl_ssl_write_internal): ditto
- test/openssl/test_pair.rb (class OpenSSL): tests
- test/ruby/test_io.rb (class TestIO): ditto
- test/socket/test_nonblock.rb (class TestSocketNonblock): ditto
- test/stringio/test_stringio.rb (class TestStringIO): ditto

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42695 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 42695 - 08/26/2013 10:41 PM - tenderlove

- io.c (io_read_nonblock): support non-blocking reads without raising exceptions. As in: io.read_nonblock(size, exception: false) [ruby-core:38666] [Feature #5138]
- ext/openssl/ossl_ssl.c (ossl_ssl_read_internal): ditto
- ext/stringio/stringio.c (strio_sysread): ditto
- io.c (rb_io_write_nonblock): support non-blocking writes without raising an exception.
- ext/openssl/ossl_ssl.c (ossl_ssl_write_internal): ditto
- test/openssl/test_pair.rb (class OpenSSL): tests
- test/ruby/test_io.rb (class TestIO): ditto
- test/socket/test_nonblock.rb (class TestSocketNonblock): ditto
- test/stringio/test_stringio.rb (class TestStringIO): ditto

Revision 42695 - 08/26/2013 10:41 PM - tenderlovemaking (Aaron Patterson)

- io.c (io_read_nonblock): support non-blocking reads without raising exceptions. As in: io.read_nonblock(size, exception: false) [ruby-core:38666] [Feature #5138]
- ext/openssl/ossl_ssl.c (ossl_ssl_read_internal): ditto
- ext/stringio/stringio.c (strio_sysread): ditto
- io.c (rb_io_write_nonblock): support non-blocking writes without raising an exception.
- ext/openssl/ossl_ssl.c (ossl_ssl_write_internal): ditto
- test/openssl/test_pair.rb (class OpenSSL): tests

- test/ruby/test_io.rb (class TestIO): ditto
- test/socket/test_nonblock.rb (class TestSocketNonblock): ditto
- test/stringio/test_stringio.rb (class TestStringIO): ditto

Revision 42695 - 08/26/2013 10:41 PM - tenderlove

- io.c (io_read_nonblock): support non-blocking reads without raising exceptions. As in: io.read_nonblock(size, exception: false) [ruby-core:38666] [Feature #5138]
- ext/openssl/ossl_ssl.c (ossl_ssl_read_internal): ditto
- ext/stringio/stringio.c (strio_sysread): ditto
- io.c (rb_io_write_nonblock): support non-blocking writes without raising an exception.
- ext/openssl/ossl_ssl.c (ossl_ssl_write_internal): ditto
- test/openssl/test_pair.rb (class OpenSSL): tests
- test/ruby/test_io.rb (class TestIO): ditto
- test/socket/test_nonblock.rb (class TestSocketNonblock): ditto
- test/stringio/test_stringio.rb (class TestStringIO): ditto

Revision 42695 - 08/26/2013 10:41 PM - tenderlove

- io.c (io_read_nonblock): support non-blocking reads without raising exceptions. As in: io.read_nonblock(size, exception: false) [ruby-core:38666] [Feature #5138]
- ext/openssl/ossl_ssl.c (ossl_ssl_read_internal): ditto
- ext/stringio/stringio.c (strio_sysread): ditto
- io.c (rb_io_write_nonblock): support non-blocking writes without raising an exception.
- ext/openssl/ossl_ssl.c (ossl_ssl_write_internal): ditto
- test/openssl/test_pair.rb (class OpenSSL): tests
- test/ruby/test_io.rb (class TestIO): ditto
- test/socket/test_nonblock.rb (class TestSocketNonblock): ditto
- test/stringio/test_stringio.rb (class TestStringIO): ditto

Revision 42695 - 08/26/2013 10:41 PM - tenderlove

- io.c (io_read_nonblock): support non-blocking reads without raising exceptions. As in: io.read_nonblock(size, exception: false) [ruby-core:38666] [Feature #5138]
- ext/openssl/ossl_ssl.c (ossl_ssl_read_internal): ditto
- ext/stringio/stringio.c (strio_sysread): ditto
- io.c (rb_io_write_nonblock): support non-blocking writes without raising an exception.
- ext/openssl/ossl_ssl.c (ossl_ssl_write_internal): ditto
- test/openssl/test_pair.rb (class OpenSSL): tests
- test/ruby/test_io.rb (class TestIO): ditto
- test/socket/test_nonblock.rb (class TestSocketNonblock): ditto
- test/stringio/test_stringio.rb (class TestStringIO): ditto

Revision 42695 - 08/26/2013 10:41 PM - tenderlove

- io.c (io_read_nonblock): support non-blocking reads without raising exceptions. As in: io.read_nonblock(size, exception: false) [ruby-core:38666] [Feature #5138]
- ext/openssl/ossl_ssl.c (ossl_ssl_read_internal): ditto
- ext/stringio/stringio.c (strio_sysread): ditto
- io.c (rb_io_write_nonblock): support non-blocking writes without raising an exception.
- ext/openssl/ossl_ssl.c (ossl_ssl_write_internal): ditto
- test/openssl/test_pair.rb (class OpenSSL): tests
- test/ruby/test_io.rb (class TestIO): ditto
- test/socket/test_nonblock.rb (class TestSocketNonblock): ditto
- test/stringio/test_stringio.rb (class TestStringIO): ditto

History

#1 - 08/02/2011 07:48 AM - shyouhei (Shyouhei Urabe)

Instead of avoiding exceptions I would like to suggest making exceptions lightweight.

"Check those return values every time you call this function" is nothing different from C. I would write my program totally in C if I have to do that way.

#2 - 08/02/2011 07:53 AM - tenderlovemaking (Aaron Patterson)

- ruby -v changed from ruby 1.9.4dev (2011-07-31 trunk 32788) [x86_64-darwin11.0.0] to -

On Tue, Aug 02, 2011 at 07:35:15AM +0900, Yehuda Katz wrote:

> Issue #5138 has been reported by Yehuda Katz.

Bug #5138: Add nonblocking IO that does not use exceptions for EOF and EWOULDBLOCK
http://redmine.ruby-lang.org/issues/5138

Author: Yehuda Katz
Status: Open
Priority: Normal
Assignee: Yukihiro Matsumoto
Category: core
Target version: 1.9.4
ruby -v: ruby 1.9.4dev (2011-07-31 trunk 32788) [x86_64-darwin11.0.0]

The current Ruby I/O classes have non-blocking methods (read_nonblock and write_nonblock). These methods will never block, and if they would block, they raise an exception instead (IO::WaitReadable or IO::WaitWritable). In addition, if the IO is at EOF, they raise an EOFError.

These exceptions are raised repeatedly in virtually every use of the non-blocking methods. This patch adds a pair of methods (try_read_nonblock and try_write_nonblock) that have the same semantics as the existing methods, but they return Symbols instead of raising exceptions for these routine cases:

- :read_would_block
- :write_would_block
- :eof

The patch contains updates for IO, StringIO, and OpenSSL. The updates are fully documented and tested.

This seems very handy for read loops. Always rescuing the exception bothered me. I think this would be very good functionality when writing pure ruby servers.

--
Aaron Patterson
http://tenderlovemaking.com/

#3 - 08/02/2011 08:23 AM - shyouhei (Shyouhei Urabe)

(08/02/2011 07:46 AM), Aaron Patterson wrote:

> This seems very handy for read loops. Always rescuing the exception bothered me. I think this would be very good functionality when writing pure ruby servers.

Well... the request is about nonblocking IO so I assume the "pure ruby server" you are talking about is single threaded. When you write such read loop, you normally use IO.select as follows:

```
read_fds = [ fd1, fd2, ... ]
loop do
   r, w, e = IO.select(read_fds)
   r.each do |f|
     str = f.readpartial        # ..... (*)
   end
end
```

At the point marked with (*), the socket in question is returned from a select call, which means, the call to f.readpartial would not block because there must be something in a socket's buffer.

So when you do a read loop, nothing bothers you, as long as you use readpartial.

#4 - 08/02/2011 08:23 AM - normalperson (Eric Wong)

Yehuda Katz wycats@gmail.com wrote:

> Issue #5138 has been reported by Yehuda Katz.

> Bug #5138: Add nonblocking IO that does not use exceptions for EOF and EWOULDBLOCK
> http://redmine.ruby-lang.org/issues/5138

Btw, I started working on this in
http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/36904

but never heard more encouragement so didn't work on it more...

> These exceptions are raised repeatedly in virtually every use of the
> non-blocking methods. This patch adds a pair of methods
> (try_read_nonblock and try_write_nonblock) that have the same
> semantics as the existing methods, but they return Symbols instead of
> raising exceptions for these routine cases:
>
> - :read_would_block
> - :write_would_block
> - :eof

Why :eof instead of nil? IO#read already returns nil on EOF

> The patch contains updates for IO, StringIO, and OpenSSL. The updates
> are fully documented and tested.

Cool.

The variable name of "kgio" has no context/meaning in your patch,
especially as the years go on. As the creator of the kgio library, I'd
rather the "kgio" name just die if its ideas are merged into Ruby.

--
Eric Wong

#5 - 08/02/2011 08:23 AM - normalperson (Eric Wong)

Urabe Shyouhei shyouhei@ruby-lang.org wrote:

> So when you do a read loop, nothing bothers you, as long as you use
> readpartial.

That use of select + readpartial is unsafe. Spurious wakeup is a
documented behavior of the select() system call, data can be received
but checksums can be incorrect and data is discarded (after process is
woken up from select()).

Calling IO#nread (from io/wait) before IO#readpartial /may/ be OK, but
I'm not sure how portable that is.

--
Eric Wong

#6 - 08/02/2011 08:23 AM - wycats (Yehuda Katz)

Yehuda Katz
Chief Technologist | Strobe
(ph) 718.877.1325

On Mon, Aug 1, 2011 at 4:07 PM, Eric Wong normalperson@yhbt.net wrote:

> Yehuda Katz wycats@gmail.com wrote:
>
> > Issue #5138 has been reported by Yehuda Katz.
> >
> > ---
> >
> > Bug #5138: Add nonblocking IO that does not use exceptions for EOF and
> > EWOULDBLOCK
> > http://redmine.ruby-lang.org/issues/5138
>
> Btw, I started working on this in
> http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/36904
> but never heard more encouragement so didn't work on it more...
>
> > These exceptions are raised repeatedly in virtually every use of the
> > non-blocking methods. This patch adds a pair of methods
> > (try_read_nonblock and try_write_nonblock) that have the same
> > semantics as the existing methods, but they return Symbols instead of
> > raising exceptions for these routine cases:

- :read_would_block
- :write_would_block
- :eof

> Why :eof instead of nil? IO#read already returns nil on EOF

Interesting. I like this and will update the patch.

> The patch contains updates for IO, StringIO, and OpenSSL. The updates
> are fully documented and tested.

> Cool.
>
> The variable name of "kgio" has no context/meaning in your patch,
> especially as the years go on. As the creator of the kgio library, I'd
> rather the "kgio" name just die if its ideas are merged into Ruby.

No problem. It was mostly an homage to your library, but I'll happily change
it to something more semantic.

> --
> Eric Wong

#7 - 08/02/2011 08:23 AM - wycats (Yehuda Katz)

In practice, it is not more verbose. Here's some example code from
net/protocol, and once try_read_nonblock is used.

Existing code:

```
def rbuf_fill
begin
@rbuf << @io.read_nonblock(BUFSIZE)
rescue IO::WaitReadable
return retry if IO.select([@io], nil, nil, @read_timeout)
raise Timeout::Error
rescue IO::WaitWritable
# OpenSSL::Buffering#read_nonblock may fail with IO::WaitWritable.
# http://www.openssl.org/support/faq.html#PROG10
return retry if IO.select(nil, [@io], nil, @read_timeout)
raise Timeout::Error
end
end
```

With try_read_nonblock:

```
def rbuf_fill
case value = @io.try_read_nonblock(BUFSIZE)
when :read_would_block
return rbuf_fill if IO.select([@io], nil, nil, @read_timeout)
raise Timeout::Error
when :write_would_block
return rbuf_fill if IO.select(nil, [@io], nil, @read_timeout)
raise Timeout::Error
when String
@rbuf << value
end
end
```

*As you can see, the control flow logic is almost identical, but we
use Symbols instead of exceptions to manage the flow.*

Yehuda Katz
Chief Technologist | Strobe
(ph) 718.877.1325

On Mon, Aug 1, 2011 at 3:49 PM, Shyouhei Urabe shyouhei@ruby-lang.orgwrote:

> Issue #5138 has been updated by Shyouhei Urabe.

Instead of avoiding exceptions I would like to suggest making exceptions lightweight.

"Check those return values every time you call this function" is nothing different from C. I would write my program totally in C if I have to do

that way.

Bug #5138: Add nonblocking IO that does not use exceptions for EOF and EWOULDBLOCK
http://redmine.ruby-lang.org/issues/5138

Author: Yehuda Katz
Status: Open
Priority: Normal
Assignee: Yukihiro Matsumoto
Category: core
Target version: 1.9.4
ruby -v: ruby 1.9.4dev (2011-07-31 trunk 32788) [x86_64-darwin11.0.0]

The current Ruby I/O classes have non-blocking methods (read_nonblock and write_nonblock). These methods will never block, and if they would block, they raise an exception instead (IO::WaitReadable or IO::WaitWritable). In addition, if the IO is at EOF, they raise an EOFError.

These exceptions are raised repeatedly in virtually every use of the non-blocking methods. This patch adds a pair of methods (try_read_nonblock and try_write_nonblock) that have the same semantics as the existing methods, but they return Symbols instead of raising exceptions for these routine cases:

- :read_would_block
- :write_would_block
- :eof

The patch contains updates for IO, StringIO, and OpenSSL. The updates are fully documented and tested.

--
http://redmine.ruby-lang.org

#8 - 08/02/2011 08:53 AM - shyouhei (Shyouhei Urabe)

(08/02/2011 08:14 AM), Eric Wong wrote:

> Urabe Shyouhei shyouhei@ruby-lang.org wrote:
>
> > So when you do a read loop, nothing bothers you, as long as you use readpartial.
>
> That use of select + readpartial is unsafe.

Unsafe how? readpatial works even without no data on a buffer.

#9 - 08/02/2011 08:53 AM - normalperson (Eric Wong)

Urabe Shyouhei shyouhei@ruby-lang.org wrote:

> (08/02/2011 08:14 AM), Eric Wong wrote:
>
> > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> >
> > > So when you do a read loop, nothing bothers you, as long as you use readpartial.
> >
> > That use of select + readpartial is unsafe.
>
> Unsafe how? readpatial works even without no data on a buffer.

readpartial will block if there's no data readable, potentially freezing
the whole process.

--
Eric Wong

#10 - 08/02/2011 08:53 AM - shyouhei (Shyouhei Urabe)

(08/02/2011 08:35 AM), Eric Wong wrote:

> Urabe Shyouhei shyouhei@ruby-lang.org wrote:
>
> > (08/02/2011 08:14 AM), Eric Wong wrote:
> >
> > > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> > >
> > > > So when you do a read loop, nothing bothers you, as long as you use
> > > > readpartial.
> > >
> > > That use of select + readpartial is unsafe.
> >
> > Unsafe how? readpatial works even without no data on a buffer.
>
> readpartial will block if there's no data readable, potentially freezing
> the whole process.

Yes but that's not catastrophic. The peer side is sending a data anyway.
Checksum incorrect packets are dropped but retransmitted sooner or later.
The process blocks during that retransmission. That won't last so long.

#11 - 08/02/2011 08:53 AM - normalperson (Eric Wong)

Urabe Shyouhei shyouhei@ruby-lang.org wrote:

> (08/02/2011 08:35 AM), Eric Wong wrote:
>
> > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> >
> > > (08/02/2011 08:14 AM), Eric Wong wrote:
> > >
> > > > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> > > >
> > > > > So when you do a read loop, nothing bothers you, as long as you use
> > > > > readpartial.
> > > >
> > > > That use of select + readpartial is unsafe.
> > >
> > > Unsafe how? readpatial works even without no data on a buffer.
> >
> > readpartial will block if there's no data readable, potentially freezing
> > the whole process.
>
> Yes but that's not catastrophic. The peer side is sending a data anyway.
> Checksum incorrect packets are dropped but retransmitted sooner or later.
> The process blocks during that retransmission. That won't last so long.

Malicious clients can take advantage of this to launch a denial-of-service
attack. Also, networks should never be considered reliable and simple
operations can fail or take a long time.

--
Eric Wong

#12 - 08/02/2011 09:02 AM - wycats (Yehuda Katz)

- File try_nonblock.diff added

Here's an updated patch that returns nil for EOF and replaces the kgio variable with no_exceptions. The docs and tests are updated as well.

#13 - 08/02/2011 09:23 AM - shyouhei (Shyouhei Urabe)

(08/02/2011 08:48 AM), Eric Wong wrote:

> Urabe Shyouhei shyouhei@ruby-lang.org wrote:
>
> > (08/02/2011 08:35 AM), Eric Wong wrote:
> >
> > > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> > >
> > > > (08/02/2011 08:14 AM), Eric Wong wrote:
> > > >
> > > > > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> > > > >
> > > > > > So when you do a read loop, nothing bothers you, as long as you use readpartial.
> > > > >
> > > > > That use of select + readpartial is unsafe.
> > > >
> > > > Unsafe how? readpatial works even without no data on a buffer.
> > >
> > > readpartial will block if there's no data readable, potentially freezing the whole process.
> >
> > Yes but that's not catastrophic. The peer side is sending a data anyway. Checksum incorrect packets are dropped but retransmitted sooner or later. The process blocks during that retransmission. That won't last so long.
>
> Malicious clients can take advantage of this to launch a denial-of-service attack

... even when you do a blocking IO. TCP's having problems on malicious clients is a known issue of the protocol I think.

> Also, networks should never be considered reliable and simple operations can fail or take a long time.

is that the problem we are talking about here? Does Yehuda need a DoS-proven read loop? or he just want a fast variant of read_nonblock?

#14 - 08/02/2011 09:23 AM - normalperson (Eric Wong)

Yehuda Katz wycats@gmail.com wrote:

> On Mon, Aug 1, 2011 at 4:07 PM, Eric Wong normalperson@yhbt.net wrote:
>
> > Yehuda Katz wycats@gmail.com wrote:
> >
> > - :read_would_block
> > - :write_would_block
> > - :eof
> >
> > Why :eof instead of nil? IO#read already returns nil on EOF
>
> Interesting. I like this and will update the patch.

I'm also curious about :*would_block vs the :wait*able names used by kgio. I picked :wait_*able for kgio since the IO::Wait*able name is already used by Ruby 1.9.2+ exceptions, so it's less of a jump.

I feel less strongly about these than nil for EOF, though it might help

with porting any kgio-using apps to newer code.

--
Eric Wong

#15 - 08/02/2011 09:29 AM - akr (Akira Tanaka)

2011/8/2 Urabe Shyouhei shyouhei@ruby-lang.org:

> (08/02/2011 08:14 AM), Eric Wong wrote:
>
> > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> >
> > > So when you do a read loop, nothing bothers you, as long as you use readpartial.
> >
> > That use of select + readpartial is unsafe.
>
> Unsafe how? readpatial works even without no data on a buffer.

I know it is not safe for SSL and gzipped stream.

IO.select works on raw sockets, i.e. encrypted/compressed stream.
The readability of the raw socket doesn't mean readability of
the decrypted/uncompressed stream.

So following may block the loop at IO.select or readpartial.

```
read_streams = [ stream1, stream2, ... ]
loop do
# If a stream have buffered data, it may block.
# (The buffer of IO class is handled by IO.select but
# other buffers are ignored.)
r, w, e = IO.select(read_streams)
r.each do |f|
# if stream doesn't send enough chunk, it may block.
str = f.readpartial
end
end
```

read_nonblock can avoid the blocking of the readpartial but
IO.select can still blocks.

```
# not tested.
read_streams = [ stream1, stream2, ... ]
loop do
# If a stream have buffered data, it may block.
# (The buffer of IO class is handled by IO.select but
# other buffers are ignored.)
r, w, e = IO.select(read_streams)
r.each do |f|
begin
str = f.read_nonblock
rescue IO::WaitReadable, IO::WaitWritable
# xxx: busy loop.
end
end
end
```

So the right way to nonblocking polymorphic read is call read_nonblock
first and call IO.select only when IO::WaitReadable or IO::WaitWritable is
raised.
(We can assume the buffer is empty if read_nonblock raise IO::WaitReadable
or IO::WaitWritable.)

```
# not tested.
read_streams = [ stream1, stream2, ... ]
readable = read_streams.dup
wait_readable = []
wait_writable = []
loop do
readable.each {|f|
```

```
begin
str = f.read_nonblock
rescue IO::WaitReadable
readable.delete f
wait_readable << f
rescue IO::WaitWritable
# OpenSSL::Buffering#read_nonblock may raise IO::WaitWritable.
readable.delete f
wait_writable << f
end
}
if readable.empty?
# call IO.select with zero timeout if readable is not empty?
rs, ws = IO.select(wait_readable, wait_writable)
if rs
wait_readable -= rs
readable.concat rs
end
if ws
wait_writable -= ws
readable.concat ws
end
end
end
```

--
Tanaka Akira

#16 - 08/02/2011 09:29 AM - normalperson (Eric Wong)

Urabe Shyouhei shyouhei@ruby-lang.org wrote:

> (08/02/2011 08:48 AM), Eric Wong wrote:
>
> > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> >
> > > (08/02/2011 08:35 AM), Eric Wong wrote:
> > >
> > > > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> > > >
> > > > > (08/02/2011 08:14 AM), Eric Wong wrote:
> > > > >
> > > > > > Urabe Shyouhei shyouhei@ruby-lang.org wrote:
> > > > > >
> > > > > > > So when you do a read loop, nothing bothers you, as long as you use readpartial.
> > > > > >
> > > > > > That use of select + readpartial is unsafe.
> > > > >
> > > > > Unsafe how? readpatial works even without no data on a buffer.
> > > >
> > > > readpartial will block if there's no data readable, potentially freezing the whole process.
> > >
> > > Yes but that's not catastrophic. The peer side is sending a data anyway. Checksum incorrect packets are dropped but retransmitted sooner or later. The process blocks during that retransmission. That won't last so long.
> >
> > Malicious clients can take advantage of this to launch a denial-of-service attack
>
> ... even when you do a blocking IO. TCP's having problems on malicious clients is a known issue of the protocol I think.

Yes, but I think one should be as defensive-as-possible for these things.

> > Also, networks should never be considered reliable and simple operations can fail or take a long time.

> is that the problem we are talking about here? Does Yehuda need a DoS-proven read loop? or he just want a fast variant of read_nonblock?

I don't know what Yehuda needs, but code proliferates and I would hate to see reliance on fragile assumptions badly affect something down the line that went beyond Yehuda's original need.

--
Eric Wong

#17 - 08/02/2011 09:29 AM - wycats (Yehuda Katz)

Yehuda Katz
Chief Technologist | Strobe
(ph) 718.877.1325

On Mon, Aug 1, 2011 at 5:08 PM, Eric Wong normalperson@yhbt.net wrote:

> Yehuda Katz wycats@gmail.com wrote:
>
> > On Mon, Aug 1, 2011 at 4:07 PM, Eric Wong normalperson@yhbt.net wrote:
> >
> > > Yehuda Katz wycats@gmail.com wrote:
> > >
> > > - :read_would_block
> > > - :write_would_block
> > > - :eof
> > >
> > > Why :eof instead of nil? IO#read already returns nil on EOF
> >
> > Interesting. I like this and will update the patch.
>
> I'm also curious about :*would_block vs the :wait*able names used by kgio. I picked :wait_*able for kgio since the IO::Wait*able name is already used by Ruby 1.9.2+ exceptions, so it's less of a jump.

I picked *_would_block based on the C EWOULDBLOCK errno, but I'd be happy to change it to match the Ruby 1.9.x exception names. I personally am a bit confused by those names (not sure what the WaitReadable adjective would mean), but it *is* the name, so I'll gladly go with it.

> I feel less strongly about these than nil for EOF, though it might help with porting any kgio-using apps to newer code.

That's good enough for me.

> --
> Eric Wong

#18 - 08/02/2011 10:13 AM - matz (Yukihiro Matsumoto)

- Tracker changed from Bug to Feature

#19 - 08/02/2011 01:01 PM - wycats (Yehuda Katz)

- File try_nonblock.diff added

Here is a new patch that uses :wait_readable and :wait_writable

#20 - 08/02/2011 05:48 PM - regularfry (Alex Young)

Shyouhei Urabe wrote:

> Instead of avoiding exceptions I would like to suggest making exceptions lightweight.

"Expected" exceptions used for control flow make $DEBUG output *really* noisy. IO is particularly bad at this.

I like this idea.

#21 - 08/03/2011 11:23 AM - normalperson (Eric Wong)

Tanaka Akira akr@fsij.org wrote:

> 2011/8/2 Eric Wong normalperson@yhbt.net:
>
> > That use of select + readpartial is unsafe. Spurious wakeup is a
> > documented behavior of the select() system call, data can be received
> > but checksums can be incorrect and data is discarded (after process is
> > woken up from select()).
>
> If you mean about Linux, it is already fixed (or have workaround) 6 years ago.
> http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/151776
>
> I don't know other example of spurious wakeup.

Maybe not exactly "spurious wakeup", but the data can be already be read
by another thread/process by the time the reader tries to read.
While uncommon for stream sockets since they lack defined atomicity
rules, it's common for UDP or other message-oriented socket protocols
(and even Unix pipes).

accept() + multiple processes sharing a listener is a common example
of this.

Yehuda's patch doesn't include non-raising variants of accept_nonblock
and connect_nonblock, but kgio also includes kgio_tryaccept and
Kgio::Socket.start(addr)[1] that do not raise on EAGAIN/EINPROGRESS.

[1] kgio refuses to do DNS lookup, addr should be pre-packed

--
Eric Wong

#22 - 08/03/2011 11:23 PM - akr (Akira Tanaka)

2011/8/2 Yehuda Katz wycats@gmail.com:

> The current Ruby I/O classes have non-blocking methods (read_nonblock and write_nonblock). These methods will never block, and if they would block, they raise an exception instead (IO::WaitReadable or IO::WaitWritable). In addition, if the IO is at EOF, they raise an EOFError.
>
> These exceptions are raised repeatedly in virtually every use of the non-blocking methods. This patch adds a pair of methods (try_read_nonblock and try_write_nonblock) that have the same semantics as the existing methods, but they return Symbols instead of raising exceptions for these routine cases:

I'm neutral about this proposal.

Although the exceptions are not exceptional,
they are very educational.
People who use *_nonblock immediately find
nonblocking I/O is not a simple replacement of blocking I/O.

I think IO#read* should raise EOFError for consistency.
(There is an exception, IO#read, though.)
--
Tanaka Akira

#23 - 08/04/2011 02:59 AM - tenderlovemaking (Aaron Patterson)

On Wed, Aug 03, 2011 at 11:03:09PM +0900, Tanaka Akira wrote:

> 2011/8/2 Yehuda Katz wycats@gmail.com:
>
> > The current Ruby I/O classes have non-blocking methods (read_nonblock and write_nonblock). These methods will never block, and if they would block, they raise an exception instead (IO::WaitReadable or IO::WaitWritable). In addition, if the IO is at EOF, they raise an EOFError.
> >
> > These exceptions are raised repeatedly in virtually every use of the non-blocking methods. This patch adds a pair of methods

> (try_read_nonblock and try_write_nonblock) that have the same semantics as the existing methods, but they return Symbols instead of raising exceptions for these routine cases:

I'm neutral about this proposal.

Although the exceptions are not exceptional,
they are very educational.

If the exceptions are not exceptional, that makes me wonder why are they exceptions at all? Since these aren't really exceptional cases, I would prefer not to rescue.

As Alex pointed out, these exceptions really make $DEBUG noisy and that also bugs me about the current behavior.

--
Aaron Patterson
http://tenderlovemaking.com/

#24 - 08/04/2011 05:23 AM - akr (Akira Tanaka)

2011/8/4 Aaron Patterson aaron@tenderlovemaking.com:

> If the exceptions are not exceptional, that makes me wonder why are they exceptions at all? Since these aren't really exceptional cases, I would prefer not to rescue.

They are errors defined by OS.
It is usual to map OS errors to exceptions Errno::*.
--
Tanaka Akira

#25 - 08/06/2011 07:24 PM - akr (Akira Tanaka)

2011/8/4 KOSAKI Motohiro kosaki.motohiro@gmail.com:

> but look, datagram_poll() has another 30~ caller. (e.g. raw socket)
>
> IOW, it's only a workaround for udp based broken application. not a fix. If I understand correctly,
> Linux kernel people don't have a plan to fix this issue by in-kernel change because it makes
> performance hurt.
>
> But, of course, if you are only talking about udp applications, you are correct.

I assumed UDP, TCP and Unix socket.
I don't know other sockets well.

select() with blocking read works too well to feel it is broken.
Is it a common sense in the kernel people that such applications are broken?

Apart from that, you picked up raw socket as an example.
Is it mean that raw socket has a limitation which cannot receive a datagram with wrong checksum?
--
Tanaka Akira

#26 - 08/07/2011 06:54 PM - kosaki (Motohiro KOSAKI)

2011/8/6 Tanaka Akira akr@fsij.org:

> 2011/8/4 KOSAKI Motohiro kosaki.motohiro@gmail.com:
>
> > but look, datagram_poll() has another 30~ caller. (e.g. raw socket)
> >
> > IOW, it's only a workaround for udp based broken application. not a fix. If I understand correctly,
> > Linux kernel people don't have a plan to fix this issue by in-kernel change because it makes
> > performance hurt.

> But, of course, if you are only talking about udp applications, you are correct.

I assumed UDP, TCP and Unix socket.
I don't know other sockets well.

select() with blocking read works too well to feel it is broken.
Is it a common sense in the kernel people that such applications are broken?

I don't know which broken or not broken. But they repeatedly refuse to
change this behavior.
Example,

http://groups.google.com/group/kernelarchive/browse_thread/thread/799ec608f1b7ea2d/43d21499059c922a?hl=ja&lnk=gst&q=select+checksum++socket#43d21499059c922a

> Apart from that, you picked up raw socket as an example.
> Is it mean that raw socket has a limitation which cannot receive a datagram
> with wrong checksum?

Yes, it can be blocked if the packet has a wrong checksum.

#27 - 08/30/2011 09:53 AM - normalperson (Eric Wong)

Shyouhei Urabe shyouhei@ruby-lang.org wrote:

> Instead of avoiding exceptions I would like to suggest making
> exceptions lightweight.

The other problem with the exception from *_nonblock is the exceptions
are extended with IO::Wait*able modules. Object#extend causes a VM
state change which expires the inline and method caches.

Expiring the method cache for this is cheap in 1.9.3, but cache
misses are probably still costly for larger apps (I have not
investigated the cost of cache misses, only cache expiry cost).

#28 - 11/08/2011 07:23 AM - normalperson (Eric Wong)

Yehuda Katz wycats@gmail.com wrote:

> Bug #5138: Add nonblocking IO that does not use exceptions for EOF and EWOULDBLOCK

I don't want this issue to get dropped, so I'm summarizing the
discussion so far for the benefit of folks who haven't followed
everything (including discussions in related threads to this one).

- readpartial + IO.select can still block indefinitely due to:
  - user space layers and buffering (SSL, gzip/zlib)
  - spurious wakeup on some socket types/kernels

- Exceptions with the current *_nonblock methods are expensive.
  - extending with IO::Wait* increments the global VM state version
    - This invalidates both the global method cache and inline caches in 1.9.3. Invaliding the caches is itself an inexpensive operation since 1.9.3, but the cost of subsequent cache misses can hurt[1]
    - This can overflow VM state version within a couple of hours/days on 32-bit systems if IO::Wait* exceptions are raised constantly. Overflowing state version may trigger very rare bugs with false-positive cache hits; vm_clear_all_inline_method_cache() is currently unimplemented to deal with this situation. Overflow is not likely to ever be a problem on 64-bit.
  - backtrace generation cost
    - Generating backtrace is more expensive in MRI 1.9/2.x than MRI 1.8, it might be even more expensive in alternative VMs
    - Additional garbage overhead from backtrace strings

- IO::Wait*/Errno::EAGAIN are very common exceptions.
  - Anybody using *_nonblock will need to care for exceptions with a begin/rescue. Changing begin/rescue into the equivalent case statement requires roughly the same amount of code.
  - These exceptions make $DEBUG unnecessarily noisy

[1] - I haven't done extensive benchmarking on method cache/inline cache effectiveness in MRI. More experienced MRI developers should know more about this topic than I do.

#29 - 11/09/2011 01:23 AM - Anonymous

On Tue, Nov 08, 2011 at 06:59:38AM +0900, Eric Wong wrote:

> Yehuda Katz wycats@gmail.com wrote:
>
> > Bug #5138: Add nonblocking IO that does not use exceptions for EOF and EWOULDBLOCK
>
> I don't want this issue to get dropped, so I'm summarizing the discussion so far for the benefit of folks who haven't followed everything (including discussions in related threads to this one).
>
> - readpartial + IO.select can still block indefinitely due to:
>   - user space layers and buffering (SSL, gzip/zlib)
>   - spurious wakeup on some socket types/kernels
> - Exceptions with the current *_nonblock methods are expensive.
>   - extending with IO::Wait* increments the global VM state version
>   - This invalidates both the global method cache and inline caches in 1.9.3. Invaliding the caches is itself an inexpensive operation since 1.9.3, but the cost of subsequent cache misses can hurt[1]
>   - This can overflow VM state version within a couple of hours/days on 32-bit systems if IO::Wait* exceptions are raised constantly. Overflowing state version may trigger very rare bugs with false-positive cache hits; vm_clear_all_inline_method_cache() is currently unimplemented to deal with this situation. Overflow is not likely to ever be a problem on 64-bit.
>   - backtrace generation cost
>   - Generating backtrace is more expensive in MRI 1.9/2.x than MRI 1.8, it might be even more expensive in alternative VMs
>   - Additional garbage overhead from backtrace strings
> - IO::Wait*/Errno::EAGAIN are very common exceptions.
>   - Anybody using *_nonblock will need to care for exceptions with a begin/rescue. Changing begin/rescue into the equivalent case statement requires roughly the same amount of code.
>   - These exceptions make $DEBUG unnecessarily noisy

I spoke with matz about this issue at RubyConf. I *think* he said it was a good feature, but he wanted a different API. I can't remember exactly.

Maybe matz can comment?

--
Aaron Patterson
http://tenderlovemaking.com/

#30 - 11/10/2011 08:23 AM - normalperson (Eric Wong)

Aaron Patterson tenderlove@ruby-lang.org wrote:

> I spoke with matz about this issue at RubyConf. I *think* he said it was a good feature, but he wanted a different API. I can't remember exactly.
>
> Maybe matz can comment?

I should also note there is one major API difference between Yehuda's patch and the original kgio API:

kgio returns the unwritten portion of the String on a partial write and nil on a full write (meaning there's nothing further to write from that buffer). kgio predates String#byteslice (and still needs to support 1.9.2), so I did the byteslice in C to avoid the overhead of doing it in Rubies without String#byteslice.

Yehuda's patch always returns the number of bytes written if anything was written, so his API is more consistent with traditional IO write

methods.

(Of course, kgio also supports libautocork-like behavior nowadays, but
that's a separate issue :>)

#31 - 11/20/2011 02:23 PM - Anonymous

On Tue, Aug 02, 2011 at 07:35:15AM +0900, Yehuda Katz wrote:

> Issue #5138 has been reported by Yehuda Katz.
>
> ---
>
> Bug #5138: Add nonblocking IO that does not use exceptions for EOF and EWOULDBLOCK
> http://redmine.ruby-lang.org/issues/5138
>
> Author: Yehuda Katz
> Status: Open
> Priority: Normal
> Assignee: Yukihiro Matsumoto
> Category: core
> Target version: 1.9.4
> ruby -v: ruby 1.9.4dev (2011-07-31 trunk 32788) [x86_64-darwin11.0.0]
>
> The current Ruby I/O classes have non-blocking methods (read_nonblock and write_nonblock). These methods will never block, and if they would block, they raise an exception instead (IO::WaitReadable or IO::WaitWritable). In addition, if the IO is at EOF, they raise an EOFError.
>
> These exceptions are raised repeatedly in virtually every use of the non-blocking methods. This patch adds a pair of methods (try_read_nonblock and try_write_nonblock) that have the same semantics as the existing methods, but they return Symbols instead of raising exceptions for these routine cases:
>
> - :read_would_block
> - :write_would_block
> - :eof
>
> The patch contains updates for IO, StringIO, and OpenSSL. The updates are fully documented and tested.

Bump. Can someone please give feedback? What needs to change in this
patch before it can be applied?

Thanks!

--
Aaron Patterson
http://tenderlovemaking.com/

#32 - 11/20/2011 03:53 PM - kosaki (Motohiro KOSAKI)

> Bump. Â Can someone please give feedback? Â What needs to change in this
> patch before it can be applied?
>
> Thanks!

Hey, you said matz has an another idea. So, we need to hear it. isn't it?

Personally, my feelings are,

- read_nonblock and try_read_nonblock have too small difference to make separate method. then, instead, I'd prefet to add an keyword argument to read_nonblock.
- I can imagine IO::WaitReadable is annoying to use. but I'm not sure how much affect it a performance. Did anyone mesure it?
- We need doc update about "select + readpartial is unsafe" issue. Can anyone volunteer it?

And, for clarify, I think nobody replay Eric pointed following issue.
Although I'm not sure it is important or not.

- Yehuda's original patch don't accept_nonblock and connect_nonblock.
- current write_nonblock return written byte. but it is unuseful. current String is designed intentionally byte operation unfrinedly.

#33 - 12/15/2011 09:23 AM - normalperson (Eric Wong)

KOSAKI Motohiro kosaki.motohiro@gmail.com wrote:

- I can imagine IO::WaitReadable is annoying to use. but I'm not sure how much affect it a performance. Did anyone mesure it?

I've released a net-wrong RubyGem which monkey patches net/* with
kgio + terrible_timeout:

http://bogomips.org/net-wrong/README

Hopefully people can test (on real applications) and report performance
differences from avoiding exceptions/temporary Classes (and
buffer-recycling)

#34 - 02/20/2012 04:41 AM - headius (Charles Nutter)

I never got to weigh in on this thread back in the day, and I'm working on IO stuff more recently, so a few thoughts...

Eric Wong wrote:

- Exceptions with the current *_nonblock methods are expensive.
  - extending with IO::Wait* increments the global VM state version

This is really poor form. I'm not sure why it hasn't been fixed yet. JRuby master solves this in a neat, compatible way: instead of raising singleton EAGAIN with WaitReable or WaitWritable mixed in, we raise EAGAINReadable or EAGAINWritable, subclasses that have already done the mixing. No other changes are needed.

- backtrace generation cost

In JRuby, EAGAIN does not generate a backtrace unless you pass a flag, since generally it's an "expected" exception and you don't want the backtrace anyway.

How does this translate to perf?

```
system ~/projects $ jruby eagain_bench.rb
1.553000   0.000000   1.553000 (  1.553000)
0.673000   0.000000   0.673000 (  0.673000)
0.667000   0.000000   0.667000 (  0.667000)
0.669000   0.000000   0.669000 (  0.669000)
0.696000   0.000000   0.696000 (  0.697000)

system ~/projects $ rvm 1.9.3 do ruby eagain_bench.rb
1.250000   0.150000   1.400000 (  1.396295)
1.250000   0.150000   1.400000 (  1.408975)
1.260000   0.150000   1.410000 (  1.405947)
1.250000   0.150000   1.400000 (  1.405391)
1.260000   0.150000   1.410000 (  1.407389)

system ~/projects $ rvm ruby-head do ruby eagain_bench.rb
1.310000   0.120000   1.430000 (  1.428627)
1.300000   0.120000   1.420000 (  1.430652)
1.310000   0.130000   1.440000 (  1.426011)
1.310000   0.120000   1.430000 (  1.430642)
1.310000   0.120000   1.430000 (  1.436441)
```

Rather well, I think.

#35 - 02/20/2012 05:19 AM - headius (Charles Nutter)

Worth pointing out that the cost of allocating an exception every time could be blunted by always raising the *same* exception object. This avoids the backtrace, construction cost, *and* mixin overhead in one shot, provided you're ok with the backtrace being meaningless.

FWIW, even all these tricks can't make an exception-based version as fast as one that simply returns nil. Modifying IO#read_nonblock in JRuby to return nil on a zero-length read improves the above benchmark 2-3x.

#36 - 03/16/2012 01:06 PM - headius (Charles Nutter)

Related: http://bugs.ruby-lang.org/issues/6154

#37 - 03/18/2012 06:46 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

#38 - 07/01/2012 12:55 AM - tenderlovemaking (Aaron Patterson)

- File feature5138.pdf added

I've attached a slide for this.

#39 - 07/01/2012 01:29 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Aaron, with due respect, I'm not sure if the examples in your slide are good enough for promoting it. The logic looks like the same in both current and proposed version. Maybe some use cases where the current approach would require too much code would demonstrate better why the new approach is helpful. Unless you're gonna to present the slide yourself or make sure the presenter will discuss the pros and contras of each approach.

#40 - 07/01/2012 02:47 AM - drbrain (Eric Hodel)

The goal is to avoid the creation hundreds (if not thousands) of Errno::EWOULDBLOCK, IO::WaitReadable and IO::WaitWritable exceptions (including their backtraces) that are immediately rescued in the course of a typical non-blocking socket read or write loop.

The usage should be as close to the same as possible to make it easy to modify code that currently uses exceptions. This way users can quickly and easily switch to the new API without spending much time on it.

#41 - 07/01/2012 04:34 AM - tenderlovemaking (Aaron Patterson)

- File feature5138.pdf added

New slide.

#42 - 07/01/2012 04:34 AM - tenderlovemaking (Aaron Patterson)

- File deleted (feature5138.pdf)

#43 - 07/01/2012 04:51 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Great! I'm just not sure if the second slide was meant to be present in the generated PDF :P

#44 - 07/01/2012 04:53 AM - Anonymous

- File noname added

On Sun, Jul 01, 2012 at 02:47:58AM +0900, drbrain (Eric Hodel) wrote:

> Issue #5138 has been updated by drbrain (Eric Hodel).
>
> The goal is to avoid the creation hundreds (if not thousands) of Errno::EWOULDBLOCK, IO::WaitReadable and IO::WaitWritable exceptions (including their backtraces) that are immediately rescued in the course of a typical non-blocking socket read or write loop.
>
> The usage should be as close to the same as possible to make it easy to modify code that currently uses exceptions. This way users can quickly and easily switch to the new API without spending much time on it.

Thanks Eric! I'll add this to the slide. <3<3<3

--
Aaron Patterson
http://tenderlovemaking.com/

#45 - 07/02/2012 01:58 AM - mame (Yusuke Endoh)

Aaron, I received your slide. Thank you!

--
Yusuke Endoh mame@tsg.ne.jp

#46 - 07/23/2012 09:54 PM - mame (Yusuke Endoh)

Yehuda Katz and Aaron Patterson,

We discussed your slide at the developer meeting (7/21), but
cannot reach agreement. Please continue to make a discussion.

Here is a discussion summary.
I hope you find it informative to improve your proposal.

- Matz was positive to the feature itself.
- Matz was NOT positive to the name `try_read_nonblock'.

- Akr suggested a new name convention about IO: read_*' for exception-style methods, and get_*' for non-exception-style methods.
- Matz prefered get_nonblock' totry_read_nonblock'. (but he showed no opinion about the convention itself)

- Matz was pondering whether or not to accept a method returning either Symbol or String.
  - He suggested Erlang-style API, always returning a two-length array whose contains state (Symbol) and data (String).

- Akr was afraid that the code example in the slide was inaccurate.
  - I didn't understand his opinion; please talk with him directly.

--
Yusuke Endoh mame@tsg.ne.jp

#47 - 11/20/2012 11:13 PM - mame (Yusuke Endoh)

- Target version changed from 1.9.4 to 2.6

#48 - 04/24/2013 05:25 AM - headius (Charles Nutter)

Trying to kick this one forward. I'm going to implement this in JRuby, perhaps via an ext (require 'io/try_nonblock' or something). It will give an opportunity to play with the API in-place using JRuby master builds or JRuby 1.7.4+.

Will report some results once I get it implemented.

#49 - 04/24/2013 06:19 AM - headius (Charles Nutter)

JRuby master (1.7.4) now has a new ext io/try_nonblock that implements just the IO portion of wycats's patch (I did not implement the StringIO and SSLSocket logic).

Numbers for each of the three modes (errnos with backtraces (slow due to JVM), errnos without backtraces, and returning symbol):

```
$ jruby -Xerrno.backtrace=true -rbenchmark -rsocket -e "sock = TCPSocket.new('google.com', 80); 10.times { puts Benchmark.measure {
100_000.times { begin; sock.read_nonblock(1000); rescue; end } } }"
12.610000   0.380000  12.990000 ( 11.157000)
9.290000   0.290000   9.580000 (  9.477000)
9.350000   0.300000   9.650000 (  9.464000)
9.260000   0.290000   9.550000 (  9.335000)
9.320000   0.300000   9.620000 (  9.341000)
9.140000   0.290000   9.430000 (  9.218000)
9.150000   0.300000   9.450000 (  9.235000)
9.730000   0.330000  10.060000 (  9.794000)
9.210000   0.290000   9.500000 (  9.275000)
9.340000   0.300000   9.640000 (  9.393000)

$ jruby -rbenchmark -rsocket -e "sock = TCPSocket.new('google.com', 80); 10.times { puts Benchmark.measure { 100_000.times { begin;
sock.read_nonblock(1000); rescue; end } } }"
3.110000   0.310000   3.420000 (  2.065000)
0.690000   0.230000   0.920000 (  0.800000)
0.670000   0.200000   0.870000 (  0.803000)
0.610000   0.220000   0.830000 (  0.798000)
0.630000   0.210000   0.840000 (  0.825000)
0.620000   0.210000   0.830000 (  0.816000)
0.610000   0.200000   0.810000 (  0.809000)
0.620000   0.210000   0.830000 (  0.811000)
0.620000   0.210000   0.830000 (  0.809000)
0.620000   0.210000   0.830000 (  0.819000)

$ jruby -rbenchmark -rsocket -rio/try_nonblock -e "sock = TCPSocket.new('google.com', 80); 10.times { puts Benchmark.measure { 100_000.times {
sock.try_read_nonblock(1000) } } }"
1.150000   0.220000   1.370000 (  0.846000)
0.320000   0.160000   0.480000 (  0.342000)
0.190000   0.180000   0.370000 (  0.346000)
0.220000   0.170000   0.390000 (  0.341000)
0.210000   0.160000   0.370000 (  0.319000)
0.210000   0.160000   0.370000 (  0.322000)
0.170000   0.160000   0.330000 (  0.319000)
0.160000   0.150000   0.310000 (  0.308000)
0.150000   0.150000   0.300000 (  0.309000)
0.170000   0.160000   0.330000 (  0.323000)
```

#50 - 07/10/2013 06:24 AM - tenderlovemaking (Aaron Patterson)

- File nonblock_no_tuple.patch added

- File nonblock_tuple.patch added

=begin
Hi, I've updated the patch to apply against trunk (please find it attached).

Matz, akr, with regard to get_* vs try_read_*, *I don't think it will work. For example, should try_write_* be set_*? I think it would look strange to say "io.set_nonblock(bytes)". We also have sysread_nonblock, should that be sysget_nonblock? Changing the method names is fine, but I don't think "get/set" pair works well.

As for Erlang style return values. It seems interesting, but that means every call to try_read_nonblock would allocate an array. The only possible return values would be:

[bytes, nil] # successful read
[nil, nil] # EOF
[nil, :wait_readable]
[nil, :wait_writable]

In this case it seems easier if we stick with one return value rather than two. I really want this feature, so I've also prepared a patch with the "tuple" solution (please find it attached).

In order to demonstrate a usecase, we can take rbuf_fill from net/http as an example. Please find the current method definition here:

https://github.com/ruby/ruby/blob/07dc8257039f69b41cca50ff49ce738d3df7b362/lib/net/protocol.rb#L151-L169

Here is what it would look like with the tuple method:

def rbuf_fill
loop do
chunk, err = @io.try_read_nonblock(BUFSIZE)

```
  case err
  when :wait_readable
    unless IO.select([@io], nil, nil, @read_timeout)
      raise Net::ReadTimeout
    end
  when :wait_writable
    # OpenSSL::Buffering#read_nonblock may fail with IO::WaitWritable.
    # http://www.openssl.org/support/faq.html#PROG10
    unless IO.select(nil, [@io], nil, @read_timeout)
      raise Net::ReadTimeout
    end
  else
    raise EOFError unless chunk
    @rbuf << chunk
    break
  end
end
```

end

Here is what it looks like with just a single return value:

def rbuf_fill
loop do
chunk = @io.try_read_nonblock(BUFSIZE)

```
  case chunk
  when :wait_readable
    unless IO.select([@io], nil, nil, @read_timeout)
      raise Net::ReadTimeout
    end
  when :wait_writable
    # OpenSSL::Buffering#read_nonblock may fail with IO::WaitWritable.
    # http://www.openssl.org/support/faq.html#PROG10
    unless IO.select(nil, [@io], nil, @read_timeout)
      raise Net::ReadTimeout
    end
  when nil then raise EOFError
  else
    @rbuf << chunk
    break
  end
end
```

end

We can express nonblocking reads with loop rather than using begin/end + an exception and retry as the loop construct.
=end

#51 - 07/10/2013 06:35 AM - tenderlovemaking (Aaron Patterson)

- File deleted (noname)

#52 - 07/10/2013 06:35 AM - tenderlovemaking (Aaron Patterson)

- File deleted (noname)

#53 - 07/10/2013 09:12 AM - tenderlovemaking (Aaron Patterson)

=begin
Just another data point. I ran the benchmarks that Charles showed on my implementation in MRI. We also see a good speed improvement on MRI:

$ ruby -rbenchmark -rsocket -e "sock = TCPSocket.new('google.com', 80); 10.times { puts Benchmark.measure { 100_000.times { begin; sock.read_nonblock(1000); rescue; end } } }"
0.650000 0.110000 0.760000 (0.769794)
0.620000 0.110000 0.730000 (0.721198)
0.600000 0.090000 0.690000 (0.701172)
0.640000 0.110000 0.750000 (0.738934)
0.600000 0.100000 0.700000 (0.702289)
0.600000 0.090000 0.690000 (0.694982)
0.630000 0.110000 0.740000 (0.737876)
0.630000 0.100000 0.730000 (0.729356)
0.640000 0.110000 0.750000 (0.743386)
0.600000 0.100000 0.700000 (0.705391)
$ ruby -rbenchmark -rsocket -e "sock = TCPSocket.new('google.com', 80); 10.times { puts Benchmark.measure { 100_000.times { begin; sock.try_read_nonblock(1000); rescue; end } } }"
0.160000 0.070000 0.230000 (0.230004)
0.150000 0.080000 0.230000 (0.224267)
0.170000 0.080000 0.250000 (0.253598)
0.150000 0.070000 0.220000 (0.221024)
0.150000 0.070000 0.220000 (0.225998)
0.160000 0.070000 0.230000 (0.223979)
0.150000 0.080000 0.230000 (0.231132)
0.160000 0.070000 0.230000 (0.233416)
0.160000 0.080000 0.240000 (0.238810)
0.150000 0.070000 0.220000 (0.222216)

=end

#54 - 07/10/2013 09:23 AM - normalperson (Eric Wong)

"tenderlovemaking (Aaron Patterson)" aaron@tenderlovemaking.com wrote:

> As for Erlang style return values. It seems interesting, but that
> means every call to try_read_nonblock would allocate an array. The
> only possible return values would be:

I prefer to avoid the requirement for allocating anything in the
common case. So "no" to the tuple retvals.

How about making the destination buffer a required argument,
and returning:

```
Integer           - length in bytes on successful read
Symbol            - :wait_readable/:wait_writable (common)
nil               - EOF
```

case ret = io.try_read(maxlen, buffer)
when Integer
process_buffer(buffer, ret)
when nil
break
when Symbol
io.**send**(ret, timeout_sec) # :wait_readable,:wait_writable
end while true

I think making the buffer a required argument makes sense anyways for
performance (not just GC, but keeping the CPU cache hot regardless of

VM/language, too).

#55 - 07/10/2013 09:53 AM - Anonymous

On Wed, Jul 10, 2013 at 09:03:19AM +0900, Eric Wong wrote:

> "tenderlovemaking (Aaron Patterson)" aaron@tenderlovemaking.com wrote:
>
> > As for Erlang style return values. It seems interesting, but that means every call to try_read_nonblock would allocate an array. The only possible return values would be:
>
> I prefer to avoid the requirement for allocating anything in the common case. So "no" to the tuple retvals.
>
> How about making the destination buffer a required argument,

Does this mean someone would have to pre-allocate the buffer? For example, the "buffer" variable below would need to be defined as:

buffer = " " * maxlen

> and returning:
>
> Integer - length in bytes on successful read
> Symbol - :wait_readable/:wait_writable (common)
> nil - EOF

Is this really advantageous over:

```
String              - the buffer read in
Symbol              - :wait_readable/:wait_writable (common)
nil                 - EOF
```

Length of bytes can be derived from buffer length.

> case ret = io.try_read(maxlen, buffer)
> when Integer
> process_buffer(buffer, ret)
> when nil
> break
> when Symbol
> io.**send**(ret, timeout_sec) # :wait_readable,:wait_writable
> end while true
>
> I think making the buffer a required argument makes sense anyways for performance (not just GC, but keeping the CPU cache hot regardless of VM/language, too).

--
Aaron Patterson
http://tenderlovemaking.com/

#56 - 07/10/2013 10:53 AM - normalperson (Eric Wong)

Aaron Patterson tenderlove@ruby-lang.org wrote:

> On Wed, Jul 10, 2013 at 09:03:19AM +0900, Eric Wong wrote:
>
> > "tenderlovemaking (Aaron Patterson)" aaron@tenderlovemaking.com wrote:
> >
> > > As for Erlang style return values. It seems interesting, but that means every call to try_read_nonblock would allocate an array. The only possible return values would be:
> >
> > I prefer to avoid the requirement for allocating anything in the common case. So "no" to the tuple retvals.
> >
> > How about making the destination buffer a required argument,

> Does this mean someone would have to pre-allocate the buffer? For example, the "buffer" variable below would need to be defined as:
>
> buffer = " " * maxlen

Yes, but it could just be: buffer = ""

Ruby will internally resize the string and won't waste memory bandwidth prefilling it with 0x20

> and returning:
>
> Integer - length in bytes on successful read
> Symbol - :wait_readable/:wait_writable (common)
> nil - EOF

> Is this really advantageous over:

```
String              - the buffer read in
Symbol              - :wait_readable/:wait_writable (common)
nil                 - EOF
```

> Length of bytes can be derived from buffer length.

It would force developers to think about buffer reuse (which helps a lot with big copy loops). Otherwise, it won't help for for apps which already reuse buffers.

#57 - 07/12/2013 02:23 AM - Anonymous

On Wed, Jul 10, 2013 at 10:52:26AM +0900, Eric Wong wrote:

> Aaron Patterson tenderlove@ruby-lang.org wrote:
>
> > On Wed, Jul 10, 2013 at 09:03:19AM +0900, Eric Wong wrote:
> >
> > > "tenderlovemaking (Aaron Patterson)" aaron@tenderlovemaking.com wrote:
> > >
> > > > As for Erlang style return values. It seems interesting, but that means every call to try_read_nonblock would allocate an array. The only possible return values would be:
> > >
> > > I prefer to avoid the requirement for allocating anything in the common case. So "no" to the tuple retvals.
> > >
> > > How about making the destination buffer a required argument,
> >
> > Does this mean someone would have to pre-allocate the buffer? For example, the "buffer" variable below would need to be defined as:
> >
> > buffer = " " * maxlen
>
> Yes, but it could just be: buffer = ""
>
> Ruby will internally resize the string and won't waste memory bandwidth prefilling it with 0x20
>
> > and returning:
> >
> > Integer - length in bytes on successful read
> > Symbol - :wait_readable/:wait_writable (common)
> > nil - EOF
>
> Is this really advantageous over:
>
> String - the buffer read in
> Symbol - :wait_readable/:wait_writable (common)

> > nil - EOF
> >
> > Length of bytes can be derived from buffer length.
>
> It would force developers to think about buffer reuse (which helps a lot
> with big copy loops). Otherwise, it won't help for for apps which
> already reuse buffers.

It occurs to me that this is pretty similar to IO#readpartial. The main
difference being that readpartial automatically retries on EWOULDBLOCK
and raises an EOF error. Maybe there should be a try_readpartial in
addition to the proposed try_read_nonblock?

WDYT?

--
Aaron Patterson
http://tenderlovemaking.com/

#58 - 07/12/2013 03:59 AM - normalperson (Eric Wong)

Aaron Patterson tenderlove@ruby-lang.org wrote:

> It occurs to me that this is pretty similar to IO#readpartial. The main
> difference being that readpartial automatically retries on EWOULDBLOCK
> and raises an EOF error. Maybe there should be a try_readpartial in
> addition to the proposed try_read_nonblock?

It's always bothered me that IO#readpartial raises EOFError, too
(especially when IO#read returns nil). So yes, try_readpartial
would be good, but try_*nonblock is much more important since
EAGAIN is more common than EOF.

Also, I also wonder if there's a generic way for us to implement
"expected exceptions":

- without needing to introduce additional methods
- without allocating new objects for common exceptions

Perhaps similar to catch/throw...

Maybe:

```
begin_expect(Errno::ENOENT)
# cause the class Errno::ENOENT to be raised,
# not an instance of Errno::ENOENT
File.open("/possibly/non-existent/file")
rescue Errno::ENOENT => e
e == Errno::ENOENT # note the '==' (not '===')
end
```

Otherwise, we might end up needing try_open, try_stat, try_lstat,
try_link, try_unlink, try_rename, etc to avoid racy/wasteful
File.exist? calls...

#59 - 07/12/2013 05:59 AM - Anonymous

On Fri, Jul 12, 2013 at 03:55:16AM +0900, Eric Wong wrote:

> Aaron Patterson tenderlove@ruby-lang.org wrote:
>
> > It occurs to me that this is pretty similar to IO#readpartial. The main
> > difference being that readpartial automatically retries on EWOULDBLOCK
> > and raises an EOF error. Maybe there should be a try_readpartial in
> > addition to the proposed try_read_nonblock?
>
> It's always bothered me that IO#readpartial raises EOFError, too
> (especially when IO#read returns nil). So yes, try_readpartial
> would be good, but try_*nonblock is much more important since
> EAGAIN is more common than EOF.

Ah, I was thinking that IO#try_readpartial would work exactly the same as try_*nonblock, but with a buffer (no exceptions, no automatic retry).

> Also, I also wonder if there's a generic way for us to implement "expected exceptions":
>
> - without needing to introduce additional methods
> - without allocating new objects for common exceptions
>
> Perhaps similar to catch/throw...
>
> Maybe:
>
> ```
> begin_expect(Errno::ENOENT)
> # cause the class Errno::ENOENT to be raised,
> # not an instance of Errno::ENOENT
> File.open("/possibly/non-existent/file")
> rescue Errno::ENOENT => e
> e == Errno::ENOENT # note the '==' (not '===')
> end
> ```
>
> Otherwise, we might end up needing try_open, try_stat, try_lstat, try_link, try_unlink, try_rename, etc to avoid racy/wasteful File.exist? calls...

I'm not sure. The big thing for me about the try_* methods is that I can use loop as the looping construct, not rescue / retry.

--
Aaron Patterson
http://tenderlovemaking.com/

#60 - 07/26/2013 04:53 PM - tenderlovemaking (Aaron Patterson)

On Tue, Aug 02, 2011 at 07:35:15AM +0900, Yehuda Katz wrote:

> Issue #5138 has been reported by Yehuda Katz.
>
> ---
>
> Bug #5138: Add nonblocking IO that does not use exceptions for EOF and EWOULDBLOCK
> http://redmine.ruby-lang.org/issues/5138
>
> Author: Yehuda Katz
> Status: Open
> Priority: Normal
> Assignee: Yukihiro Matsumoto
> Category: core
> Target version: 1.9.4
> ruby -v: ruby 1.9.4dev (2011-07-31 trunk 32788) [x86_64-darwin11.0.0]
>
> The current Ruby I/O classes have non-blocking methods (read_nonblock and write_nonblock). These methods will never block, and if they would block, they raise an exception instead (IO::WaitReadable or IO::WaitWritable). In addition, if the IO is at EOF, they raise an EOFError.
>
> These exceptions are raised repeatedly in virtually every use of the non-blocking methods. This patch adds a pair of methods (try_read_nonblock and try_write_nonblock) that have the same semantics as the existing methods, but they return Symbols instead of raising exceptions for these routine cases:
>
> - :read_would_block
> - :write_would_block
> - :eof
>
> The patch contains updates for IO, StringIO, and OpenSSL. The updates are fully documented and tested.

In the last developer meeting, matz suggested that the existing nonblocking IO functions take an extra option to specify whether or not exceptions should be raised.

https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20130712#Non-blocking-reads-without-exceptions-Feature-5138

Something like this:

```
chunk = @io.read_nonblock(BUFSIZE, exception: false)
```

I am happy with a solution like this.

--
Aaron Patterson
http://tenderlovemaking.com/

#61 - 07/27/2013 08:23 AM - normalperson (Eric Wong)

Aaron Patterson tenderlove@ruby-lang.org wrote:

> In the last developer meeting, matz suggested that the existing
> nonblocking IO functions take an extra option to specify whether or not
> exceptions should be raised.
>
> https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20130712#Non-blocking-reads-without-exceptions-Feature-5138
>
> Something like this:
>
> ```
> chunk = @io.read_nonblock(BUFSIZE, exception: false)
> ```
>
> I am happy with a solution like this.

OK, I am also happy (especially since small hashes are packed, now).
So the existing outbuf arg also becomes a keyword arg?

```
chunk = @io.read_nonblock(BUFSIZE, exception: false, outbuf: string)
```

#62 - 07/27/2013 01:36 PM - matz (Yukihiro Matsumoto)

- Assignee changed from matz (Yukihiro Matsumoto) to tenderlovemaking (Aaron Patterson)

In the developers' meeting on 2007-07-27, we concluded we accepted the proposal from Aaron in
https://dl.dropboxusercontent.com/u/582984/again_5138.pdf.

Matz.

#63 - 08/27/2013 07:41 AM - tenderlovemaking (Aaron Patterson)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r42695.
Yehuda, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- io.c (io_read_nonblock): support non-blocking reads without raising exceptions. As in: io.read_nonblock(size, exception: false) [ruby-core:38666] [Feature #5138]
- ext/openssl/ossl_ssl.c (ossl_ssl_read_internal): ditto
- ext/stringio/stringio.c (strio_sysread): ditto
- io.c (rb_io_write_nonblock): support non-blocking writes without raising an exception.
- ext/openssl/ossl_ssl.c (ossl_ssl_write_internal): ditto
- test/openssl/test_pair.rb (class OpenSSL): tests
- test/ruby/test_io.rb (class TestIO): ditto
- test/socket/test_nonblock.rb (class TestSocketNonblock): ditto
- test/stringio/test_stringio.rb (class TestStringIO): ditto

#64 - 09/27/2013 08:14 PM - headius (Charles Nutter)

Hah... I was just stopping by to suggest the keyword argument as a compromise form...and it turns out that's exactly what we went with. Excellent!

#65 - 05/30/2014 05:53 AM - headius (Charles Nutter)

Charles Nutter wrote:

> JRuby master (1.7.4) now has a new ext io/try_nonblock that implements just the IO portion of wycats's patch (I did not implement the StringIO and SSLSocket logic).

Just closing the circle here... JRuby 9000 will remove the hidden io/try_nonblock (which we never *really* intended to ship). It's ok, though...we'll ship support for the :exception option.

Carry on!

Files

try_nonblock.diff	19.8 KB	08/02/2011	wycats (Yehuda Katz)
try_nonblock.diff	20.3 KB	08/02/2011	wycats (Yehuda Katz)
try_nonblock.diff	20.2 KB	08/02/2011	wycats (Yehuda Katz)
noname	500 Bytes	08/04/2011	tenderlovemaking (Aaron Patterson)
noname	500 Bytes	11/20/2011	Anonymous
feature5138.pdf	38.2 KB	07/01/2012	tenderlovemaking (Aaron Patterson)
noname	500 Bytes	07/01/2012	Anonymous
nonblock_no_tuple.patch	20.3 KB	07/10/2013	tenderlovemaking (Aaron Patterson)
nonblock_tuple.patch	21.9 KB	07/10/2013	tenderlovemaking (Aaron Patterson)