Ruby trunk - Bug #8875

Select is not usable with SSLSocket

09/08/2013 02:17 AM - headius (Charles Nutter)

Status:	Rejected		
Priority:	Normal		
Assignee:	MartinBosslet (Martin Bosslet)		
Target version:			
ruby -v:	all	**Backport:**	1.9.3: UNKNOWN, 2.0.0: UNKNOWN

Description

Because of the various levels of buffering SSLSocket employs, it is not possible to reliably use IO.select to check when it has data available.

SSLSocket wraps a normal IO that it uses for reading and writing unencrypted data. This IO has its own buffers, at the OS/libc level.

Select normally operates against IO, checking whether data has been buffered or is available on the wire. However, in order to decrypt data on the wire, SSLSocket often needs to read more data than it needs, potentially draining the stream. This is problem #1.

This problem can be mitigated by making IO.select know that it's an SSLSocket and that it may have its own buffers.

However, there's another layer of buffering that happens in openssl/buffering.rb, where read, readpartial, read_nonblock, and methods that call them eventually hit fill_rbuf, which can easily drain both the IO buffers and the SSLSocket buffers into a Ruby-land buffer IO.select does not know about.

An example script is here: https://gist.github.com/headius/6477345

In investigating why this hangs on JRuby (under the original assumption that it was a JRuby issue) I realized that fill_rbuff is reading 16k bytes at a time to try to fill its internal buffer. This effectively drains all data in all buffers visible to IO.select, causing select to hang after the first read.

ruby-head (a few months old), Ruby 1.9.3p253, Ruby 1.8.7p358, JRuby (all versions), and Rubinius (all versions) are affected, because we all share buffering.rb which is where the problem lies.

This may be a known issue, but we continue to get bug reports from Ruby users claiming JRuby is failing to support select + SSLSocket correctly. I'd like to figure out if there's anything we as a community can do to fix this.

History

#1 - 09/08/2013 02:57 AM - headius (Charles Nutter)

I have been experimenting with some fixes for this.

In JRuby, my first fix was to make IO.select aware of SSLSocket's native buffers by adding a method to query if SSLSocket had buffered data itself. This adds the socket to the list of pending read streams and does not attempt to do a blocking select on it. This fixes the simple issue of sysread/sysread_nonblock reading more data than requested and potentially draining the stream and allows select to work correctly.

However, the buffering issue is harder to fix. I believe buffering.rb needs to go away entirely, or at least needs to not buffer data on its own.

A partial patch for buffering.rb (does not fix all uses of the Ruby-land buffer) allows my original case to run to completion, since the only buffers are the ones in SSLSocket proper and my IO.select patch can see those.

Here's the patch as it stands right now: https://gist.github.com/headius/6477733

#2 - 09/08/2013 03:02 AM - headius (Charles Nutter)

See also recent comments on http://jira.codehaus.org/browse/JRUBY-6874

#3 - 09/08/2013 05:23 AM - normalperson (Eric Wong)

"headius (Charles Nutter)" headius@headius.com wrote:

> However, the buffering issue is harder to fix. I believe buffering.rb needs to go away entirely, or at least needs to not buffer data on its own.

I think so, too. I believe any sort of explicit buffering by Ruby on
socket/pipe wrappers is wrong and only leads to surprising behavior and
bugs.

Anyways, I'm not sure if read/readpartial/write for IO.select should
ever be recommended. select() manpage in Linux recommends using
O_NONBLOCK for all I/O operations because of spurious wakeups, too.
So perhaps we should only be doing the _nonblock variants anyways.

(Also, consider the multi-threaded case where several threads are all
doing select + (recvmsg|accept) on the same shared socket)

#4 - 09/08/2013 07:03 AM - akr (Akira Tanaka)

2013/9/8 headius (Charles Nutter) headius@headius.com:

> In JRuby, my first fix was to make IO.select aware of SSLSocket's native buffers by adding a method to query if SSLSocket had buffered data itself. This adds the socket to the list of pending read streams and does not attempt to do a blocking select on it. This fixes the simple issue of sysread/sysread_nonblock reading more data than requested and potentially draining the stream and allows select to work correctly.

I think the right fix is that application uses read_nonblock before select.
See the document of IO#read_nonblock and OpenSSL::Buffering#read_nonblock.
I wrote this problem in [ruby-core:38681].

> However, the buffering issue is harder to fix. I believe buffering.rb needs to go away entirely, or at least needs to not buffer data on its own.

OpenSSL::Buffering#gets may be difficult to implement efficiently
without buffering.
--
Tanaka Akira

#5 - 09/08/2013 11:14 PM - headius (Charles Nutter)

akr (Akira Tanaka) wrote:

> 2013/9/8 headius (Charles Nutter) headius@headius.com:
>
> > In JRuby, my first fix was to make IO.select aware of SSLSocket's native buffers by adding a method to query if SSLSocket had buffered data itself. This adds the socket to the list of pending read streams and does not attempt to do a blocking select on it. This fixes the simple issue of sysread/sysread_nonblock reading more data than requested and potentially draining the stream and allows select to work correctly.
>
> I think the right fix is that application uses read_nonblock before select.
> See the document of IO#read_nonblock and OpenSSL::Buffering#read_nonblock.
> I wrote this problem in [ruby-core:38681].

I would agree, except that users are shown, through examples online and in source, that SSLSocket is "IO-like" and can be used anywhere an IO can be used. IO can do buffered IO and still be selectable. SSLSocket cannot.

This is a shame because without the buffering in buffering.rb, it *would* be feasible to make select work with SSLSocket, as I have done in JRuby.

> However, the buffering issue is harder to fix. I believe buffering.rb needs to go away entirely, or at least needs to not buffer data on its own.

> OpenSSL::Buffering#gets may be difficult to implement efficiently
> without buffering.

Difficult but not impossible :-) If the buffer in buffering.rb was known to SSLSocket proper and SSLSocket included it when reporting to IO.select that it has data buffered, all would be fine. This could either mean moving the implementations of gets and friends into C code to work directly against the SSLSocket buffers, or moving @rbuffer into a native location so it can be efficiently queried for data.

I may attempt to implement the latter proposal for JRuby, and then we could translate it for MRI. The bottom line is that there are too many buffers and not enough visibility into them. So we need to fix one thing or the other.

- Charlie

#6 - 09/09/2013 02:40 AM - drbrain (Eric Hodel)

- Status changed from Open to Assigned

- Assignee set to MartinBosslet (Martin Bosslet)

#7 - 09/10/2013 10:53 AM - akr (Akira Tanaka)

2013/9/8 headius (Charles Nutter) headius@headius.com:

> I would agree, except that users are shown, through examples online and in source, that SSLSocket is "IO-like" and can be used anywhere an IO can be used. IO can do buffered IO and still be selectable. SSLSocket cannot.
>
> This is a shame because without the buffering in buffering.rb, it *would* be feasible to make select work with SSLSocket, as I have done in JRuby.

What's happen when the remote side of SSL protocol sends a partial record?
In that case, select notify readability (of encrypted data) but no decrypted
data readable.

What's also happen when the remote side request renegotiation and local side
write system call blocks?
It is difficult to say it is not happen.

I think checking bufer emptyness of SSL is not enough to avoid blocking at
blocking read invocation after select.
--
Tanaka Akira

#8 - 09/27/2013 07:41 PM - headius (Charles Nutter)

akr (Akira Tanaka) wrote:

> 2013/9/8 headius (Charles Nutter) headius@headius.com:
>
> > I would agree, except that users are shown, through examples online and in source, that SSLSocket is "IO-like" and can be used anywhere an IO can be used. IO can do buffered IO and still be selectable. SSLSocket cannot.
> >
> > This is a shame because without the buffering in buffering.rb, it *would* be feasible to make select work with SSLSocket, as I have done in JRuby.
>
> What's happen when the remote side of SSL protocol sends a partial record?
> In that case, select notify readability (of encrypted data) but no decrypted
> data readable.

Doing a select followed by a blocking read of more data than is actually available would block on any socket. The amount of data available in this case is therefore 0 even if the socket is readable.

You should use read_nonblock in combination with select, so only what is available without blocking gets read off the wire. In this case, it would be an empty result (empty string or nil) or raise EAGAIN. The read logic, however, would drain the socket of its partial record into the encrypted buffer. Could we not just associate the buffer check the decrypted buffer, and always attempt to drain the encrypted buffer on any read?

1. Code selects on socket. Decrypted buffer is empty and no data is on the wire yet, so it blocks.
2. Partial record is written and select returns.
3. read_nonblock triggers read from the socket into encrypted buffer.
4. Not enough encrypted data is available to decrypt, so the data remains in the encrypted buffer.
5. read_nonblock raises EAGAIN
6. Code selects on socket. Decrypted buffer is still empty and no data is on the wire yet, so it blocks.
7. The rest of the partial record arrives on the wire. Select returns.
8. read_nonblock drains the rest of the data from the wire into encrypted buffer and drains encrypted buffer into decrypted buffer.

At this point, if the read_nonblock requested all of the decrypted data, that buffer would be drained and select would block again. If read_nonblock requested less than the available data, some would remain in the buffer and select would not block.

> What's also happen when the remote side request renegotiation and local side
> write system call blocks?
> It is difficult to say it is not happen.

It may be that you would need to select for both read and write to ensure this case succeeds. I think it would be a rare occurrence, though, since it will almost always be possible to write into the socket's internal buffer the small amount of data needed for renegotiation.

#9 - 10/07/2013 07:43 PM - akr (Akira Tanaka)

headius (Charles Nutter) wrote:

> You should use read_nonblock in combination with select, so only what is available without blocking gets read off the wire. In this case, it would be an empty result (empty string or nil) or raise EAGAIN. The read logic, however, would drain the socket of its partial record into the encrypted buffer. Could we not just associate the buffer check the decrypted buffer, and always attempt to drain the encrypted buffer on any read?

Yes. We should use read_nonblock.

> It may be that you would need to select for both read and write to ensure this case succeeds. I think it would be a rare occurrence, though, since it will almost always be possible to write into the socket's internal buffer the small amount of data needed for renegotiation.

I think you are optimistic.

#10 - 01/18/2014 02:20 PM - akr (Akira Tanaka)

- Status changed from Assigned to Rejected

Applications should use read_nonblock and read_nonblock doesn't block.

So this is a problem of applications which use IO.select and blocking methods of SSLSocket.