Ruby master - Bug #9356

TCPSocket.new does not seem to handle INTR

01/03/2014 07:29 PM - charliesome (Charlie Somerville)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	-	**Backport:**	1.9.3: UNKNOWN, 2.0.0: REQUIRED, 2.1: REQUIRED

Description

TCPSocket.new does not seem to handle EINTR properly.

In the attached test script, I try to open a TCP connection to my server and make an HTTP request while a background thread continually sends a signal to the process.

This causes the #write call to fail with:

x.rb:13:in write': Socket is not connected (Errno::ENOTCONN)
from x.rb:13:in'

This also appears to affect 2.0.0. 1.9.3 is unaffected.

Related issues:

Related to Ruby master - Bug #9547: TCPSocket.new causes an infinite loop whe...	**Closed**	**02/21/2014**

Associated revisions

Revision 7914a872 - 09/17/2014 09:20 PM - normal

socket (rsock_connect): fix and refactor for blocking

- ext/socket/init.c (rsock_connect): refactor for blocking (wait_connectable): clear error before wait [Bug #9356]

We no longer use non-blocking sockets to emulate blocking behavior,
so eliminate error-prone and confusing platform-dependent code.
According to POSIX, connect() only needs to be called once in the
face of EINTR, so do not loop on it.

Before waiting on connect, drop any pending errors, since
rb_wait_for_single_fd may not clear the existing error
properly.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@47617 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 47617 - 09/17/2014 09:20 PM - normal

socket (rsock_connect): fix and refactor for blocking

- ext/socket/init.c (rsock_connect): refactor for blocking (wait_connectable): clear error before wait [Bug #9356]

We no longer use non-blocking sockets to emulate blocking behavior,
so eliminate error-prone and confusing platform-dependent code.
According to POSIX, connect() only needs to be called once in the
face of EINTR, so do not loop on it.

Before waiting on connect, drop any pending errors, since
rb_wait_for_single_fd may not clear the existing error
properly.

Revision 47617 - 09/17/2014 09:20 PM - normalperson (Eric Wong)

socket (rsock_connect): fix and refactor for blocking

- ext/socket/init.c (rsock_connect): refactor for blocking (wait_connectable): clear error before wait [Bug #9356]

We no longer use non-blocking sockets to emulate blocking behavior,

so eliminate error-prone and confusing platform-dependent code.
According to POSIX, connect() only needs to be called once in the
face of EINTR, so do not loop on it.

Before waiting on connect, drop any pending errors, since
rb_wait_for_single_fd may not clear the existing error
properly.

Revision 47617 - 09/17/2014 09:20 PM - normal

socket (rsock_connect): fix and refactor for blocking

- ext/socket/init.c (rsock_connect): refactor for blocking (wait_connectable): clear error before wait [Bug #9356]

We no longer use non-blocking sockets to emulate blocking behavior,
so eliminate error-prone and confusing platform-dependent code.
According to POSIX, connect() only needs to be called once in the
face of EINTR, so do not loop on it.

Before waiting on connect, drop any pending errors, since
rb_wait_for_single_fd may not clear the existing error
properly.

Revision 47617 - 09/17/2014 09:20 PM - normal

socket (rsock_connect): fix and refactor for blocking

- ext/socket/init.c (rsock_connect): refactor for blocking (wait_connectable): clear error before wait [Bug #9356]

We no longer use non-blocking sockets to emulate blocking behavior,
so eliminate error-prone and confusing platform-dependent code.
According to POSIX, connect() only needs to be called once in the
face of EINTR, so do not loop on it.

Before waiting on connect, drop any pending errors, since
rb_wait_for_single_fd may not clear the existing error
properly.

Revision 47617 - 09/17/2014 09:20 PM - normal

socket (rsock_connect): fix and refactor for blocking

- ext/socket/init.c (rsock_connect): refactor for blocking (wait_connectable): clear error before wait [Bug #9356]

We no longer use non-blocking sockets to emulate blocking behavior,
so eliminate error-prone and confusing platform-dependent code.
According to POSIX, connect() only needs to be called once in the
face of EINTR, so do not loop on it.

Before waiting on connect, drop any pending errors, since
rb_wait_for_single_fd may not clear the existing error
properly.

Revision 47617 - 09/17/2014 09:20 PM - normal

socket (rsock_connect): fix and refactor for blocking

- ext/socket/init.c (rsock_connect): refactor for blocking (wait_connectable): clear error before wait [Bug #9356]

We no longer use non-blocking sockets to emulate blocking behavior,
so eliminate error-prone and confusing platform-dependent code.
According to POSIX, connect() only needs to be called once in the
face of EINTR, so do not loop on it.

Before waiting on connect, drop any pending errors, since
rb_wait_for_single_fd may not clear the existing error
properly.

History

#1 - 01/04/2014 02:59 AM - normalperson (Eric Wong)

This might be a bug exposed due to r36944
("avoid unnecessary select() calls before doing I/O")
which I don't want to revert.

Does the following fix it?

--- a/ext/socket/init.c
+++ b/ext/socket/init.c
@@ -400,12 +400,6 @@ rsock_connect(int fd, const struct sockaddr *sockaddr, int len, int socks)
status = (int)BLOCKING_REGION_FD(func, &arg);
if (status < 0) {
switch (errno) {

- case EINTR: -#if defined(ERESTART)
- case ERESTART: -#endif
- continue; - case EAGAIN: #ifdef EINPROGRESS case EINPROGRESS: @@ -426,6 +420,12 @@ rsock_connect(int fd, const struct sockaddr *sockaddr, int len, int socks) #if WAIT_IN_PROGRESS > 0 wait_in_progress = WAIT_IN_PROGRESS; #endif
- case EINTR: +#if defined(ERESTART)
- case ERESTART: +#endif
- continue; + status = wait_connectable(fd); if (status) { break; ---------- Fwiw, I'm not sure if the WAIT_IN_PROGRESS + getsockopt SO_ERROR is even necessary when we can just do wait_connectable(fd) followed by an eventual write/send/read/recv...

#2 - 01/04/2014 02:05 PM - charliesome (Charlie Somerville)

Eric: Unfortunately your patch doesn't fix it, still getting the same ENOTCONN error.

#3 - 01/04/2014 04:23 PM - normalperson (Eric Wong)

Thanks for trying. This is probably specific to *BSD sockets
implementation, so I can't reproduce it at the moment.

Can you try reverting parts of r36944 which affect IO#write?
("avoid unnecessary select() calls before doing I/O")

#4 - 02/08/2014 02:40 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> Thanks for trying. This is probably specific to *BSD sockets
> implementation, so I can't reproduce it at the moment.

Not happening on FreeBSD 9.x kernels, at least. I could not reproduce
the problem on FreeBSD 9.2 nor Debian GNU/kFreeBSD sid (x86_64).

#5 - 02/18/2014 09:34 AM - shugo (Shugo Maeda)

Eric Wong wrote:

> Eric Wong normalperson@yhbt.net wrote:
>
> > Thanks for trying. This is probably specific to *BSD sockets
> > implementation, so I can't reproduce it at the moment.
>
> Not happening on FreeBSD 9.x kernels, at least. I could not reproduce
> the problem on FreeBSD 9.2 nor Debian GNU/kFreeBSD sid (x86_64).

On FreeBSD 10, Errno::ENOTCONN isn't raised, but Ruby goes in an infinite loop because getsockopt(2) in wait_connectable() sets sockerr to 0.

Does anybody know why the following code in ext/socket/init.c is necessary?

```
        if (sockerr == 0)
            continue;       /* workaround for winsock */
```

#6 - 02/18/2014 09:58 AM - usa (Usaku NAKAMURA)

- ruby -v changed from ruby 2.1.0p0 (2013-12-25 revision 44422) [x86_64-darwin13.0] to -

Hi,

In message "[ruby-core:60819] [ruby-trunk - Bug #9356] TCPSocket.new does not seem to handle INTR"
on Feb.18,2014 18:34:40, shugo@ruby-lang.org wrote:

> Does anybody know why the following code in ext/socket/init.c is necessary?
>
> ```
>         if (sockerr == 0)
> ```

```
            continue;       /* workaround for winsock */
```

It was introduced at r7931 by me (9 years ago!),
but I forgot the reason.
If my comment of those days is believed, we may wrap it
with #ifdef_WIN32.

Regards,
--
U.Nakamura usa@garbagecollect.jp

#7 - 02/18/2014 10:43 AM - normalperson (Eric Wong)

"U.Nakamura" usa@garbagecollect.jp wrote:

> In message "[ruby-core:60819] [ruby-trunk - Bug #9356] TCPSocket.new does not seem to handle INTR"
> on Feb.18,2014 18:34:40, shugo@ruby-lang.org wrote:
>
> > Does anybody know why the following code in ext/socket/init.c is necessary?
> >
> > ```
> >         if (sockerr == 0)
> >             continue;       /* workaround for winsock */
> > ```
>
> It was introduced at r7931 by me (9 years ago!),
> but I forgot the reason.
> If my comment of those days is believed, we may wrap it
> with #ifdef_WIN32.

OK. I wonder if we should even use getsockopt(SO_ERROR) at all.

I know there's much literature which recommends it, but any error check
in this way is racy. Better to let any subsequent
write/read/send/recv/etc error out.

The following should work:

connect() -> EINPROGRESS
rb_wait_for_single_fd -> (must retry on EINTR)
(user calls) write() -> 0, ENOTCONN, EPIPE, ...

Note: rb_wait_for_single_fd is necessary in FreeBSD (at least) to avoid
ENOTCONN, Linux just returns EAGAIN on write if write immediately after
connect.

#8 - 02/19/2014 03:11 AM - shugo (Shugo Maeda)

Eric Wong wrote:

> OK. I wonder if we should even use getsockopt(SO_ERROR) at all.
>
> I know there's much literature which recommends it, but any error check
> in this way is racy. Better to let any subsequent
> write/read/send/recv/etc error out.

Could you describe such a race condition in detail?

I don't see a race condition on FreeBSD 10.

The problem I've seen is that rb_wait_for_single_fd() returns RB_WAITFD_OUT, and getsockopt() with SO_ERROR doesn't return any error (this is an expected behavior when the socket is connected), and Ruby goes in an infinite loop.

This problem was introduced by r31424, and usa is not guilty at least for the problem I've seen.

How about to fix the code as follows?

```
        if (sockerr == 0) {
            if (revents & RB_WAITFD_OUT) {
                break;
            }
            else {
                continue;       /* control is reached here on winsock? */
```

```
            }
        }
```

And the following code seems to be unnecessary.

```
        if ((revents & (RB_WAITFD_IN|RB_WAITFD_OUT)) == RB_WAITFD_OUT) {
            ret = 0;
            break;
        }
```

As the following comment says, the intentions of if (revents & (RB_WAITFD_IN|RB_WAITFD_OUT)) { might be if (revents & RB_WIATFD_IN && revents & RB_WAITFD_OUT), but I guess revents & RB_WAITFD_IN is true not only when a failure occurs, but when data is ready to read.

```
    /*
     * Stevens book says, successful finish turn on RB_WAITFD_OUT and
     * failure finish turn on both RB_WAITFD_IN and RB_WAITFD_OUT.
     */
```

Note that the original problem reported by Charlie must be a different issue.
Charlie, could you show the output of dtruss when the problem happens?

#9 - 02/19/2014 08:19 AM - normalperson (Eric Wong)

shugo@ruby-lang.org wrote:

> Eric Wong wrote:
>
> > OK. I wonder if we should even use getsockopt(SO_ERROR) at all.
> >
> > I know there's much literature which recommends it, but any error check
> > in this way is racy. Better to let any subsequent
> > write/read/send/recv/etc error out.
>
> Could you describe such a race condition in detail?

```
getsockopt(SO_ERROR) => no error
<kernel sees disconnect>
read/write => EPIPE, ENOTCONN, ...
```

Since we must check all (future) read/write operations for errors
anyways, getsockopt(SO_ERROR) is worthless.

> I don't see a race condition on FreeBSD 10.
>
> The problem I've seen is that rb_wait_for_single_fd() returns RB_WAITFD_OUT, and getsockopt() with SO_ERROR doesn't return any error (this is an expected behavior when the socket is connected), and Ruby goes in an infinite loop.
>
> This problem was introduced by r31424, and usa is not guilty at least for the problem I've seen.
>
> How about to fix the code as follows?
>
> ```
>         if (sockerr == 0) {
>             if (revents & RB_WAITFD_OUT) {
>                 break;
>             }
>             else {
>                 continue;       /* control is reached here on winsock? */
>             }
>         }
> ```

Maybe, but I wonder if we can just drop a lot of code...

```
http://bogomips.org/ruby.git/patch?id=a4212dc9516f4
```

#10 - 02/19/2014 09:01 AM - shugo (Shugo Maeda)

Eric Wong wrote:

> Could you describe such a race condition in detail?

getsockopt(SO_ERROR) => no error

> read/write => EPIPE, ENOTCONN, ...
>
> Since we must check all (future) read/write operations for errors

Ah, I see. However, if getsockopt() returns no error, we can know that at least connect() succeeded, right? I'm not sure whether it's a so-called race condition.

> anyways, getsockopt(SO_ERROR) is worthless.

I don't think getsockopt(SO_ERROR) is worthless because users can know error information sooner and more exactly than subsequent read() or write().

> Maybe, but I wonder if we can just drop a lot of code...
>
> http://bogomips.org/ruby.git/patch?id=a4212dc9516f4

With the patch, connect() is called again even if getsockopt(SO_ERROR) returns an error, so Errno::EINVAL is raised. It's confusing behavior.

#11 - 02/19/2014 09:33 AM - normalperson (Eric Wong)

shugo@ruby-lang.org wrote:

> Eric Wong wrote:
>
> > Could you describe such a race condition in detail?
>
> getsockopt(SO_ERROR) => no error
>
> read/write => EPIPE, ENOTCONN, ...
>
> Since we must check all (future) read/write operations for errors
>
> Ah, I see. However, if getsockopt() returns no error, we can know that at least connect() succeeded, right? I'm not sure whether it's a so-called race condition.

I'm not sure if we can know for sure due to implementation differences.
And even if connect() succeeded, the server could decide to disconnect right away after accept() due to overload, so probably less important.

> anyways, getsockopt(SO_ERROR) is worthless.
>
> I don't think getsockopt(SO_ERROR) is worthless because users can know error information sooner and more exactly than subsequent read() or write().

The error may be seen sooner, but I think the errors are uncommon
and most users will not care. They will see any error and handle it
their own way.

> Maybe, but I wonder if we can just drop a lot of code...
>
> http://bogomips.org/ruby.git/patch?id=a4212dc9516f4
>
> With the patch, connect() is called again even if getsockopt(SO_ERROR) returns an error, so Errno::EINVAL is raised. It's confusing behavior.

Ah, I forget the outer for(;;) loop. Maybe it's better to not loop,
the WAIT_IN_PROGRESS stuff is confusing...

#12 - 02/19/2014 09:33 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> Ah, I forget the outer for(;;) loop. Maybe it's better to not loop,

> the WAIT_IN_PROGRESS stuff is confusing...

I have no idea how portable this is:
http://bogomips.org/ruby.git/patch?id=f5e2eb00e5

Btw, I suspect the WAIT_IN_PROGRESS stuff is carried over from the
1.8 days where all sockets were non-blocking by default, and overly
complicated as a result. I don't even think EINPROGRESS/EAGAIN is
possible, only EINTR/ERESTART.

#13 - 02/19/2014 12:00 PM - shugo (Shugo Maeda)

Eric Wong wrote:

> > Ah, I see. However, if getsockopt() returns no error, we can know
> > that at least connect() succeeded, right? I'm not sure whether it's a
> > so-called race condition.
>
> I'm not sure if we can know for sure due to implementation differences.

Hmm..., do you know any implementation where getsockopt(SO_ERROR) causes a problem?
If there's such an implementation, it would be better to remove the call, but I don't come up with such a situation.

By contraries, getsockopt(SO_ERROR) might be necessary for winsock, if usa's comment is right.

> And even if connect() succeeded, the server could decide to disconnect
> right away after accept() due to overload, so probably less important.

It applies equally to connect() without signal interruption.
TCPSocket.new should handle EINTR as transparently as possible, I think.

> > > anyways, getsockopt(SO_ERROR) is worthless.
>
> > I don't think getsockopt(SO_ERROR) is worthless because users can know
> > error information sooner and more exactly than subsequent read() or
> > write().
>
> The error may be seen sooner, but I think the errors are uncommon
> and most users will not care. They will see any error and handle it
> their own way.

Most applications might handle errors roughly, but logging the exact error information would help trouble shooting.

> Ah, I forget the outer for(;;) loop. Maybe it's better to not loop,
> the WAIT_IN_PROGRESS stuff is confusing...

I agree that the code is complicated, and it's better to simplify it, if possible.

> I have no idea how portable this is:
> http://bogomips.org/ruby.git/patch?id=f5e2eb00e5
>
> Btw, I suspect the WAIT_IN_PROGRESS stuff is carried over from the
> 1.8 days where all sockets were non-blocking by default, and overly
> complicated as a result. I don't even think EINPROGRESS/EAGAIN is
> possible, only EINTR/ERESTART.

The patch worked both on Linux 3.2.0 and FreeBSD 10.0 for blocking sockets with signal interruption.

It might be better to re-call connect() on ERESTART as the error suggests, but I'm not sure.
It seems to be a Linux way to re-call connect(), but there's no description about ERESTART in Linux's manual of connect(2) and instead there's a reference to POSIX.1-2001 (SUSv3), to which Linux seem not to conform. Funny.

#14 - 02/19/2014 08:43 PM - normalperson (Eric Wong)

shugo@ruby-lang.org wrote:

Eric Wong wrote:

> Ah, I see. However, if getsockopt() returns no error, we can know
> that at least connect() succeeded, right? I'm not sure whether it's a
> so-called race condition.

I'm not sure if we can know for sure due to implementation differences.

Hmm..., do you know any implementation where getsockopt(SO_ERROR) causes a problem?
If there's such an implementation, it would be better to remove the call, but I don't come up with such a situation.

I'm not sure about implementations of getsockopt(SO_ERROR),
there are too many differences from what I see.

However, our use of getsockopt(SO_ERROR) causes problems
for maintainability and seems to lead to bugs.

By contraries, getsockopt(SO_ERROR) might be necessary for winsock, if usa's comment is right.

I'm not sure...
usa: can you try my last patch?
http://bogomips.org/ruby.git/patch?id=f5e2eb00e5

> And even if connect() succeeded, the server could decide to disconnect
> right away after accept() due to overload, so probably less important.

It applies equally to connect() without signal interruption.
TCPSocket.new should handle EINTR as transparently as possible, I think.

Agreed, and my f5e2eb00e5 patch handles EINTR.

> > anyways, getsockopt(SO_ERROR) is worthless.
>
> I don't think getsockopt(SO_ERROR) is worthless because users can know
> error information sooner and more exactly than subsequent read() or
> write().

The error may be seen sooner, but I think the errors are uncommon
and most users will not care. They will see any error and handle it
their own way.

Most applications might handle errors roughly, but logging the exact error information would help trouble shooting.

Yes, and connect() still shows ETIMEDOUT/ECONNRESET/EHOSTUNREACH/etc.
Interrupted connect() is rare, so I think having the extra
hard-to-maintain code causes more problems than it solves.
I don't think users will care which of connect/write/read fails,
they just need to know an error happened.

> I have no idea how portable this is:
> http://bogomips.org/ruby.git/patch?id=f5e2eb00e5
>
> Btw, I suspect the WAIT_IN_PROGRESS stuff is carried over from the
> 1.8 days where all sockets were non-blocking by default, and overly
> complicated as a result. I don't even think EINPROGRESS/EAGAIN is
> possible, only EINTR/ERESTART.

The patch worked both on Linux 3.2.0 and FreeBSD 10.0 for blocking sockets with signal interruption.

Thanks for testing. I expect it to have no problem on most *BSD.
I mainly wanted some Win/Solaris/Apple users to try it.

It might be better to re-call connect() on ERESTART as the error suggests, but I'm not sure.

> It seems to be a Linux way to re-call connect(), but there's no description about ERESTART in Linux's manual of connect(2) and instead there's a reference to POSIX.1-2001 (SUSv3), to which Linux seem not to conform. Funny.

I'm not sure, either. ERESTART should be handled by the system C library and not seen by us, even. Not sure which (or any currently supported) systems leak ERESTART...

#15 - 02/20/2014 08:26 AM - shugo (Shugo Maeda)

Eric Wong wrote:

> I'm not sure about implementations of getsockopt(SO_ERROR), there are too many differences from what I see.
>
> However, our use of getsockopt(SO_ERROR) causes problems for maintainability and seems to lead to bugs.

Hmm..., the problem I've seen seem not to be a platform-dependent issue, but a simple bug, because the current code goes in an infinite loop when the connection is established without errors.
The current code wouldn't work on any platform except Linux, where connect() can be restartable without errors when it's interrupted by signals, so control never reach wait_connectable().

If I don't misunderstand Kosaki-san's intention, the code might be able to be simplified by removing for(;;) and waiting only RB_WAITFD_OUT (but I don't know whether it works well with winsock...).

What do you think, Kosaki-san?

> > And even if connect() succeeded, the server could decide to disconnect right away after accept() due to overload, so probably less important.
>
> It applies equally to connect() without signal interruption.
> TCPSocket.new should handle EINTR as transparently as possible, I think.

> Agreed, and my f5e2eb00e5 patch handles EINTR.

By "transparently" I meant that if TCPSocket.new raises an exception when it's not interrupted by a signal and fails, it should also raise an exception when it's interrupted by a signal and asynchronous connect() fails.

> Most applications might handle errors roughly, but logging the exact error information would help trouble shooting.

> Yes, and connect() still shows ETIMEDOUT/ECONNRESET/EHOSTUNREACH/etc.
> Interrupted connect() is rare, so I think having the extra
> hard-to-maintain code causes more problems than it solves.
> I don't think users will care which of connect/write/read fails,
> they just need to know an error happened.

It depends on how complex error handling by getsockopt() is, and we have a slight difference of opinion here.
I'd like to hear others' thoughts.

> > It might be better to re-call connect() on ERESTART as the error suggests, but I'm not sure.
> > It seems to be a Linux way to re-call connect(), but there's no description about ERESTART in Linux's manual of connect(2) and instead there's a reference to POSIX.1-2001 (SUSv3), to which Linux seem not to conform. Funny.
>
> I'm not sure, either. ERESTART should be handled by the system C library and not seen by us, even. Not sure which (or any currently supported) systems leak ERESTART...

Linux's connect() does return ERESTART when it's interrupted by a signal.
On Linux, connect() is restartable. Please see the following page:

http://www.madore.org/~david/computers/connect-intr.html

The above page describes connect() returns EINTR on Linux, but it seems to return ERESTART nowadays.
(And it describes connect() returns EADDRINUSE instead of EALREADY on FreeBSD, but it returns EALREADY now.)

#16 - 02/20/2014 10:09 PM - normalperson (Eric Wong)

shugo@ruby-lang.org wrote:

> I'd like to hear others' thoughts.

Likewise. We need to hear folks with more experience on other OSes.

> Linux's connect() does return ERESTART when it's interrupted by a signal.
> On Linux, connect() is restartable. Please see the following page:
>
> http://www.madore.org/~david/computers/connect-intr.html
>
> The above page describes connect() returns EINTR on Linux, but it seems to return ERESTART nowadays.
> (And it describes connect() returns EADDRINUSE instead of EALREADY on FreeBSD, but it returns EALREADY now.)

Interesting. Anyways I'm not against handling ERESTART.

Note, POSIX connect manpage says this:

```
If connect() is interrupted by a signal that is caught while blocked
waiting to establish a connection, connect() shall fail and set errno
to [EINTR], but the connection request shall not be aborted, and the
connection shall be established asynchronously.
```

ref: http://pubs.opengroup.org/onlinepubs/009695399/functions/connect.html
Of course, not every system is POSIX-compliant.

Anyways, I have an alternative (v3) patch here which retries connect()
on EINTR and ERESTART:
http://bogomips.org/ruby.git/patch?id=8f48b1862
(also note my new switch/case style to avoid inline ifdef :)

However, I still prefer my v2 if possible:
http://bogomips.org/ruby.git/patch?id=f5e2eb00e5

#17 - 02/21/2014 03:43 AM - shugo (Shugo Maeda)

Eric Wong wrote:

> Anyways, I have an alternative (v3) patch here which retries connect()
> on EINTR and ERESTART:
> http://bogomips.org/ruby.git/patch?id=8f48b1862
> (also note my new switch/case style to avoid inline ifdef :)
>
> However, I still prefer my v2 if possible:
> http://bogomips.org/ruby.git/patch?id=f5e2eb00e5

I prefer the latter too, but is break missing in the case clause?

Linux's man page of connect(2) says it's conforming to POSIX.1-2001 (in spite of the fact that it returns ERESTART), so I hope we don't need the second connect(). What do you think, Kosaki-san?

Or only retry on ERESTART is enough. We don't need the nested switch statements in the former patch, do we?

And I'd like to add the following minimal error handling code to wait_connectable():

```
static int
wait_connectable(int fd)
{
    int sockerr;
    socklen_t sockerrlen;

    /*
     * Stevens book says, successful finish turn on RB_WAITFD_OUT and
     * failure finish turn on both RB_WAITFD_IN and RB_WAITFD_OUT.
     * So it's enough to wait only RB_WAITFD_OUT and check the pending error
     * by getsockopt().
     *
     * Note: rb_wait_for_single_fd already retries on EINTR/ERESTART
     */
    int revents = rb_wait_for_single_fd(fd, RB_WAITFD_OUT, NULL);
```

```
    if (revents < 0)
        rb_bug_errno("rb_wait_for_single_fd used improperly", errno);

    sockerrlen = (socklen_t)sizeof(sockerr);
    if (getsockopt(fd, SOL_SOCKET, SO_ERROR, (void *)&sockerr, &sockerrlen) < 0)
        return -1;
    if (sockerr != 0) {
        errno = sockerr;
        return -1;
    }

    return 0;
}
```

This doesn't include difficult-to-read platform-dependent code which causes infinite loops.

#18 - 02/21/2014 07:38 AM - normalperson (Eric Wong)

shugo@ruby-lang.org wrote:

> Eric Wong wrote:
>
> > Anyways, I have an alternative (v3) patch here which retries connect()
> > on EINTR and ERESTART:
> > http://bogomips.org/ruby.git/patch?id=8f48b1862
> > (also note my new switch/case style to avoid inline ifdef :)
> >
> > However, I still prefer my v2 if possible:
> > http://bogomips.org/ruby.git/patch?id=f5e2eb00e5
>
> I prefer the latter too, but is break missing in the case clause?

I don't usually add it for the last case... Do some compilers need it?

> Linux's man page of connect(2) says it's conforming to POSIX.1-2001 (in spite of the fact that it returns ERESTART), so I hope we don't need the second connect(). What do you think, Kosaki-san?
>
> Or only retry on ERESTART is enough. We don't need the nested switch statements in the former patch, do we?

I'm not sure... Anyways, I think I may realize what happened with
Charlie's EINTR...

> And I'd like to add the following minimal error handling code to wait_connectable():

I think it is OK with a minor tweak below:

```
static int
wait_connectable(int fd)
{
    int sockerr;
    socklen_t sockerrlen;

    /*
     * Stevens book says, successful finish turn on RB_WAITFD_OUT and
     * failure finish turn on both RB_WAITFD_IN and RB_WAITFD_OUT.
     * So it's enough to wait only RB_WAITFD_OUT and check the pending error
     * by getsockopt().
     *
     * Note: rb_wait_for_single_fd already retries on EINTR/ERESTART
     */
    int revents = rb_wait_for_single_fd(fd, RB_WAITFD_OUT, NULL);

    if (revents < 0)
     rb_bug_errno("rb_wait_for_single_fd used improperly", errno);

    sockerrlen = (socklen_t)sizeof(sockerr);
    if (getsockopt(fd, SOL_SOCKET, SO_ERROR, (void *)&sockerr, &sockerrlen) < 0)
        return -1;
    if (sockerr != 0) {
        errno = sockerr;
```

We should guard against sockerr setting errno to
EINTR, ERESTART, EINPROGRESS, EALREADY, EISCONN here...

Here is what I think happened in Charlie's original case:

```
connect() -> EINTR, kernel socket remembers this!
retry connect() -> EALREADY
wait_connectable()
    rb_wait_for_single_fd() -> success
    getsockopt(SO_ERROR), sockerr = EINTR from first connect!
    errno = sockerr -> raise :(
```

#19 - 02/21/2014 08:17 AM - shugo (Shugo Maeda)

Eric Wong wrote:

> > However, I still prefer my v2 if possible:
> > http://bogomips.org/ruby.git/patch?id=f5e2eb00e5
>
> I prefer the latter too, but is break missing in the case clause?

I don't usually add it for the last case... Do some compilers need it?

It's OK if it's your style. I just noticed it when I was trying to add an extra case clause for ERESTART.

```
    sockerrlen = (socklen_t)sizeof(sockerr);
    if (getsockopt(fd, SOL_SOCKET, SO_ERROR, (void *)&sockerr, &sockerrlen) < 0)
        return -1;
    if (sockerr != 0) {
        errno = sockerr;
```

> We should guard against sockerr setting errno to
> EINTR, ERESTART, EINPROGRESS, EALREADY, EISCONN here...

Agreed. You mean to expose only usual errors to users and to hide unusual platform-dependent errors, right?

> Here is what I think happened in Charlie's original case:
>
> connect() -> EINTR, kernel socket remembers this!
> retry connect() -> EALREADY
> wait_connectable()
> rb_wait_for_single_fd() -> success
> getsockopt(SO_ERROR), sockerr = EINTR from first connect!
> errno = sockerr -> raise :(

Charlie said Errno::ENOTCONN is raised by TCPSocket#write, not by TCPSocket.new.
I don't know why....

#20 - 02/21/2014 01:11 PM - shugo (Shugo Maeda)

- File wait_connectable_infinite_loop_minimal_fix.diff added

Shugo Maeda wrote:

> On FreeBSD 10, Errno::ENOTCONN isn't raised, but Ruby goes in an infinite loop because getsockopt(2) in wait_connectable() sets sockerr to 0.

Can I commit the attached minimal fix as a workaround for this infinite loop bug?

Ruby 1.9.3 release is scheduled on Feb 24, and this is the last chance to fix normal bugs.
It's hard to estimate the impact of Eric's patch by then.

#21 - 02/21/2014 03:06 PM - shugo (Shugo Maeda)

Shugo Maeda wrote:

> Shugo Maeda wrote:

> > On FreeBSD 10, Errno::ENOTCONN isn't raised, but Ruby goes in an infinite loop because getsockopt(2) in wait_connectable() sets sockerr to 0.
>
> Can I commit the attached minimal fix as a workaround for this infinite loop bug?
>
> Ruby 1.9.3 release is scheduled on Feb 24, and this is the last chance to fix normal bugs.
> It's hard to estimate the impact of Eric's patch by then.

I talked with Naruse-san and Nakamura-san, and have created #9547 and have committed the workaround.

#22 - 02/22/2014 12:48 AM - normalperson (Eric Wong)

shugo@ruby-lang.org wrote:

> Eric Wong wrote:
>
> > We should guard against sockerr setting errno to
> > EINTR, ERESTART, EINPROGRESS, EALREADY, EISCONN here...
>
> Agreed. You mean to expose only usual errors to users and to hide unusual
> platform-dependent errors, right?

Yes, because we already handle most of those with rb_wait_for_single_fd.

http://bogomips.org/ruby.git/patch?id=d8241102f54
git://80x24.org/ruby socket-connect-check-v4

> Charlie said Errno::ENOTCONN is raised by TCPSocket#write, not by TCPSocket.new.
> I don't know why....

Maybe one additional change helps:
http://bogomips.org/ruby.git/patch?id=66ca3633f43
git://80x24.org/ruby socket-connect-check-v5

I'm out of ideas. Charlie?

#23 - 02/22/2014 12:52 AM - usa (Usaku NAKAMURA)

- Related to Bug #9547: TCPSocket.new causes an infinite loop when interrupted by a signal added

#24 - 02/22/2014 05:34 AM - shugo (Shugo Maeda)

Eric Wong wrote:

> shugo@ruby-lang.org wrote:
>
> > Eric Wong wrote:
> >
> > > We should guard against sockerr setting errno to
> > > EINTR, ERESTART, EINPROGRESS, EALREADY, EISCONN here...
> >
> > Agreed. You mean to expose only usual errors to users and to hide unusual
> > platform-dependent errors, right?
>
> Yes, because we already handle most of those with rb_wait_for_single_fd.

Definitely.

> http://bogomips.org/ruby.git/patch?id=d8241102f54
> git://80x24.org/ruby socket-connect-check-v4

The code looks fine, but please remove the following comment in wait_connectable().

```
 * So it's enough to wait only RB_WAITFD_OUT and check the pending error
 * by getsockopt().
```

Or there might be no need to wait RB_WAITFD_IN. I'm not sure.

#25 - 02/22/2014 06:49 AM - normalperson (Eric Wong)

shugo@ruby-lang.org wrote:

> http://bogomips.org/ruby.git/patch?id=d8241102f54
> git://80x24.org/ruby socket-connect-check-v4

> The code looks fine, but please remove the following comment in wait_connectable().

```
 * So it's enough to wait only RB_WAITFD_OUT and check the pending error
 * by getsockopt().
```

OK, I'll remove if we commit these versions.

> Or there might be no need to wait RB_WAITFD_IN. I'm not sure.

No need on Linux, but I think it is also harmless.

I'd like to hear from Charlie to see if -v4 or -v5 can fix
his error, first. -v5 might be more better, in fact..

#26 - 09/16/2014 03:37 PM - noice@email.cz (NoICE Reaver)

Any news on this issue? Something is causing this ENOTCONN issue on OS X and this issue might be related:

13.3.0 Darwin Kernel Version 13.3.0: Tue Jun 3 21:27:35 PDT 2014; root:xnu-2422.110.17~1/RELEASE_X86_64 x86_64

ruby 2.0.0p253

```
Errno::ENOTCONN: Socket is not connected
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/protocol.rb:211:in `write'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/protocol.rb:211:in `write0'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/protocol.rb:185:in `block in write'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/protocol.rb:202:in `writing'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/protocol.rb:184:in `write'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/http/generic_request.rb:325:in `write_header'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/http/generic_request.rb:136:in `exec'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/http.rb:1404:in `block in transport_request'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/http.rb:1403:in `catch'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/http.rb:1403:in `transport_request'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/http.rb:1376:in `request'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:319:in `block in open_http'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/net/http.rb:852:in `start'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:313:in `open_http'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:708:in `buffer_open'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:210:in `block in open_loop'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:208:in `catch'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:208:in `open_loop'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:149:in `open_uri'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:688:in `open'
/Users/noice/.rvm/rubies/ruby-2.0.0-p353/lib/ruby/2.0.0/open-uri.rb:34:in `open'
...
```

ruby 2.1.2p95

```
Errno::ENOTCONN: Socket is not connected
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/protocol.rb:211:in `write'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/protocol.rb:211:in `write0'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/protocol.rb:185:in `block in write'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/protocol.rb:202:in `writing'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/protocol.rb:184:in `write'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/http/generic_request.rb:325:in `write_header'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/http/generic_request.rb:136:in `exec'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/http.rb:1406:in `block in transport_request'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/http.rb:1405:in `catch'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/http.rb:1405:in `transport_request'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/http.rb:1378:in `request'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:319:in `block in open_http'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/net/http.rb:853:in `start'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:313:in `open_http'
```

```
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:724:in `buffer_open'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:210:in `block in open_loop'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:208:in `catch'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:208:in `open_loop'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:149:in `open_uri'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:704:in `open'
/Users/noice/.rvm/rubies/ruby-2.1.2/lib/ruby/2.1.0/open-uri.rb:34:in `open'
...
```

#27 - 09/17/2014 07:58 AM - normalperson (Eric Wong)

noice@email.cz wrote:

> Any news on this issue? Something is causing this ENOTCONN issue on OS X and this issue might be related:

Can you try the -v4/-v5 patches I proposed? I'm pretty sure they work (I haven't thought about this in a while), but Charlie never came back to test...

#28 - 09/17/2014 09:29 AM - noice@email.cz (NoICE Reaver)

Tried both patches at once on current ruby HEAD, let it run for 20 minutes and the issue didn't happen. So .. the patches work :)

Just to be sure, tried to run it with pure ruby head and the issue is still there:

```
Errno::ENOTCONN: Socket is not connected
/Users/noice/.rvm/rubies/ruby-head-pure/lib/ruby/2.2.0/net/protocol.rb:211:in `write'
/Users/noice/.rvm/rubies/ruby-head-pure/lib/ruby/2.2.0/net/protocol.rb:211:in `write0'
/Users/noice/.rvm/rubies/ruby-head-pure/lib/ruby/2.2.0/net/protocol.rb:185:in `block in write'
/Users/noice/.rvm/rubies/ruby-head-pure/lib/ruby/2.2.0/net/protocol.rb:202:in `writing'
/Users/noice/.rvm/rubies/ruby-head-pure/lib/ruby/2.2.0/net/protocol.rb:184:in `write'
/Users/noice/.rvm/rubies/ruby-head-pure/lib/ruby/2.2.0/net/http/generic_request.rb:328:in `write_header'
```

#29 - 09/17/2014 09:20 PM - normalperson (Eric Wong)

- Backport changed from 1.9.3: UNKNOWN, 2.0.0: UNKNOWN, 2.1: UNKNOWN to 1.9.3: UNKNOWN, 2.0.0: REQUIRED, 2.1: REQUIRED

#30 - 09/17/2014 09:21 PM - Anonymous

- Status changed from Open to Closed

- % Done changed from 0 to 100

Applied in changeset r47617.

---

socket (rsock_connect): fix and refactor for blocking

- ext/socket/init.c (rsock_connect): refactor for blocking (wait_connectable): clear error before wait [Bug #9356]

We no longer use non-blocking sockets to emulate blocking behavior,
so eliminate error-prone and confusing platform-dependent code.
According to POSIX, connect() only needs to be called once in the
face of EINTR, so do not loop on it.

Before waiting on connect, drop any pending errors, since
rb_wait_for_single_fd may not clear the existing error
properly.

#31 - 09/17/2014 09:30 PM - normalperson (Eric Wong)

I've squashed my v5 and v4 patches together as r47617. Thanks NoICE for the confirmation on OSX (where this problem was originally reported).

usa: We may need additional testing on WIN32, but I think rb_wait_for_single_fd works around many existing compatibility issues. Thanks.

#32 - 09/18/2014 03:03 PM - usa (Usaku NAKAMURA)

Perhaps, we should write a test that actually causes EINTR during connect.

But, when I committed r7931, there was a failure of an existing test, I guess.

At the moment, such failure has not been observed.
So, I think that there is not a big portability problem in this patch.

Thank you, Eric.

#33 - 09/18/2014 07:32 PM - normalperson (Eric Wong)

usa@garbagecollect.jp wrote:

> Perhaps, we should write a test that actually causes EINTR during connect.
>
> But, when I committed r7931, there was a failure of an existing test, I guess.
> At the moment, such failure has not been observed.
> So, I think that there is not a big portability problem in this patch.

Thanks for confirming. Unfortunately, EINTR is not easy to trigger on a
connect test case to localhost. Not even -Wl,--wrap in GNU ld is useful
for testing this, it is dependent on socket state in the kernel.

#34 - 09/18/2014 11:30 PM - usa (Usaku NAKAMURA)

Eric Wong wrote:

> Thanks for confirming. Unfortunately, EINTR is not easy to trigger on a
> connect test case to localhost. Not even -Wl,--wrap in GNU ld is useful
> for testing this, it is dependent on socket state in the kernel.

Thank you, I had thought it would be so :-P
Then, let's wait users' reports or CI failures (of course, I don't want to see them and I hope your patch is all correct.)

Files

socket-eintr.rb	207 Bytes	01/03/2014	charliesome (Charlie Somerville)
wait_connectable_infinite_loop_minimal_fix.diff	478 Bytes	02/21/2014	shugo (Shugo Maeda)