# Ruby - Bug #6288

## Change error message for thread block to be less misleading

04/13/2012 06:51 PM - rklemme (Robert Klemme)

| | | | |
|---|---|---|---|
| **Status:** | Closed | | |
| **Priority:** | Normal | | |
| **Assignee:** | mame (Yusuke Endoh) | | |
| **Target version:** | | | |
| **ruby -v:** | ruby 1.9.3p125 (2012-02-16) [i386-cygwin] | **Backport:** | |

**Description**

Test case:

```
11:50:07 ~$ ruby19 -r thread -e 'q=SizedQueue.new 10;1_000_000.times {|i| p i;q.enq i}'
0
1
2
3
4
5
6
7
8
9
10
/opt/lib/ruby/1.9.1/thread.rb:301:in sleep': deadlock detected (fatal) from /opt/lib/ruby/1.9.1/thread.rb:301:in block in push'
from internal:prelude:10:in synchronize' from /opt/lib/ruby/1.9.1/thread.rb:297:in push'
from -e:1:in block in <main>' from -e:1:in times'
from -e:1:in `'
```

This is not a deadlock, but there is no other thread which could wake up main thread. Deadlock is misleading because in a more complex scenario where I had the error initially it wasn't obvious that the other thread had died and I looked for the wrong error in my code.

### Associated revisions

**Revision 7320d83753d90ff7fcc78567589f1eb52a7de9a0 - 04/23/2012 03:27 PM - mame (Yusuke Endoh)**

- thread.c (rb_check_deadlock): refine an error message of deadlock detection. [ruby-core:44336] [Bug #6288]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@35449 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

**Revision 7320d837 - 04/23/2012 03:27 PM - mame (Yusuke Endoh)**

- thread.c (rb_check_deadlock): refine an error message of deadlock detection. [ruby-core:44336] [Bug #6288]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@35449 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

### History

**#1 - 04/13/2012 11:07 PM - mame (Yusuke Endoh)**

*- Status changed from Open to Feedback*

Is "possible deadlock detected" better?

Please elaborate "a more complex scenario" with small example.

--
Yusuke Endoh mame@tsg.ne.jp

#### #2 - 04/14/2012 12:10 AM - rklemme (Robert Klemme)

Hi,

thanks for looking into this!

mame (Yusuke Endoh) wrote:

> Is "possible deadlock detected" better?

If I understand properly what the deadlock check does (see also #5258) it merely verifies that there is at least one thread left which could wake up this thread. So I'd rather have something like "No live threads left. Deadlock?", but then again my understanding of the code in question is not too thorough.

> Please elaborate "a more complex scenario" with small example.

```
$ ruby19 -r thread -e 'q=SizedQueue.new(100);r=Thread.new {until (x=q.deq).nil?; raise "SilentError";end};1000.times {|i| q.enq i};q.enq nil'
/opt/lib/ruby/1.9.1/thread.rb:301:in sleep': deadlock detected (fatal) from /opt/lib/ruby/1.9.1/thread.rb:301:in block in push'
from internal:prelude:10:in synchronize' from /opt/lib/ruby/1.9.1/thread.rb:297:in push'
from -e:1:in block in <main>' from -e:1:in times'
from -e:1:in `'
```

Basically what happens is that the reader thread silently dies as you can see if you set Thread.abort_on_exception:

```
$ ruby19 -r thread -e 'Thread.abort_on_exception=true;q=SizedQueue.new(100);r=Thread.new {until (x=q.deq).nil?; raise
"SilentError";end};1000.times {|i| q.enq i};q.enq nil'
-e:1:in `block in ': SilentError (RuntimeError)
```

Kind regards

robert

#### #3 - 04/14/2012 09:12 AM - mame (Yusuke Endoh)

*- Status changed from Feedback to Assigned*

*- Assignee set to mame (Yusuke Endoh)*

Hello,

2012/4/14 rklemme (Robert Klemme) shortcutter@googlemail.com:

> mame (Yusuke Endoh) wrote:
>
> > Is "possible deadlock detected" better?
>
> If I understand properly what the deadlock check does (see also #5258) it merely verifies that there is at least one thread left which could wake up this thread. So I'd rather have something like "No live threads left. Deadlock?", but then again my understanding of the code in question is not too thorough.

Looks reasonable to me. I'll commit unless there is no objection.

> > Please elaborate "a more complex scenario" with small example.
>
> ```
> $ ruby19 -r thread -e 'q=SizedQueue.new(100);r=Thread.new {until (x=q.deq).nil?; raise "SilentError";end};1000.times {|i| q.enq i};q.enq nil'
> /opt/lib/ruby/1.9.1/thread.rb:301:in sleep': deadlock detected (fatal) from /opt/lib/ruby/1.9.1/thread.rb:301:in block in push'
>  from internal:prelude:10:in synchronize' from /opt/lib/ruby/1.9.1/thread.rb:297:in push'
>  from -e:1:in block in <main>' from -e:1:in times'
>  from -e:1:in `'
> ```
>
> Basically what happens is that the reader thread silently dies as you can see if you set Thread.abort_on_exception:

It does not make sense. Did you write the code to wait forever?
If so, you should write "sleep" simply. If not, your code is
actually "deadlocked", in a common sense.

Why didn't you insist that Thread.abort_on_exception be true by
default? I can't understand why you blame deadlock detection.

--
Yusuke Endoh mame@tsg.ne.jp

#### #4 - 04/16/2012 09:07 PM - rklemme (Robert Klemme)

mame (Yusuke Endoh) wrote:

> Hello,
>
> 2012/4/14 rklemme (Robert Klemme) shortcutter@googlemail.com:
>
> > mame (Yusuke Endoh) wrote:
> >
> > > Is "possible deadlock detected" better?
> >
> > If I understand properly what the deadlock check does (see also #5258) it merely verifies that there is at least one thread left which could wake up this thread. So I'd rather have something like "No live threads left. Deadlock?", but then again my understanding of the code in question is not too thorough.
>
> Looks reasonable to me. I'll commit unless there is no objection.

Great, thank you!

> > Please elaborate "a more complex scenario" with small example.
>
> $ ruby19 -r thread -e 'q=SizedQueue.new(100);r=Thread.new {until (x=q.deq).nil?; raise "SilentError";end};1000.times {|i| q.enq i};q.enq nil'
> /opt/lib/ruby/1.9.1/thread.rb:301:in sleep': deadlock detected (fatal) from /opt/lib/ruby/1.9.1/thread.rb:301:in block in push'
> from internal:prelude:10:in synchronize' from /opt/lib/ruby/1.9.1/thread.rb:297:in push'
> from -e:1:in block in <main>' from -e:1:in times'
> from -e:1:in `'
>
> Basically what happens is that the reader thread silently dies as you can see if you set Thread.abort_on_exception:
>
> It does not make sense. Did you write the code to wait forever?

I am not sure what you mean. The real code was more complex and the exception was thrown from another method - unintentionally of course. This is just a simplified example to illustrate the situation.

> If so, you should write "sleep" simply. If not, your code is
> actually "deadlocked", in a common sense.

There is no deadlock because there are no two threads (or processes) accessing resources in a bad order. I am not aware of any deadlock which can be caused by a single thread only. If you find a definition of deadlock which needs only a single thread / action / process please let us know.

"A deadlock is a situation wherein two or more competing actions are each waiting for the other to finish, and thus neither ever does."
http://en.wikipedia.org/wiki/Deadlock

> Why didn't you insist that Thread.abort_on_exception be true by
> default? I can't understand why you blame deadlock detection.

Well, others have done already before. :-) Also, regardless of abort_on_exception the error message for this particular situation is at least misleading if not plain wrong (according to definition of "deadlock"). The default value of abort_on_exception does not change that fact a bit. I mean, the same would happen with a different default of abort_on_exception and someone setting it explicitly to false.

Kind regards

robert

#### #5 - 04/16/2012 10:23 PM - mame (Yusuke Endoh)

Hello,

2012/4/16 rklemme (Robert Klemme) shortcutter@googlemail.com:

> > If so, you should write "sleep" simply. If not, your code is
> > actually "deadlocked", in a common sense.

> There is no deadlock because there are no two threads (or processes) accessing resources in a bad order. I am not aware of any deadlock which can be caused by a single thread only. If you find a definition of deadlock which needs only a single thread / action / process please let us know.
>
> "A deadlock is a situation wherein two or more competing actions are each waiting for the other to finish, and thus neither ever does." http://en.wikipedia.org/wiki/Deadlock

I see.

But, what do you think about the following?

```
m = Mutex.new
m.lock
m.lock  #=> deadlock; recursive locking
```

The article of wikipedia also says:

"For non-recursive locks, a lock may be entered only once (where a
single thread entering twice without unlocking will cause a
deadlock..."

The definition of "deadlock" is difficult :-)

My informal definition of "deadlock" is, a situation where
all threads are waiting for an action of other threads.
For example, Thread.stop, Queue#pop, Mutex#lock, etc. may
wait for other threads. If the only thread do so in a
single threaded program, *all* threads are indeed waiting,
which is a deadlock in my definition.

Anyway, regardless of the definition, I think your message
proposal is better than the current, so I'll change it.
Thanks!

--
Yusuke Endoh mame@tsg.ne.jp

#### #6 - 04/24/2012 12:27 AM - mame (Yusuke Endoh)

*- Status changed from Assigned to Closed*

*- % Done changed from 0 to 100*

This issue was solved with changeset r35449.
Robert, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

---

- thread.c (rb_check_deadlock): refine an error message of deadlock detection. [ruby-core:44336] [Bug #6288]