Ruby master - Bug #4266
Timeouts in threads cause "ThreadError: deadlock; recursive locking"
01/12/2011 01:59 AM - cjbouraro (Christopher Bottaro)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>kosaki (Motohiro KOSAKI)</td>
</tr>
<tr>
<td>Target version:</td>
<td>2.0.0</td>
</tr>
<tr>
<td>Backport:</td>
<td></td>
</tr>
</tbody>
</table>

ruby -v:
ruby 1.9.2p136 (2010-12-25 revision 30365) [x86_64-darwin10.6.0]

Description
=end

Run the attached file (or this pastie http://pastie.org/1449542) a few times and you'll eventually get:

ThreadError: deadlock; recursive locking:
  <internal:prelude>:8:in lock'
  bin/deadlock_test.rb:86:in block (4 levels) in <main>'
  /Users/cjbottaro/.rvm/rubies/ruby-1.9.2-p136/lib/ruby/1.9.1/timeout.rb:57:intimeout'
  bin/deadlock_test.rb:85:in block (3 levels) in <main>'
  bin/deadlock_test.rb:83:intimes'
  bin/deadlock_test.rb:83:in `block (2 levels) in '

I've had the script run successfully over 5 times in a row before getting the errors, so if it doesn't happen the first few times... keep trying.

The problem doesn't happen in 1.8.7 or Jruby, but does happen in 1.9.1.
=end

Related issues:
- Related to Ruby master - Bug #4285: Ruby don't have asynchronous exception s... Closed 01/17/2011
- Related to Ruby master - Bug #4289: Timeouts in threads cause SEGV Closed 01/18/2011

Associated revisions
Revision 6c56dae4 - 11/19/2012 10:22 AM - kosaki (Motohiro KOSAKI)
  prelude.rb: Moved Mutex#synchronize to
  thread.c (rb_mutex_synchronize_m): here. [Bug #4266]
  git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37724 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 37724 - 11/19/2012 10:22 AM - kosaki (Motohiro KOSAKI)
  prelude.rb: Moved Mutex#synchronize to
  thread.c (rb_mutex_synchronize_m): here. [Bug #4266]

Revision 37724 - 11/19/2012 10:22 AM - kosaki (Motohiro KOSAKI)
  prelude.rb: Moved Mutex#synchronize to
  thread.c (rb_mutex_synchronize_m): here. [Bug #4266]

Revision 37724 - 11/19/2012 10:22 AM - kosaki (Motohiro KOSAKI)
  prelude.rb: Moved Mutex#synchronize to
  thread.c (rb_mutex_synchronize_m): here. [Bug #4266]

Revision 37724 - 11/19/2012 10:22 AM - kosaki (Motohiro KOSAKI)
  prelude.rb: Moved Mutex#synchronize to
  thread.c (rb_mutex_synchronize_m): here. [Bug #4266]

Revision 37724 - 11/19/2012 10:22 AM - kosaki (Motohiro KOSAKI)
  prelude.rb: Moved Mutex#synchronize to
  thread.c (rb_mutex_synchronize_m): here. [Bug #4266]

Revision 37724 - 11/19/2012 10:22 AM - kosaki (Motohiro KOSAKI)
  prelude.rb: Moved Mutex#synchronize to
  thread.c (rb_mutex_synchronize_m): here. [Bug #4266]

Revision 37724 - 11/19/2012 10:22 AM - kosaki (Motohiro KOSAKI)
  prelude.rb: Moved Mutex#synchronize to
  thread.c (rb_mutex_synchronize_m): here. [Bug #4266]
FYI, JRuby does not use timeout.rb anymore (as of JRuby 1.3ish I think). We use a native implementation based on Java’s timed, thread-pooling executor in java.util.concurrent. The original timeout has many race conditions that are very difficult to fix.

A question was raised about whether #to should understand context, as in knowing a top-level object being coerced to YAML should include the --- header. The answer seems to come from Marshal.

Marshal.dump(object) knows how to produce the marshal header, linking, and so on. Marshal is the master of this format. But Marshal defers to the objects themselves for actual content to go into that marshaled output, calling _dump or marshal_dump on each object in turn. With the #to protocol, Marshal.dump could be implemented as:

def Marshal.dump(obj)
  emit_header
  obj.to(Marshal)
end

In the same way that marshal_dump is used today.

Another concern raised is whether #to should be expected on all objects all the time, rather than just always calling MyClass.coerce(obj). The answer is simple: you want individual source types to control how they coerce to a target type, rather than expecting the target type to know about all possible sources. The default protocol where #to calls type.coerce would simply be a default behavior.

Sorry about that, wrong bug. Too many bug windows open.

This test program also makes ruby crash on my environment (trunk + linux x86_64).

Related [Bug #3880]?

--- Control frame information -----------------------------
c:0011 p:----- s:0036 b:0036 l:000035 d:000035 CFUNC :exception
c:0010 p:----- s:0034 b:0034 l:000033 d:000033 CFUNC :lock
c:0007 p:0109 s:0026 b:0026 l:001ad0 d:001ad0 METHOD /usr/local/lib/ruby/1.9.1/timeout.rb:57
c:0005 p:----- s:0011 b:0011 l:000010 d:000010 FINISH
c:0004 p:----- s:0009 b:0009 l:000008 d:000008 CFUNC :times
c:0003 p:0010 s:0006 b:0006 l:000bb8 d:00012c8 BLOCK deadlock_test.rb:18
c:0002 p:----- s:0004 b:0004 l:000003 d:000003 FINISH

08/07/2021 2/6
#6 - 01/12/2011 02:41 PM - kosaki (Motohiro KOSAKI)

> Hm, the crash issue seems different and unrelated one from this. because ruby_1_9_2 can reproduce recursive deadlock issue, but can't reproduce crash.

#7 - 01/12/2011 02:45 PM - kosaki (Motohiro KOSAKI)

- File diffdiff added
- Priority changed from Normal to 5

> Old days, C# had similar issue. and they changed Monitor.enter(= our Mutex.lock) semantics.

http://www.bluebytesoftware.com/blog/2007/01/30/MonitorEnterThreadAbortsAndOrphanedLocks.aspx

So, Can we take similar way? The attached patch kill a race in mutex.synchronize. At least, my ruby_1_9_2 branch don't reproduce the issue anymore.

Thanks.

#8 - 01/12/2011 03:01 PM - kosaki (Motohiro KOSAKI)

- Assignee set to kot (Koichi Sasada)

> And, I think asynchronous exception should be blocked and delayed while running ensure block.
Otherwise Mutex.unlock can be bypassed.

ko1, What do you think?
=end

---

That said, our Mutex.synchronize() method has two unsafe point.

```ruby
def synchronize
  self.lock
  // (1)
  begin
    yield
  ensure
  // (2)
  self.unlock rescue nil
end
end
```

If the mutex got thread#raise at (1), it doesn't call unlock because it haven't entered -begin- block. And if got at (2), it also doesn't call unlock because we are no longer in -begin- block.

My patch fixed only (1).

Thanks.
=end

---

Alternative one is here. This one fixes both (1) and (2). But it only fixes Mutex#synchronize. That said, timeout[] still might not call ruby level ensure block.

---

Hi,

I plan to commit mutex-synchronize-use-c-implementation.patch at this weekend. So if anyone have objection, please let me know soon.
[end]

---

This issue also applies to MonitorMixin (monitor.rb) as it uses the same lock/begin yield/ensure block.
=end

---

This patch works both with and without the mutex-synchronize-use-c-implementation.patch posted here.
=end

---

This issue also applies to MonitorMixin (monitor.rb) as it uses the same lock/begin yield/ensure block.
=end

---

I have a short-term fix for the related issue in Bug #4289
http://redmine.ruby-lang.org/attachments/1572/0001-timeout.rb-avoid-introducing-new-class-for-every-tim.patch

It works both with and without the mutex-synchronize-use-c-implementation.patch posted here.
=end

---

This issue also applies to MonitorMixin (monitor.rb) as it uses the same lock/begin yield/ensure block.
=end

---

This patch fixes Mutex#synchronize. That said, timeout[] still might not call ruby level ensure block.
=end
#16 - 06/26/2011 04:11 PM - nahi (Hiroshi Nakamura)
- Target version changed from 1.9.2 to 1.9.3

#17 - 06/26/2011 07:10 PM - ko1 (Koichi Sasada)
- Target version changed from 1.9.3 to 2.0.0

Let's talk it after 1.9.3 release.

#18 - 08/13/2011 08:55 PM - jtara (Jon Tara)

Brian Gugliemetti wrote:

File monitor-synchronize-use-c-implementation.patch added

There is a parameter mis-match between the declaration and definition of rb_mon_synchronize. Compilation fails on at least OSX.

#19 - 04/19/2012 08:13 AM - MingVonsalis (Ming Vonsalis)

jtara (Jon Tara) wrote:

Brian Gugliemetti wrote:

File monitor-synchronize-use-c-implementation.patch added

There is a parameter mis-match between the declaration and definition of rb_mon_synchronize. Compilation fails on at least OSX.

http://originsofinuitart.posterous.com/

#20 - 11/17/2012 12:43 AM - trapni (Christian Parpart)

We just ran into this very ugly bug in production, too. We would really appreciate a fix in 1.9.4, a pure bugfix release of the 1.9 branch since this is a major bug. :)

#21 - 11/17/2012 01:04 AM - ko1 (Koichi Sasada)

- Assignee changed from ko1 (Koichi Sasada) to kosaki (Motohiro KOSAKI)

I passed this ticket to thread specialist.

#22 - 11/17/2012 10:31 AM - kosaki (Motohiro KOSAKI)

Brian,

Can you please provide us a reproducer for you monitor.rb issue? And can you please make a new ticket for your issue to prevent cross line discussion? I plan to close this ticket soon and I also hope to don't forget your issue.

Thanks.

#23 - 11/19/2012 07:22 PM - kosaki (Motohiro KOSAKI)

- Status changed from Assigned to Closed
- % Done changed from 0 to 100

This issue was solved with changeset r37724.

Christopher, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- prelude.rb: Moved Mutex#synchronize to
- thread.c (rb_mutex_synchronize_m): here. [Bug #4266]

Files

deadlock_test.rb 554 Bytes 01/12/2011 cjbottaro (Christopher Bottaro)
diffdiff 1.38 KB 01/12/2011 kosaki (Motohiro KOSAKI)
mutex-synchronize-use-c-implementation.patch 1.28 KB 01/12/2011 kosaki (Motohiro KOSAKI)