Ruby master - Bug #12298

Indeterministic ruby behavior when another thread is killed

04/18/2016 12:58 PM - rupert (Robert Pankowecki)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
<tr>
<td>ruby -v:</td>
<td>ruby 2.3.0p0 (2015-12-25 revision 53290) [x86_64-linux]</td>
</tr>
<tr>
<td>Backport:</td>
<td>2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN</td>
</tr>
</tbody>
</table>

Description

```ruby
#!ruby
require 'securerandom'
class MyThread < ::Thread; end

def delay
  15
end

def run
  loop { work }
  rescue Exception => e
    puts "#{Time.now} Exception"
  ensure
    puts "#{Time.now} stopping agent"
  end

  puts "#{Time.now} start work"
  10_000_000.times { SecureRandom.hex }
  rescue StandardError => e
    puts "#{Time.now} Error"
  ensure
    puts "#{Time.now} start sleep"
    sleep(delay)
    puts "#{Time.now} finished sleep"
  end

  t = MyThread.new{ run }

  at_exit do
    puts "#{Time.now} killing thread"
    Thread.kill(t)
    puts "#{Time.now} killed thread"
  end

  sleep(10)
  exit
```

I tried running this script multiple times in ruby 2.1.10, 2.2.4, 2.3.0 and I get inconsistent behavior. Sometimes the main thread does not wait for the second thread.

2016-04-15 11:07:09 +0200 start work
2016-04-15 11:07:19 +0200 killing thread
2016-04-15 11:07:19 +0200 killed thread
2016-04-15 11:07:19 +0200 stopping agent

And sometimes it does.

2016-04-15 11:07:26 +0200 start work
I appears that under higher CPU usage the 2nd scenario is more likely. In normal conditions the 1st happens more often probably.

I described the whole story in my blogpost

I am not sure which behavior is ruby default (I assume not waiting for other threads) but sometimes apparently ruby does wait for other threads to finish.

History

#1 - 04/18/2016 01:00 PM - rupert (Robert Pankowecki)
Related Honeybadger bug: https://github.com/honeybadger-io/honeybadger-ruby/issues/186

#2 - 04/19/2016 06:47 PM - drbrain (Eric Hodel)
- File a.txt added
- File b.txt added
- File 12998.rb added
- Status changed from Open to Rejected

This is not a ruby bug. Thread scheduling is inherently non-deterministic.

Sometimes you'll switch to the work thread before reaching rb_thread_terminate_all which allows the ensure to run, sometimes you won't.

I've attached a reduced test case and logs from ruby built with THREAD_DEBUG=1

Honeybadger should probably not sleep in ensure as a way to reduce thrashing when work finishes (I'm guessing that's what they use the sleep for) and should use a different mechanism instead.

#3 - 04/20/2016 06:09 AM - shevegen (Robert A. Heiler)

Hmm. Although the report was already rejected, and even if we all may agree that the honeybadger code was not brilliant, I feel that the overall issue here in regards to Threads may be useful for more people in the future too.

For instance, without the blog explanation, where else would you find that much information about ruby code used for "real"? From the official documentation of Threads?

http://ruby-doc.org/core-2.3.0/Thread.html

The documentation is not bad at all, mind you, but the blog semi-taught me more than the documentation would.

There may also be small improvements. We have instance methods like:

"See also the instance methods alive? and stop?"

In the code he checked whether the thread was aborting:

Thread.current.status == "aborting"

This could be simplified if the ruby code would allow for this check:

Thread.current.aborting?

Or perhaps even

Thread.aborting?

(I do not really know Threads that well that I can suggest an API that makes sense / is logical.)

Without that block, I would probably have never been able to figure out that a thread is not just alive or dead but may be in between
the two like the schroedinger cat.

Eric Hodel wrote:

This is not a ruby bug. Thread scheduling is inherently non-deterministic.

Sometimes you'll switch to the work thread before reaching rb_thread_terminate_all which allows the ensure to run, sometimes you won't.

I get that. But why does Ruby wait for the second thread to finish? Or is it that Ruby always waits for those threads to finish, but the Thread gets un-catchable Exception and only runs the ensure blocks and that's what is happening?

In ruby_cleanup:

First we mark the main thread as killed: [https://github.com/ruby/ruby/blob/trunk/eval.c#L186](https://github.com/ruby/ruby/blob/trunk/eval.c#L186)

Then we kill all threads: [https://github.com/ruby/ruby/blob/trunk/eval.c#L191](https://github.com/ruby/ruby/blob/trunk/eval.c#L191)

Then we start at_exit hooks: (via ruby_finalize_1) [https://github.com/ruby/ruby/blob/trunk/eval.c#L222](https://github.com/ruby/ruby/blob/trunk/eval.c#L222)

Sometimes ruby switches to the killed thread right away, so it sleeps in the ensure in your test case. It can't switch back to the main thread because it is marked as killed.

Sometimes ruby doesn't switch to the killed thread so it gets two signals and never sleeps (because the second arrives in the ensure when it tries to sleep).

### Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.txt</td>
<td>3.09 KB</td>
<td>04/19/2016</td>
<td>drbrain (Eric Hodel)</td>
</tr>
<tr>
<td>b.txt</td>
<td>3.58 KB</td>
<td>04/19/2016</td>
<td>drbrain (Eric Hodel)</td>
</tr>
<tr>
<td>12998.rb</td>
<td>508 Bytes</td>
<td>04/19/2016</td>
<td>drbrain (Eric Hodel)</td>
</tr>
</tbody>
</table>