Ruby trunk - Bug #5768

TestRequire#test_race_exceptionで競合するケースがまだある

12/17/2011 10:48 AM - naruse (Yui NARUSE)

Status:	Closed		
Priority:	Normal		
Assignee:	kosaki (Motohiro KOSAKI)		
Target version:	2.0.0		
ruby -v:	-	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

まだrequireで競合するケースが残っています。
現在のテストだと確率的にしか起きませんが、以下の通り変更すると確実に起きるようになります。

diff --git a/test/ruby/test_require.rb b/test/ruby/test_require.rb
index 9186a6f..262a5ef 100644
--- a/test/ruby/test_require.rb
+++ b/test/ruby/test_require.rb
@@ -352,7 +352,7 @@ class TestRequire < Test::Unit::TestCase
TestRequire.scratch << :pre
Thread.pass until t2 = TestRequire.scratch[1]
Thread.pass until t2.stop?
-open(**FILE**, "w") {|f| f.puts "TestRequire.scratch << :post"}
+open(**FILE**, "w") {|f| f.puts "TestRequire.scratch << :post"; f.puts "t1,t2=TestRequire.scratch[1, 2];if Thread.current == t2;
Thread.pass until t1.stopped?; end"}
raise "con1"
EOS
tmp.close
@@ -364,6 +364,7 @@ raise "con1"
t2_res = nil

```
t1 = Thread.new do
```

- [redacted]

begin
require(path)
rescue RuntimeError
@@ -389,8 +390,8 @@ raise "con1"
assert_nothing_raised(ThreadError, bug5754) {t1.join}
assert_nothing_raised(ThreadError, bug5754) {t2.join}

- assert_equal(true, (t1_res ^ t2_res), bug5754)
- assert_equal([:pre, t2, :post, :t2, :t1], scratch, bug5754)
- assert_equal(true, (t1_res ^ t2_res), bug5754 + " t1:#{t1_res} t2:#{t2_res}")
- assert_equal([:pre, t1, t2, :post, :t2, :t1], scratch, bug5754)

ensure
tmp.close(true) if tmp
end

Related issues:

Related to Ruby trunk - Bug #5754: Double require bug in 1.9.3	**Closed**	**12/13/2011**

Associated revisions

Revision 82ab1e18 - 12/30/2011 10:25 PM - naruse (Yui NARUSE)

- thread.c (rb_barrier_waiting): save the number of waiting threads in RBASIC()->flags. [ruby-dev:45002] [Bug #5768]
- thread.c (rb_barrier_wait): increment and decrement around rb_mutex_lock, and use rb_barrier_waiting().
- thread.c (rb_barrier_release): use rb_barrier_waiting().
- thread.c (rb_barrier_destroy): ditto.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@34163 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 34163 - 12/30/2011 10:25 PM - naruse (Yui NARUSE)

- thread.c (rb_barrier_waiting): save the number of waiting threads in RBASIC()->flags. [ruby-dev:45002] [Bug #5768]
- thread.c (rb_barrier_wait): increment and decrement around rb_mutex_lock, and use rb_barrier_waiting().
- thread.c (rb_barrier_release): use rb_barrier_waiting().
- thread.c (rb_barrier_destroy): ditto.

Revision 34163 - 12/30/2011 10:25 PM - naruse (Yui NARUSE)

- thread.c (rb_barrier_waiting): save the number of waiting threads in RBASIC()->flags. [ruby-dev:45002] [Bug #5768]
- thread.c (rb_barrier_wait): increment and decrement around rb_mutex_lock, and use rb_barrier_waiting().

- thread.c (rb_barrier_release): use rb_barrier_waiting().
- thread.c (rb_barrier_destroy): ditto.

Revision 34163 - 12/30/2011 10:25 PM - naruse (Yui NARUSE)

- thread.c (rb_barrier_waiting): save the number of waiting threads in RBASIC()->flags. [ruby-dev:45002] [Bug #5768]
- thread.c (rb_barrier_wait): increment and decrement around rb_mutex_lock, and use rb_barrier_waiting().
- thread.c (rb_barrier_release): use rb_barrier_waiting().
- thread.c (rb_barrier_destroy): ditto.

Revision 34163 - 12/30/2011 10:25 PM - naruse (Yui NARUSE)

- thread.c (rb_barrier_waiting): save the number of waiting threads in RBASIC()->flags. [ruby-dev:45002] [Bug #5768]
- thread.c (rb_barrier_wait): increment and decrement around rb_mutex_lock, and use rb_barrier_waiting().
- thread.c (rb_barrier_release): use rb_barrier_waiting().
- thread.c (rb_barrier_destroy): ditto.

Revision 34163 - 12/30/2011 10:25 PM - naruse (Yui NARUSE)

- thread.c (rb_barrier_waiting): save the number of waiting threads in RBASIC()->flags. [ruby-dev:45002] [Bug #5768]
- thread.c (rb_barrier_wait): increment and decrement around rb_mutex_lock, and use rb_barrier_waiting().
- thread.c (rb_barrier_release): use rb_barrier_waiting().
- thread.c (rb_barrier_destroy): ditto.

History

#1 - 12/19/2011 10:58 AM - nagachika (Tomoyuki Chikanaga)

bug5754 .rb の中で t2 が Thread.current になっているようだったのと、scratch の配列の順序が違っているようだったので少し変更してみました。

```
diff --git a/test/ruby/test_require.rb b/test/ruby/test_require.rb
index 9186a6f..02f7efe 100644
--- a/test/ruby/test_require.rb
+++ b/test/ruby/test_require.rb
@@ -349,10 +349,11 @@ class TestRequire < Test::Unit::TestCase
tmp = Tempfile.new(%w"bug5754 .rb")
path = tmp.path
tmp.print <<-EOS
+Thread.pass until TestRequire.scratch[1]
TestRequire.scratch << :pre
-Thread.pass until t2 = TestRequire.scratch[1]
+t2 = TestRequire.scratch[0,2].reject{|t| t == Thread.current}[0]
Thread.pass until t2.stop?
-open(FILE, "w") {|f| f.puts "TestRequire.scratch << :post"}
+open(FILE, "w") {|f| f.puts "TestRequire.scratch << :post"; f.puts "t1,t2=TestRequire.scratch[0, 2];if Thread.current == t2; Thread.pass until
t1.stopped?; end"}
raise "con1"
EOS
tmp.close
@@ -364,6 +365,8 @@ raise "con1"
t2_res = nil

t1 = Thread.new do
```

- Thread.pass until t1
- [redacted]

```
begin
require(path)
rescue RuntimeError
@@ -376,6 +379,7 @@ raise "con1"
end

t2 = Thread.new do
```

- [redacted]

Thread.pass until scratch[0]
begin
scratch << t2
@@ -389,8 +393,8 @@ raise "con1"
assert_nothing_raised(ThreadError, bug5754) {t1.join}
assert_nothing_raised(ThreadError, bug5754) {t2.join}

- assert_equal(true, (t1_res ^ t2_res), bug5754)
- assert_equal([:pre, t2, :post, :t2, :t1], scratch, bug5754)
- assert_equal(true, (t1_res ^ t2_res), bug5754 + " t1:#{t1_res} t2:#{t2_res}")
- assert_equal([t1, t2, :pre, :post, :t2, :t1], scratch, bug5754)
ensure
tmp.close(true) if tmp
end

この状態で test_race_exception を実行すると以下のように ThreadError が(require のところで)発生するようになりました。

1) Failure:
test_race_exception(TestRequire) [/home/nagachika/opt/ruby-trunk/src/ruby/test/ruby/test_require.rb:386]:
.
Exception raised:
<#>.

とり急ぎご報告まで。

#2 - 12/20/2011 01:43 PM - naruse (Yui NARUSE)

追いかけた結果、どうも lock に用いている rb_barrier_release/rb_barrier_destroy や rb_mutex_lock のマルチスレッド対応がいまいちだったようです。
前者は rb_mutex_unlock を外した直後から、待っていた他のスレッドが動き始めるのでもはや mutex->cond_waiting は古い情報となってしまう点、
後者は GVL_UNLOCK_BEGIN() した直後から、メインのスレッドが動いてしまう可能性があるのがダメですね。

diff --git a/test/ruby/test_require.rb b/test/ruby/test_require.rb
index 9186a6f..f1d8d12 100644
--- a/test/ruby/test_require.rb
+++ b/test/ruby/test_require.rb
@@ -350,9 +350,18 @@ class TestRequire < Test::Unit::TestCase
path = tmp.path
tmp.print <<-EOS
TestRequire.scratch << :pre
-Thread.pass until t2 = TestRequire.scratch[1]
+TestRequire.scratch << Thread.current
+Thread.pass until t2 = TestRequire.scratch[2]
Thread.pass until t2.stop?
-open(**FILE**, "w") {|f| f.puts "TestRequire.scratch << :post"}

+open(**FILE**, "w") do |f|

- f.puts "t1, t2 = TestRequire.scratch[1, 2]"
- f.puts "if Thread.current == t2"
- f.puts " TestRequire.scratch << :post"
- f.puts " until t1.stop?"
- f.puts "   Thread.pass"
- f.puts " end"

- f.puts "end"
  +end
  raise "con1"
  EOS
  tmp.close
  @@ -368,7 +377,6 @@ raise "con1"
  require(path)
  rescue RuntimeError

end

t1_res = require(path)

Thread.pass until fin
@@ -376,7 +384,7 @@ raise "con1"
end

t2 = Thread.new do

- [redacted]

- [redacted]

begin
scratch << t2
t2_res = require(path)
@@ -389,8 +397,8 @@ raise "con1"
assert_nothing_raised(ThreadError, bug5754) {t1.join}
assert_nothing_raised(ThreadError, bug5754) {t2.join}

- assert_equal(true, (t1_res ^ t2_res), bug5754)

- assert_equal([:pre, t2, :post, :t2, :t1], scratch, bug5754)

- assert_equal(true, (t1_res ^ t2_res), bug5754 + " t1:#{t1_res} t2:#{t2_res}")

- assert_equal([:pre, t1, t2, :post, :t2, :t1], scratch, bug5754)
  ensure
  tmp.close(true) if tmp
  end
  diff --git a/thread.c b/thread.c
  index ced10c2..c46653b 100644
  --- a/thread.c
  +++ b/thread.c
  @@ -3409,7 +3409,6 @@ lock_func(rb_thread_t *th, rb_mutex_t *mutex, int timeout_ms)
  int interrupted = 0;
  int err = 0;

- mutex->cond_waiting++;
  for (;;) {
  if (!mutex->th) {
  mutex->th = th;
  @@ -3438,7 +3437,6 @@ lock_func(rb_thread_t *th, rb_mutex_t *mutex, int timeout_ms)
  err = 0;
  }
  }

- mutex->cond_waiting--;

return interrupted;
}

@@ -3493,10 +3491,12 @@ rb_mutex_lock(VALUE self)
if (vm_living_thread_num(th->vm) == th->vm->sleeper) {
timeout_ms = 100;
}

- [redacted]

```
GVL_UNLOCK_BEGIN();
interrupted = lock_func(th, mutex, timeout_ms);
native_mutex_unlock(&mutex->lock);
GVL_UNLOCK_END();
```

- [redacted]

```
reset_unblock_function(th, &oldubf);
```

@@ -3727,9 +3727,11 @@ rb_barrier_release(VALUE self)
{
VALUE mutex = GetBarrierPtr(self);
rb_mutex_t *m;

- rb_mutex_unlock(mutex);
- int waiting; GetMutexPtr(mutex, m);
- return m->cond_waiting > 0 ? Qtrue : Qfalse;
- waiting = m->cond_waiting;
- rb_mutex_unlock(mutex);
- return waiting > 0 ? Qtrue : Qfalse; }

/*
@@ -3740,10 +3742,12 @@ rb_barrier_destroy(VALUE self)
{
VALUE mutex = GetBarrierPtr(self);
rb_mutex_t *m;

- int waiting; DATA_PTR(self) = 0;
- rb_mutex_unlock(mutex); GetMutexPtr(mutex, m);
- return m->cond_waiting > 0 ? Qtrue : Qfalse;
- waiting = m->cond_waiting;
- rb_mutex_unlock(mutex);
- return waiting > 0 ? Qtrue : Qfalse; }

int

#3 - 12/20/2011 02:10 PM - kosaki (Motohiro KOSAKI)

- Category set to core

- Assignee changed from nobu (Nobuyoshi Nakada) to kosaki (Motohiro KOSAKI)

- Target version set to 2.0.0

lock_funcやrb_mutex_lockを変更するのは反対なので、いったん預かります

#4 - 12/20/2011 03:59 PM - nobu (Nobuyoshi Nakada)

- ruby -v changed from ruby 2.0.0dev (2011-12-16 trunk 34058) [x86_64-freebsd9.0] to -

なかだです。

(11/12/20 13:43), Yui NARUSE wrote:

> 追いかけた結果、どうも lock に用いている rb_barrier_release/rb_barrier_destroy や rb_mutex_lock のマルチスレッド対応がいまいちだったようです。
> 前者は rb_mutex_unlock を外した直後から、待っていた他のスレッドが動き始めるのでもはや mutex->cond_waiting は古い情報となってしまう点、
> 後者は GVL_UNLOCK_BEGIN() した直後から、メインのスレッドが動いてしまう可能性があるのがダメですね。

たしかに、rb_mutex_lockがGVLを先に開放していることを見落としていました。
rb_barrier_release/rb_barrier_destroyが呼ばれた時点ではmutexがロックされていることが前提ですので、後半のBarrier部分の変更だけで充分な気がします。

```
diff --git a/test/ruby/test_require.rb b/test/ruby/test_require.rb
index 9186a6f..f1d8d12 100644
--- a/test/ruby/test_require.rb
+++ b/test/ruby/test_require.rb
@@ -350,9 +350,18 @@ class TestRequire < Test::Unit::TestCase
path = tmp.path
tmp.print <<-EOS
TestRequire.scratch << :pre
-Thread.pass until t2 = TestRequire.scratch[1]
+TestRequire.scratch << Thread.current
+Thread.pass until t2 = TestRequire.scratch[2]
Thread.pass until t2.stop?
-open(FILE, "w") {|f| f.puts "TestRequire.scratch << :post"}
+open(FILE, "w") do |f|
+  f.puts "t1, t2 = TestRequire.scratch[1, 2]"
+  f.puts "if Thread.current == t2"
+  f.puts "  TestRequire.scratch << :post"
+  f.puts "  until t1.stop?"
+  f.puts "    Thread.pass"
+  f.puts "  end"
+  f.puts "end" +end raise "con1"
```

この変更ではt2でロードされることを想定していますが、最初のrequireが例外で中断したあとにt1,t2のどちらが実際にロードすることになるかは不定のはずです。

--
--- 僕の前にBugはない。
--- 僕の後ろにBugはできる。
中田 伸悦

#5 - 12/20/2011 04:55 PM - naruse (Yui NARUSE)

Nobuyoshi Nakada wrote:

> rb_barrier_release/rb_barrier_destroyが呼ばれた時点ではmutexがロックされていることが前提ですので、後半のBarrier部分の変更だけで充分な気がします。

その前提は成立しません。

このテストにおいて、t1 は t2.stop? == true を待ってから、例外によって require を抜けます。
言い換えると、t2 が rb_barrier_wait 内の rb_mutex_lock 内の native_cond_wait に到達するのを待ってから、
例外によって require を例外を抜けたいと思っています。
最低限のラインとしては、mutex->cond_waiting++ されてから抜けないといけません。

さて、t2 における rb_mutex_lock ですが、
thread.c の 3481行目に th->status = THREAD_STOPPED_FOREVER という行があります。
これ以降、Thread#stop?はtrueを返すようになりますが、まだ GVL にとって他のスレッドは動けないので大丈夫です。
そこから10行ほど下ると、GVL_UNLOCK_BEGIN();という行があります。
ここから他のスレッドが動けるようになります、まだこのスレッドは STOP してないのに。
t1 がこの状況で動くと、例外を投げて require を抜け、load_unlock が呼ばれます。
つまりこの時点で前提は崩れます。
そして rb_barrier_release を呼び、まだ 0 のままのmutex->cond_waiting を見て Qfalse を返すため、
loading_tbl から当該パスは削除されてしまいます。
一方、t1 はそれと平行して lock_funcに入り、やっとmutex->cond_waiting++します。
30行ほど下って、native_cond_waitで待ちますが、上記のケースの場合この時にはもう手遅れになっています。

まとめると、GVL_UNLOCK_BEGINから、mutex->cond_waiting++の間に、他のスレッドが動いて、
mutex->cond_waitingを見ると、0のままであり、待ってる人がいないと思うので、
待ち人用の何かを破壊したりすると、待ってた人が発狂する。
具体的には、load.c の load_unlock の rb_barrier_destroy/rb_barrier_destroy と、st_delete あたり、というわけです。

```
diff --git a/test/ruby/test_require.rb b/test/ruby/test_require.rb
index 9186a6f..f1d8d12 100644
--- a/test/ruby/test_require.rb
+++ b/test/ruby/test_require.rb
@@ -350,9 +350,18 @@ class TestRequire < Test::Unit::TestCase
path = tmp.path
tmp.print <<-EOS
TestRequire.scratch << :pre
-Thread.pass until t2 = TestRequire.scratch[1]
+TestRequire.scratch << Thread.current
+Thread.pass until t2 = TestRequire.scratch[2]
Thread.pass until t2.stop?
-open(FILE, "w") {|f| f.puts "TestRequire.scratch << :post"}
+open(FILE, "w") do |f|
+  f.puts "t1, t2 = TestRequire.scratch[1, 2]"
+  f.puts "if Thread.current == t2"
+  f.puts "  TestRequire.scratch << :post"
+  f.puts "  until t1.stop?"
+  f.puts "    Thread.pass"
+  f.puts "  end"
+  f.puts "end" +end raise "con1"
```

> この変更ではt2でロードされることを想定していますが、最初のrequireが例外で中断したあとにt1,t2のどちらが実際にロードすることになるかは不定のはずです。

あぁ、なるほど。なんで :post が含まれないケースがあるのかなぁと思っていました。

#6 - 01/05/2012 10:58 AM - naruse (Yui NARUSE)

とりあえず r34163 で mutex->cond_waiting を用いるのをやめ、RBASIC()->flags に待っているスレッドの数を保存するように変更しました。
これで、問題自体は直ったはずですし、現在まで再現していません。
他の方法で実装し直したいと仰っていたので、小崎さんにアサインされたままにはしておきます。

#7 - 11/26/2012 08:05 PM - kosaki (Motohiro KOSAKI)

- Status changed from Assigned to Closed

さすがに時期的に、書き直しはあきらめるべきなので、close