Ruby master - Bug #14999

ConditionVariable doesn't reacquire the Mutex if Thread#kill-ed

08/16/2018 12:59 PM - Eregon (Benoit Daloze)

Status:	Closed		
Priority:	Normal		
Assignee:	normalperson (Eric Wong)		
Target version:			
ruby -v:	ruby 2.6.0dev (2018-08-16 trunk 64394) [x86_64-linux]	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

These specs reproduce the issue:
https://raw.githubusercontent.com/oracle/truffleruby/master/spec/ruby/library/conditionvariable/wait_spec.rb

I can commit them, but of course they will fail.
Feel free to just copy it to MRI's spec/ruby/library/conditionvariable/wait_spec.rb
(I would do it when synchronizing specs anyway)

I'd guess it's caused by the recent timer thread changes or so.
This spec works fine on 2.5.1 and older.

Just copy-pasting the spec out allows for a standalone reproducer:

```
m = Mutex.new
cv = ConditionVariable.new
in_synchronize = false
owned = nil

th = Thread.new do
  m.synchronize do
    in_synchronize = true
    begin
      cv.wait(m)
    ensure
      owned = m.owned?
      $stderr.puts "\nThe Thread doesn't own the Mutex!" unless owned
    end
  end
end

# wait for m to acquire the mutex
Thread.pass until in_synchronize
# wait until th is sleeping (ie waiting)
Thread.pass while th.status and th.status != "sleep"

m.synchronize {
  cv.signal
  # Wait that the thread is blocked on acquiring the Mutex
  sleep 0.001
  # Kill the thread, yet the thread should first acquire the Mutex before going on
  th.kill
}

th.join
```

Result on trunk:

```
The Thread doesn't own the Mutex!
#<Thread:0x0000565216f4ba28@cv_bug.rb:6 aborting> terminated with exception (report_on_exception i
s true):
Traceback (most recent call last):
    1: from cv_bug.rb:7:in `block in <main>'
```

```
cv_bug.rb:7:in `synchronize': Attempt to unlock a mutex which is not locked (ThreadError)
Traceback (most recent call last):
    1: from cv_bug.rb:7:in `block in <main>'
cv_bug.rb:7:in `synchronize': Attempt to unlock a mutex which is not locked (ThreadError)
```

Associated revisions

Revision 2cf3bd5b - 08/16/2018 07:59 PM - normal

thread_sync.c (rb_mutex_lock): acquire lock before being killed

We (the thread acquiring the mutex) need to acquire the mutex
before being killed to work with ConditionVariable#wait.

Thus we reinstate the acquire-immediately-after-sleeping logic
from pre-r63711 while still retaining the
acquire-after-checking-for-interrupts logic from r63711.

This regression was introduced in
commit 501069b8a4013f2e3fdde35c50e9527ef0061963 (r63711)
("thread_sync.c (rb_mutex_lock): fix deadlock") for
[Bug #14841]

[ruby-core:88503] [Bug #14999] [Bug #14841]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64398 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64398 - 08/16/2018 07:59 PM - normalperson (Eric Wong)

thread_sync.c (rb_mutex_lock): acquire lock before being killed

We (the thread acquiring the mutex) need to acquire the mutex
before being killed to work with ConditionVariable#wait.

Thus we reinstate the acquire-immediately-after-sleeping logic
from pre-r63711 while still retaining the
acquire-after-checking-for-interrupts logic from r63711.

This regression was introduced in
commit 501069b8a4013f2e3fdde35c50e9527ef0061963 (r63711)
("thread_sync.c (rb_mutex_lock): fix deadlock") for
[Bug #14841]

[ruby-core:88503] [Bug #14999] [Bug #14841]

Revision 64398 - 08/16/2018 07:59 PM - normal

thread_sync.c (rb_mutex_lock): acquire lock before being killed

We (the thread acquiring the mutex) need to acquire the mutex
before being killed to work with ConditionVariable#wait.

Thus we reinstate the acquire-immediately-after-sleeping logic
from pre-r63711 while still retaining the
acquire-after-checking-for-interrupts logic from r63711.

This regression was introduced in
commit 501069b8a4013f2e3fdde35c50e9527ef0061963 (r63711)
("thread_sync.c (rb_mutex_lock): fix deadlock") for
[Bug #14841]

[ruby-core:88503] [Bug #14999] [Bug #14841]

Revision 3993cd80 - 08/17/2018 09:51 AM - Eregon (Benoit Daloze)

Integrate new specs for ConditionVariable#wait to prevent regressions

- See [Bug #14999].

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64409 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64409 - 08/17/2018 09:51 AM - Eregon (Benoit Daloze)

Integrate new specs for ConditionVariable#wait to prevent regressions

- See [Bug #14999].

Revision 64409 - 08/17/2018 09:51 AM - Eregon (Benoit Daloze)

Integrate new specs for ConditionVariable#wait to prevent regressions

- See [Bug #14999].

Revision d7ddbff2 - 08/18/2018 04:24 AM - normal

thread_sync.c (do_sleep): avoid thread-switch/interrupt check

Calling rb_mutex_sleep directly should avoid
thread-switching/interrupt checking which can lead to occasional
failures.

Unfortunately, this means overriding Mutex#sleep is no longer
supported. Will let this commit run for a bit see if CI failures
from ConditionVariable specs continue...

cf. https://rubyci.org/logs/rubyci.s3.amazonaws.com/ubuntu/ruby-trunk/log/20180817T213003Z.fail.html.gz

[ruby-core:88524] [Bug #14999]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64436 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64436 - 08/18/2018 04:24 AM - normalperson (Eric Wong)

thread_sync.c (do_sleep): avoid thread-switch/interrupt check

Calling rb_mutex_sleep directly should avoid
thread-switching/interrupt checking which can lead to occasional
failures.

Unfortunately, this means overriding Mutex#sleep is no longer
supported. Will let this commit run for a bit see if CI failures
from ConditionVariable specs continue...

cf. https://rubyci.org/logs/rubyci.s3.amazonaws.com/ubuntu/ruby-trunk/log/20180817T213003Z.fail.html.gz

[ruby-core:88524] [Bug #14999]

Revision 64436 - 08/18/2018 04:24 AM - normal

thread_sync.c (do_sleep): avoid thread-switch/interrupt check

Calling rb_mutex_sleep directly should avoid
thread-switching/interrupt checking which can lead to occasional
failures.

Unfortunately, this means overriding Mutex#sleep is no longer
supported. Will let this commit run for a bit see if CI failures
from ConditionVariable specs continue...

cf. https://rubyci.org/logs/rubyci.s3.amazonaws.com/ubuntu/ruby-trunk/log/20180817T213003Z.fail.html.gz

[ruby-core:88524] [Bug #14999]

Revision c742050e - 08/18/2018 01:52 PM - Eregon (Benoit Daloze)

Revert r64441

- This reverts commit 647fc1227a4146ecbfeb0d59358abc8d99cd8ae6: "thread_sync.c (rb_mutex_synchronize): only unlock if we own the mutex"
- Let's try to preserve the semantics of always being locked inside Mutex#synchronize, even if an exception interrupts ConditionVariable#wait.
- As discussed on [Bug #14999].

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64448 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64448 - 08/18/2018 01:52 PM - Eregon (Benoit Daloze)

Revert r64441

- This reverts commit 647fc1227a4146ecbfeb0d59358abc8d99cd8ae6: "thread_sync.c (rb_mutex_synchronize): only unlock if we own the mutex"
- Let's try to preserve the semantics of always being locked inside Mutex#synchronize, even if an exception interrupts ConditionVariable#wait.
- As discussed on [Bug #14999].

Revision 64448 - 08/18/2018 01:52 PM - Eregon (Benoit Daloze)

Revert r64441

- This reverts commit 647fc1227a4146ecbfeb0d59358abc8d99cd8ae6: "thread_sync.c (rb_mutex_synchronize): only unlock if we own the mutex"
- Let's try to preserve the semantics of always being locked inside Mutex#synchronize, even if an exception interrupts ConditionVariable#wait.
- As discussed on [Bug #14999].

Revision d2afeb94 - 08/19/2018 12:01 AM - normal

thread_pthread.c: reset timeslice delay when uncontended

This matches the behavior of old timer thread more closely
and seems to fix [Bug #14999] when limited to a single CPU.
I cannot reproduce the error on a multi-core system unless
I use schedtool to force affinity to a single CPU:

schedtool -a 0x01 -e make test-spec \
MSPECOPT='-R1000 spec/ruby/library/conditionvariable/wait_spec.rb'

While it may be good enough to pass the spec, I don't have
huge degree of confidence in the interrupt handling robustness
under extremely heavy load (these may be ancient bugs, though).

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64467 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64467 - 08/19/2018 12:01 AM - normalperson (Eric Wong)

thread_pthread.c: reset timeslice delay when uncontended

This matches the behavior of old timer thread more closely
and seems to fix [Bug #14999] when limited to a single CPU.
I cannot reproduce the error on a multi-core system unless
I use schedtool to force affinity to a single CPU:

schedtool -a 0x01 -e make test-spec \
MSPECOPT='-R1000 spec/ruby/library/conditionvariable/wait_spec.rb'

While it may be good enough to pass the spec, I don't have
huge degree of confidence in the interrupt handling robustness
under extremely heavy load (these may be ancient bugs, though).

Revision 64467 - 08/19/2018 12:01 AM - normal

thread_pthread.c: reset timeslice delay when uncontended

This matches the behavior of old timer thread more closely
and seems to fix [Bug #14999] when limited to a single CPU.
I cannot reproduce the error on a multi-core system unless
I use schedtool to force affinity to a single CPU:

schedtool -a 0x01 -e make test-spec \
MSPECOPT='-R1000 spec/ruby/library/conditionvariable/wait_spec.rb'

While it may be good enough to pass the spec, I don't have
huge degree of confidence in the interrupt handling robustness
under extremely heavy load (these may be ancient bugs, though).

Revision d9cb9017 - 08/19/2018 10:20 PM - normal

thread_sync.c (rb_mutex_sleep): disable interrupt checking in ensure

This is needed to reliably fix ConditionVariable#wait on Thread#kill
[Bug #14999] because there is still a chance an interrupt could fire
and prevent lock acquisition after an ensure statement.

Arguably, rb_mutex_lock itself should be uninterruptible, but that
already prevents bootstraptest/test_thread.rb from completing and
probably breaks existing use cases.

For reference, POSIX expressly forbids EINTR from pthread_mutex_lock.

[ruby-core:88556] [Bug #14999]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64476 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64476 - 08/19/2018 10:20 PM - normalperson (Eric Wong)

thread_sync.c (rb_mutex_sleep): disable interrupt checking in ensure

This is needed to reliably fix ConditionVariable#wait on Thread#kill
[Bug #14999] because there is still a chance an interrupt could fire
and prevent lock acquisition after an ensure statement.

Arguably, rb_mutex_lock itself should be uninterruptible, but that
already prevents bootstraptest/test_thread.rb from completing and
probably breaks existing use cases.

For reference, POSIX expressly forbids EINTR from pthread_mutex_lock.

[ruby-core:88556] [Bug #14999]

Revision 64476 - 08/19/2018 10:20 PM - normal

thread_sync.c (rb_mutex_sleep): disable interrupt checking in ensure

This is needed to reliably fix ConditionVariable#wait on Thread#kill
[Bug #14999] because there is still a chance an interrupt could fire
and prevent lock acquisition after an ensure statement.

Arguably, rb_mutex_lock itself should be uninterruptible, but that
already prevents bootstraptest/test_thread.rb from completing and
probably breaks existing use cases.

For reference, POSIX expressly forbids EINTR from pthread_mutex_lock.

[ruby-core:88556] [Bug #14999]

Revision c561f04b - 10/11/2018 02:40 PM - nagachika (Tomoyuki Chikanaga)

merge revision(s) 63711,64398: [Backport #14841]

```
    thread_sync.c (rb_mutex_lock): fix deadlock

    * thread_sync.c (rb_mutex_lock): fix deadlock
      [ruby-core:87467] [Bug #14841]

    thread_sync.c (rb_mutex_lock): acquire lock before being killed

    We (the thread acquiring the mutex) need to acquire the mutex
    before being killed to work with ConditionVariable#wait.

    Thus we reinstate the acquire-immediately-after-sleeping logic
    from pre-r63711 while still retaining the
    acquire-after-checking-for-interrupts logic from r63711.

    This regression was introduced in
    commit 501069b8a4013f2e3fdde35c50e9527ef0061963 (r63711)
    ("thread_sync.c (rb_mutex_lock): fix deadlock") for
    [Bug #14841]

    [ruby-core:88503] [Bug #14999] [Bug #14841]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_5@64998 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64998 - 10/11/2018 02:40 PM - nagachika (Tomoyuki Chikanaga)

merge revision(s) 63711,64398: [Backport #14841]

```
thread_sync.c (rb_mutex_lock): fix deadlock

* thread_sync.c (rb_mutex_lock): fix deadlock
  [ruby-core:87467] [Bug #14841]

thread_sync.c (rb_mutex_lock): acquire lock before being killed

We (the thread acquiring the mutex) need to acquire the mutex
before being killed to work with ConditionVariable#wait.

Thus we reinstate the acquire-immediately-after-sleeping logic
from pre-r63711 while still retaining the
acquire-after-checking-for-interrupts logic from r63711.
```

```
This regression was introduced in
commit 501069b8a4013f2e3fdde35c50e9527ef0061963 (r63711)
("thread_sync.c (rb_mutex_lock): fix deadlock") for
[Bug #14841]

[ruby-core:88503] [Bug #14999] [Bug #14841]
```

Revision f5b0b984 - 10/17/2018 08:40 AM - usa (Usaku NAKAMURA)

merge revision(s) 63711,64398: [Backport #14841]

```
    thread_sync.c (rb_mutex_lock): fix deadlock

    * thread_sync.c (rb_mutex_lock): fix deadlock
      [ruby-core:87467] [Bug #14841]

    thread_sync.c (rb_mutex_lock): acquire lock before being killed

    We (the thread acquiring the mutex) need to acquire the mutex
    before being killed to work with ConditionVariable#wait.

    Thus we reinstate the acquire-immediately-after-sleeping logic
    from pre-r63711 while still retaining the
    acquire-after-checking-for-interrupts logic from r63711.

    This regression was introduced in
    commit 501069b8a4013f2e3fdde35c50e9527ef0061963 (r63711)
    ("thread_sync.c (rb_mutex_lock): fix deadlock") for
    [Bug #14841]

    [ruby-core:88503] [Bug #14999] [Bug #14841]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_4@65115 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 65115 - 10/17/2018 08:40 AM - usa (Usaku NAKAMURA)

merge revision(s) 63711,64398: [Backport #14841]

```
thread_sync.c (rb_mutex_lock): fix deadlock

* thread_sync.c (rb_mutex_lock): fix deadlock
  [ruby-core:87467] [Bug #14841]

thread_sync.c (rb_mutex_lock): acquire lock before being killed

We (the thread acquiring the mutex) need to acquire the mutex
before being killed to work with ConditionVariable#wait.

Thus we reinstate the acquire-immediately-after-sleeping logic
from pre-r63711 while still retaining the
acquire-after-checking-for-interrupts logic from r63711.

This regression was introduced in
commit 501069b8a4013f2e3fdde35c50e9527ef0061963 (r63711)
("thread_sync.c (rb_mutex_lock): fix deadlock") for
[Bug #14841]

[ruby-core:88503] [Bug #14999] [Bug #14841]
```

Revision 9ab19d79 - 01/16/2019 12:04 PM - nagachika (Tomoyuki Chikanaga)

merge revision(s) 64476: [Backport #15084]

```
    thread_sync.c (rb_mutex_sleep): disable interrupt checking in ensure

    This is needed to reliably fix ConditionVariable#wait on Thread#kill
    [Bug #14999] because there is still a chance an interrupt could fire
    and prevent lock acquisition after an ensure statement.

    Arguably, rb_mutex_lock itself should be uninterruptible, but that
    already prevents bootstraptest/test_thread.rb from completing and
    probably breaks existing use cases.

    For reference, POSIX expressly forbids EINTR from pthread_mutex_lock.
```

```
    [ruby-core:88556] [Bug #14999]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_5@66837 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 66837 - 01/16/2019 12:04 PM - nagachika (Tomoyuki Chikanaga)

merge revision(s) 64476: [Backport #15084]

```
thread_sync.c (rb_mutex_sleep): disable interrupt checking in ensure

This is needed to reliably fix ConditionVariable#wait on Thread#kill
[Bug #14999] because there is still a chance an interrupt could fire
and prevent lock acquisition after an ensure statement.

Arguably, rb_mutex_lock itself should be uninterruptible, but that
already prevents bootstraptest/test_thread.rb from completing and
probably breaks existing use cases.

For reference, POSIX expressly forbids EINTR from pthread_mutex_lock.

[ruby-core:88556] [Bug #14999]
```

History

#1 - 08/16/2018 01:00 PM - Eregon (Benoit Daloze)

- Description updated

#2 - 08/16/2018 08:12 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> Bug #14999: ConditionVariable doesn't require the Mutex if Thread#kill-ed
> https://bugs.ruby-lang.org/issues/14999

r64398

Thank you; it was actually a regression introduced while fixing
[Bug #14841]. I was going to investigate this today (same as
[ruby-core:88498]), anyways; but you saved me a bunch of time.

Btw, I haven't come up with a better reproducer for [Bug #14841]
in the test suite. It takes forever :/

#3 - 08/16/2018 11:57 PM - Anonymous

- Status changed from Open to Closed

Applied in changeset commit:ruby-git|2cf3bd5bb2a7c4724e528577d37a883fe80a1122.

---

thread_sync.c (rb_mutex_lock): acquire lock before being killed

We (the thread acquiring the mutex) need to acquire the mutex
before being killed to work with ConditionVariable#wait.

Thus we reinstate the acquire-immediately-after-sleeping logic
from pre-r63711 while still retaining the
acquire-after-checking-for-interrupts logic from r63711.

This regression was introduced in
commit 501069b8a4013f2e3fdde35c50e9527ef0061963 (r63711)
("thread_sync.c (rb_mutex_lock): fix deadlock") for
[Bug #14841]

[ruby-core:88503] [Bug #14999] [Bug #14841]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64398 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

#4 - 08/17/2018 09:40 AM - Eregon (Benoit Daloze)

- Subject changed from ConditionVariable doesn't require the Mutex if Thread#kill-ed to ConditionVariable doesn't reacquire the Mutex if Thread#kill-ed

#5 - 08/17/2018 11:54 PM - Eregon (Benoit Daloze)

normalperson (Eric Wong) I added the specs in r64409.
However I just saw that the spec failed once on Ubuntu:
https://rubyci.org/logs/rubyci.s3.amazonaws.com/ubuntu/ruby-trunk/log/20180817T213003Z.fail.html.gz
Any idea?

#6 - 08/18/2018 12:32 AM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> normalperson (Eric Wong) I added the specs in r64409.
> However I just saw that the spec failed once on Ubuntu:
> https://rubyci.org/logs/rubyci.s3.amazonaws.com/ubuntu/ruby-trunk/log/20180817T213003Z.fail.html.gz

It might be because of rb_funcallv switching and hitting
interrupts. I think this should fix it, but it can make
an incompatibility if somebody relies on redefining
Mutex#sleep...

```
diff --git a/thread_sync.c b/thread_sync.c
index 5e511af0db..87c17316a9 100644
--- a/thread_sync.c
+++ b/thread_sync.c
@@ -1358,7 +1358,7 @@ static VALUE
do_sleep(VALUE args)
{
struct sleep_call *p = (struct sleep_call *)args;
-    return rb_funcallv(p->mutex, id_sleep, 1, &p->timeout);
+    return rb_mutex_sleep(p->mutex, p->timeout);
}

static VALUE
```

#7 - 08/18/2018 04:42 AM - normalperson (Eric Wong)

Eric Wong wrote:

> eregontp@gmail.com wrote:
>
> > normalperson (Eric Wong) I added the specs in r64409.
> > However I just saw that the spec failed once on Ubuntu:
> > https://rubyci.org/logs/rubyci.s3.amazonaws.com/ubuntu/ruby-trunk/log/20180817T213003Z.fail.html.gz
>
> It might be because of rb_funcallv switching and hitting
> interrupts. I think this should fix it, but it can make
> an incompatibility if somebody relies on redefining
> Mutex#sleep...

Committed a version which only avoids switching on r64436
We'll see if CI alerts are quieter, now.

However, I don't know what to do about Mutex_m since that still
hits the funcall path. We can't blindly mask out interrupts
because we need Mutex#sleep to react to Thread#run.

Also, please don't add similar specs for Mutex_m, yet.

#8 - 08/18/2018 06:32 AM - normalperson (Eric Wong)

Eric Wong wrote:

> Committed a version which only avoids switching on r64436
> We'll see if CI alerts are quieter, now.

No good. Now testing:
https://80x24.org/spew/20180818062657.3424-1-e@80x24.org/raw
(should commit when done testing, but I need food :x)

Also, https://bugs.ruby-lang.org/issues/15002 should
fix spurious wakeups from ConditionVariable#wait which
might cause some spec failures under MJIT.

#9 - 08/18/2018 11:05 AM - Eregon (Benoit Daloze)

What was wrong with the first patch?
It looks good from a quick glance, although indeed it doesn't deal with Mutex_m.
Maybe Thread.handle_interrupt(Object => :on_blocking) (or the equivalent in C) would help?

> No good. Now testing:
> https://80x24.org/spew/20180818062657.3424-1-e@80x24.org/raw

I don't think that's OK for semantics.
IMHO, anything in the synchronize block should hold the Mutex, otherwise we break users' assumptions.
So even if there is an exception waking ConditionVariable#wait, I believe users expect to own the Mutex during that exception, until getting out of the synchronize.
Otherwise, it's very surprising if the ensure block sometimes has, sometimes doesn't have the Mutex and so can't reliably modify data structures protected by the Mutex.
I also believe this was the behavior on Ruby 2.5 and older, but I might be wrong (at least I couldn't observe the spec to fail in thousands of tries).

#10 - 08/18/2018 11:13 AM - Eregon (Benoit Daloze)

- Status changed from Closed to Open

#11 - 08/18/2018 11:42 AM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> What was wrong with the first patch?

I still saw CI failures with it:

http://ci.rvm.jp/results/trunk_gcc7@silicon-docker/1234097

In retrospect, it could've also been fixed with [Feature #15002]

> It looks good from a quick glance, although indeed it doesn't deal with Mutex_m.
> Maybe Thread.handle_interrupt(Object => :on_blocking) (or the equivalent in C) would help?

I considered that, but the problem might stem from blocking in
an unexpected place (because CV#wait is being spuriously woken).

> > No good. Now testing:
> > https://80x24.org/spew/20180818062657.3424-1-e@80x24.org/raw
>
> I don't think that's OK for semantics.
> IMHO, anything in the synchronize block should hold the Mutex, otherwise we break users' assumptions.
> So even if there is an exception waking ConditionVariable#wait, I believe users expect to own the Mutex during that exception, until getting out of the synchronize.
> Otherwise, it's very surprising if the ensure block sometimes has, sometimes doesn't have the Mutex and so can't reliably modify data structures protected by the Mutex.
> I also believe this was the behavior on Ruby 2.5 and older, but I might be wrong (at least I couldn't observe the spec to fail in thousands of tries).

OK, I didn't consider the ensure case. So maybe you can revert
r64441 (I'm going AFK for a while), because there's a chance
r64444 also fixed the issue.

Last alert email I got was 2018-08-18 08:59Z, so it's been a
while since I got an alert (this is from ko1's ruby-alerts
list, I don't know if you're on it).

#12 - 08/18/2018 01:50 PM - Eregon (Benoit Daloze)

OK, I'll revert r64441 since I think we should preserve the semantics of always being locked inside Mutex#synchronize, even if an exception interrupts ConditionVariable#wait.

Let's keep thinking about how to implement that correctly.
Mutex_m would be nice to have correct too, but I think it's less critical as I suspect very few people use it.

FWIW, we're trying to optimize ConditionVariable in TruffleRuby and it's proving quite tricky to:

- not miss ConditionVariable#signal / ConditionVariable#broadcast
- handles other interrupts in #wait
- make sure to reacquire the Mutex when going out of #wait

- still handle interrupts/thread switching in the re-acquire

Maybe the original Ruby code can be an inspiration here: see lib/thread.rb at r42801 or
https://github.com/ruby/ruby/blob/324df61eea3e4c107e419ea3c685eaea4da7fd5e/lib/thread.rb#L65-L81

#13 - 08/18/2018 01:54 PM - Eregon (Benoit Daloze)

- Status changed from Open to Closed

Applied in changeset trunk|r64448.

Revert r64441

- This reverts commit 647fc1227a4146ecbfeb0d59358abc8d99cd8ae6: "thread_sync.c (rb_mutex_synchronize): only unlock if we own the mutex"
- Let's try to preserve the semantics of always being locked inside Mutex#synchronize, even if an exception interrupts ConditionVariable#wait.
- As discussed on [Bug #14999].

#14 - 08/18/2018 01:54 PM - Eregon (Benoit Daloze)

- Status changed from Closed to Open

#15 - 08/18/2018 03:56 PM - Eregon (Benoit Daloze)

It failed for http://ci.rvm.jp/results/trunk_gcc6@silicon-docker/1235550

#16 - 08/18/2018 07:35 PM - Eregon (Benoit Daloze)

- Status changed from Open to Closed

Applied in changeset commit:ruby-git|c742050ea5fd30108f913383c0fafc4614adb04c.

Revert r64441

- This reverts commit 647fc1227a4146ecbfeb0d59358abc8d99cd8ae6: "thread_sync.c (rb_mutex_synchronize): only unlock if we own the mutex"
- Let's try to preserve the semantics of always being locked inside Mutex#synchronize, even if an exception interrupts ConditionVariable#wait.
- As discussed on [Bug #14999].

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64448 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

#17 - 08/18/2018 09:22 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> It failed for http://ci.rvm.jp/results/trunk_gcc6@silicon-docker/1235550

Maybe wishful thinking, but r64464 might be the right fix.

Time for Me#sleep

#18 - 08/19/2018 12:13 AM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

> Maybe wishful thinking, but r64464 might be the right fix.

Seems it's not yet fully fixed unfortunately:
http://ci.rvm.jp/results/trunk_clang_39@silicon-docker/1236189
http://ci.rvm.jp/results/trunk_gcc5@silicon-docker/1236243
http://ci.rvm.jp/results/trunk-asserts-nopara@silicon-docker/1236250
http://ci.rvm.jp/results/trunk_gcc5@silicon-docker/1236262
http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1236304

I think we need to give some thinking on this,
and I don't want to stress you to fix it to fix the build.
(although this should be solved soon IMHO, latest before the next preview/RC)

So I propose to "tag" the spec so it won't run in CI, so you can focus on it when there is time.
I'll do that tomorrow if you agree.

It would be also be interesting to see if Ruby <= 2.5 actually ensured the semantics I described above.

#19 - 08/19/2018 10:04 AM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> I think we need to give some thinking on this,
> and I don't want to stress you to fix it to fix the build.
> (although this should be solved soon IMHO, latest before the next preview/RC)

It needs to be fixed ASAP.

r64467 seems to make CI happy, at least (along with r64398,
which was an absolutely necessary fix). The key difference with
r64467 fix is I figured out I needed to reproduce the failure
reliably by using a single CPU (schedtool -a 0x1 -e ...)

> So I propose to "tag" the spec so it won't run in CI, so you can focus on it when there is time.
> I'll do that tomorrow if you agree.

Please don't. CI is happy at the moment, at least.

> It would be also be interesting to see if Ruby <= 2.5 actually
> ensured the semantics I described above.

I was able to reproduce the problem with the old
timer-thread using "schedtool -a 0x1".

"git bisect" points to r63498 ("thread_pthread.c: enable thread
cache by default"), which is HIGHLY unexpected.

I'm not seeing how thread caching can be the cause of the bug,
but rather it makes an existing bug more apparent by being
faster and hitting the race sooner.

The thread cache only operates at the pthreads level, so there's
no way any Ruby-level interrupt could hit across Ruby Thread
lifetimes. pthread_kill is never used in the
Mutex/ConditionVariable code paths; and different pthreads
condvars are used for sleeping/wakeups. So it is a mystery as
to how thread caching ends up affecting Ruby-level interrupt
handling...

What thread-caching does do is make the MSPECOPT "-R" option
noticeably faster

Fwiw, I can still reproduce the failures with low timeouts:

```
diff --git a/thread_pthread.c b/thread_pthread.c
index 2fd60ddd4a..e8da3ee9c2 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
@@ -187,7 +187,7 @@ do_gvl_timer(rb_vm_t *vm, rb_thread_t *th)

if (vm->gvl.timer_err == ETIMEDOUT) {
ts.tv_sec = 0;
-         ts.tv_nsec = TIME_QUANTUM_NSEC;
+         ts.tv_nsec = 1;
ts = native_cond_timeout(&nd->cond.gvlq, ts);
}
vm->gvl.timer = th;
```

So there's definitely more to investigate until I'm satisified
with the reliability of this (but I'm too tired, now).

#20 - 08/19/2018 07:52 PM - normalperson (Eric Wong)

Eric Wong wrote:

> Fwiw, I can still reproduce the failures with low timeouts:
>
> ```
> diff --git a/thread_pthread.c b/thread_pthread.c
> index 2fd60ddd4a..e8da3ee9c2 100644
> --- a/thread_pthread.c
> ```

```
+++ b/thread_pthread.c
@@ -187,7 +187,7 @@ do_gvl_timer(rb_vm_t *vm, rb_thread_t *th)

     if (vm->gvl.timer_err == ETIMEDOUT) {
         ts.tv_sec = 0;
-        ts.tv_nsec = TIME_QUANTUM_NSEC;
+        ts.tv_nsec = 1;
         ts = native_cond_timeout(&nd->cond.gvlq, ts);
     }
     vm->gvl.timer = th;
```

> So there's definitely more to investigate until I'm satisified
> with the reliability of this (but I'm too tired, now).

I think the only way to work reliably is to disable interrupt
checking when attempting to lock a mutex in ensure:

https://80x24.org/spew/20180819193954.13807-1-e@80x24.org/raw

We can rely on deadlock checking to prevent problems; but it
could still affect user-experience if Ctrl-C takes a while
shrug

Currently testing, will take a while.

#21 - 08/19/2018 09:42 PM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

> r64467 seems to make CI happy, at least

Great!

> I figured out I needed to reproduce the failure
> reliably by using a single CPU (schedtool -a 0x1 -e ...)

I could also verify it fails before your commit, and passes after with:

```
taskset -c 1 mspec-run -R1000 library/conditionvariable/wait_spec.rb
```

> "git bisect" points to r63498 ("thread_pthread.c: enable thread
> cache by default"), which is HIGHLY unexpected.

So it seems it doesn't happen in Ruby <= 2.5 due to thread caching being disabled then?

> I'm not seeing how thread caching can be the cause of the bug,
> but rather it makes an existing bug more apparent by being
> faster and hitting the race sooner.

No idea either, but I guess a bug could lurk in there.
I did experience a few weird bugs when doing thread caching in TruffleRuby
(e.g., inline caches based on pthread_self() are incorrect as multiple Ruby threads could use the same pthread thread)

> What thread-caching does do is make the MSPECOPT "-R" option
> noticeably faster

Right, because creating threads is so much faster.
Maybe the fact pthread startup is done each time has some effect on races
(e.g., the pthread thread has a perfect view of the caller thread when starting,
but not true with caching, but maybe the GVL still ensure that)

> I think the only way to work reliably is to disable interrupt
> checking when attempting to lock a mutex in ensure:

At least if the interrupt causes to escape the function (e.g., Thread#raise/Thread#kill),
then that escape must be delayed after re-acquiring the Mutex.

In normal Mutex#lock and Mutex#synchronize, it's fine to raise/kill before taking the Mutex,

because it's before entering the critical section, but unfortunately here we are inside the critical section.

> it could still affect user-experience if Ctrl-C takes a while

When would that happen?
If some other Thread holds the Mutex too long?
It's probably never a good idea to hold a Mutex for seconds or a long time, especially when using a ConditionVariable.
It seems fair enough to delay the interrupt in such a case, as the main Thread is waiting for the Mutex.
Maybe we could warn if re-acquiring takes too long and it becomes a problem in practice (I think it won't).

#22 - 08/19/2018 11:12 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> normalperson (Eric Wong) wrote:
>
> > "git bisect" points to r63498 ("thread_pthread.c: enable thread cache by default"), which is HIGHLY unexpected.
>
> So it seems it doesn't happen in Ruby <= 2.5 due to thread caching being disabled then?

Right; but that might just be "luck"...

> > I'm not seeing how thread caching can be the cause of the bug, but rather it makes an existing bug more apparent by being faster and hitting the race sooner.
>
> No idea either, but I guess a bug could lurk in there. I did experience a few weird bugs when doing thread caching in TruffleRuby (e.g., inline caches based on pthread_self() are incorrect as multiple Ruby threads could use the same pthread thread)

At least for the core VM and standard library; I don't think pthread_self is a factor; and we already clobber our TSD pointer since r63836

> > What thread-caching does do is make the MSPECOPT "-R" option noticeably faster
>
> Right, because creating threads is so much faster. Maybe the fact pthread startup is done each time has some effect on races (e.g., the pthread thread has a perfect view of the caller thread when starting, but not true with caching, but maybe the GVL still ensure that)

Yes, it might affect accounting done in the kernel; particularly when bound to a single CPU. And it still takes -R1000 to reliably reproduce the problem with a single CPU; -R500 was sometimes successful, even.

> > I think the only way to work reliably is to disable interrupt checking when attempting to lock a mutex in ensure:
>
> At least if the interrupt causes to escape the function (e.g., Thread#raise/Thread#kill), then that escape must be delayed after re-acquiring the Mutex.

Yes, so I've committed r64476

> In normal Mutex#lock and Mutex#synchronize, it's fine to raise/kill before taking the Mutex, because it's before entering the critical section, but unfortunately here we are inside the critical section.

So this is why pthread_mutex_lock is forbidden from returning
EINTR by POSIX.

Mutex#lock/synchronize should have been uninterruptible, too;
but it's too late to change that as it would break compatibility
with existing code (bootstraptest/test_thread.rb already breaks).

> > it could still affect user-experience if Ctrl-C takes a while
>
> When would that happen?
> If some other Thread holds the Mutex too long?

Yes, but I suppose it's not a real problem.

> It's probably never a good idea to hold a Mutex for seconds or
> a long time, especially when using a ConditionVariable.
> It seems fair enough to delay the interrupt in such a case, as
> the main Thread is waiting for the Mutex.

Agreed(*).

> Maybe we could warn if re-acquiring takes too long and it
> becomes a problem in practice (I think it won't).

We already have deadlock checking to alert users to real
problems, so it's probably not a problem in practice.

(*) Along those lines, I think the "lock" idiom for GVL is not
ideal for performance (kosaki rewrote the GVL for 1.9.3 to
optimize for contention as a result). I might try replacing
the GVL with a platform-independent scheduler which limits
parallelism to the data structures the GVL currently protects.

Note: This will NOT solve the parallelism problem which
exists in C Ruby; that is a much bigger task (but still
doable with a scheduler, and perhaps even more so).

#23 - 08/20/2018 01:30 PM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

> (*) Along those lines, I think the "lock" idiom for GVL is not
> ideal for performance (kosaki rewrote the GVL for 1.9.3 to
> optimize for contention as a result). I might try replacing
> the GVL with a platform-independent scheduler which limits
> parallelism to the data structures the GVL currently protects.

I don't follow, which data structures would the scheduler protect?
Internal VM data structures but not e.g. Ruby Hash?

> Note: This will NOT solve the parallelism problem which
> exists in C Ruby; that is a much bigger task (but still
> doable with a scheduler, and perhaps even more so).

Now I am curious, how would a scheduler solve the
lack of parallelism at the Ruby level in CRuby?

I'm particularly interested since I looked at concurrency in Ruby
and specifically TruffleRuby during my PhD, and how to have GIL-like
guarantees and yet allow to run Ruby code in parallel.

#24 - 08/20/2018 08:12 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> normalperson (Eric Wong) wrote:
>
> > (*) Along those lines, I think the "lock" idiom for GVL is not

> ideal for performance (kosaki rewrote the GVL for 1.9.3 to
> optimize for contention as a result). I might try replacing
> the GVL with a platform-independent scheduler which limits
> parallelism to the data structures the GVL currently protects.

I don't follow, which data structures would the scheduler protect?
Internal VM data structures but not e.g. Ruby Hash?

Everything the GVL currently protects, including the Ruby Hash.

> Note: This will NOT solve the parallelism problem which
> exists in C Ruby; that is a much bigger task (but still
> doable with a scheduler, and perhaps even more so).

Now I am curious, how would a scheduler solve the
lack of parallelism at the Ruby level in CRuby?

It might allow cheaper entry/exit for blocking regions;
so we can use more blocking regions to achieve parallelism.

GVL release/acquire is expensive at the moment; so there's
places where we hold the GVL (e.g. readdir calls) and sacrifice
parallelism when dealing with slow filesystems.

Also, pthread_cond_wait is just like ConditionVariable#wait in
that it must re-lock the mutex after sleeping. This re-lock
can be contended, and it should not be necessary if we move
away from condvars for some platforms (Linux and win32).
The signaller (FUTEX_WAKE/SetEvent) can remove from the waitqueue
while it's holding the lock, so the sleeper won't have to.

Anyways, just a small idea in my head and totally unproven.

#25 - 09/05/2018 07:05 AM - larskanis (Lars Kanis)

> So it seems it doesn't happen in Ruby <= 2.5 due to thread
> caching being disabled then?

Right; but that might just be "luck"...

I get this error on ruby-2.4.4 [i386-mingw32] and [i386-mingw32] while nightly builds of RubyInstaller:
https://ci.appveyor.com/project/larskanis/rubyinstaller2-hbuor/build/1.0.722/job/dva7ug4vako5jhrl#L1295
It appears with a probability of around 50%.

#26 - 09/05/2018 12:18 PM - Eregon (Benoit Daloze)

larskanis (Lars Kanis) Thanks for the report.
I also noticed it happens on RubyCI for Ruby 2.4 and earlier
(example: https://rubyci.org/logs/rubyci.s3.amazonaws.com/arch/ruby-2.4/log/20180905T105615Z.fail.html.gz)

I'm not sure what's best.

normalperson (Eric Wong) Do you think we could backport the fix (or the logic behind the fix since the code changed quite a bit) to Ruby 2.4 and 2.3?
Interestingly enough, it seems to not fail on Ruby 2.5 at least in recent RubyCI builds.

For now, I'll mark the spec as ruby_bug on Ruby < 2.5.

#27 - 09/06/2018 05:42 PM - normalperson (Eric Wong)

lars@greiz-reinsdorf.de wrote:

> Issue #14999 has been updated by larskanis (Lars Kanis).
>
> > So it seems it doesn't happen in Ruby <= 2.5 due to thread
> > caching being disabled then?

> Right; but that might just be "luck"...

> I get this error on ruby-2.4.4 [i386-mingw32] and [i386-mingw32] while nightly builds of RubyInstaller:
> https://ci.appveyor.com/project/larskanis/rubyinstaller2-hbuor/build/1.0.722/job/dva7ug4vako5jhrl#L1295
> It appears with a probability of around 50%.

Thanks for the reminder (this bugfix already feels like a year ago):

Backport requested: https://bugs.ruby-lang.org/issues/15084