Backport193 - Backport #5130

Thread.pass sticks on OpenBSD

08/01/2011 03:51 PM - naruse (Yui NARUSE)

Status:	Closed	
Priority:	Normal	
Assignee:	yugui (Yuki Sonoda)	

Description

=begin
On OpenBSD 4.9, following script will stick.

./miniruby -ve'Thread.new{Thread.pass}'
=end

Associated revisions

Revision 871c6923 - 11/09/2011 05:10 PM - kosaki (Motohiro KOSAKI)

- thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield(). [Bug #5130] [ruby-core:38647]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@33693 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 33693 - 11/09/2011 05:10 PM - kosaki (Motohiro KOSAKI)

- thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield(). [Bug #5130] [ruby-core:38647]

Revision 33693 - 11/09/2011 05:10 PM - kosaki (Motohiro KOSAKI)

- thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield(). [Bug #5130] [ruby-core:38647]

Revision 33693 - 11/09/2011 05:10 PM - kosaki (Motohiro KOSAKI)

- thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield(). [Bug #5130] [ruby-core:38647]

Revision 33693 - 11/09/2011 05:10 PM - kosaki (Motohiro KOSAKI)

- thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield(). [Bug #5130] [ruby-core:38647]

Revision 33693 - 11/09/2011 05:10 PM - kosaki (Motohiro KOSAKI)

- thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield(). [Bug #5130] [ruby-core:38647]

Revision 33693 - 11/09/2011 05:10 PM - kosaki (Motohiro KOSAKI)

- thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield(). [Bug #5130] [ruby-core:38647]

Revision 8d82657a - 01/03/2012 12:09 AM - kosaki (Motohiro KOSAKI)

merge revision(s) 33693:

```
    * thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield().
      [Bug #5130] [ruby-core:38647]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_1_9_3@34179 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 34179 - 01/03/2012 12:09 AM - kosaki (Motohiro KOSAKI)

merge revision(s) 33693:

```
* thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield().
  [Bug #5130] [ruby-core:38647]
```

History

#1 - 08/03/2011 11:52 AM - kosaki (Motohiro KOSAKI)

- Category set to core

- Status changed from Open to Assigned

- Assignee set to kosaki (Motohiro KOSAKI)

#2 - 08/03/2011 12:15 PM - jeremyevans0 (Jeremy Evans)

Here's gdb output from attaching to a process (OpenBSD-current, not 4.9), and using ruby instead of miniruby:

```
$ ruby19 -e 'Thread.new{Thread.pass}'
$ gdb ruby19 17940
(gdb) bt
#0  0x0000000205b31da7 in pthread_cond_broadcast (cond=0x20b4bb040) at /usr/src/lib/libpthread/uthread/uthread_cond.c:655
#1  0x0000000204bb3709 in native_cond_broadcast () from /usr/local/lib/libruby19.so.1.91
#2  0x0000000204bb4424 in rb_thread_schedule_limits () from /usr/local/lib/libruby19.so.1.91
#3  0x0000000204bb476b in rb_thread_schedule () from /usr/local/lib/libruby19.so.1.91
#4  0x0000000204bb4a6e in rb_thread_terminate_all () from /usr/local/lib/libruby19.so.1.91
#5  0x0000000204abc8da in ruby_cleanup () from /usr/local/lib/libruby19.so.1.91
#6  0x0000000204abc996 in ruby_run_node () from /usr/local/lib/libruby19.so.1.91
#7  0x0000000000400c42 in main ()
(gdb) info threads
4 process 17940, thread 0x208e0f000 (RUNNING)  0x0000000205b31da7 in pthread_cond_broadcast (cond=0x20b4bb040) at
/usr/src/lib/libpthread/uthread/uthread_cond.c:655
3 process 17940, thread 0x2034be000 (SELECT_WAIT)  _thread_kern_sched (scp=0x0) at /usr/src/lib/libpthread/uthread/uthread_kern.c:495
2 process 17940, thread 0x206059000 (COND_WAIT)  _thread_kern_sched (scp=0x0) at /usr/src/lib/libpthread/uthread/uthread_kern.c:495
1 process 17940, thread 0x20a045000 (RUNNING)  _thread_kern_sched (scp=0x0) at /usr/src/lib/libpthread/uthread/uthread_kern.c:495
```

I'll try to spend some time debugging this tomorrow.

#3 - 08/03/2011 07:14 PM - kosaki (Motohiro KOSAKI)

- Status changed from Assigned to Feedback

#4 - 08/03/2011 07:23 PM - kosaki (Motohiro KOSAKI)

- ruby -v changed from ruby 1.9.4dev (2011-07-31 trunk 32788) [x86_64-openbsd4.9] to -

> $ ruby19 -e 'Thread.new{Thread.pass}'
> $ gdb ruby19 17940
> (gdb) bt
> #0 x0000000205b31da7 in pthread_cond_broadcast (cond=0x20b4bb040) at /usr/src/lib/libpthread/uthread/uthread_cond.c:655

I'm surprised this stack. pthread_cond_broadcast don't have to be hang
up even *if* the
argument is wrong. I suspect it's OpenBSD bug.
Jeremy, Could you please handle this issue?

> #1 x0000000204bb3709 in native_cond_broadcast () from /usr/local/lib/libruby19.so.1.91
> #2 x0000000204bb4424 in rb_thread_schedule_limits () from /usr/local/lib/libruby19.so.1.91
> #3 x0000000204bb476b in rb_thread_schedule () from /usr/local/lib/libruby19.so.1.91
> #4 x0000000204bb4a6e in rb_thread_terminate_all () from /usr/local/lib/libruby19.so.1.91

#5 - 08/04/2011 05:30 AM - jeremyevans0 (Jeremy Evans)

- File ruby193_thread_debug.log added

I built ruby with -DTHREAD_DEBUG, and am attaching a partial log for ruby19 -e 'Thread.new{Thread.pass}'. It's partial because it goes into an infinite loop at the end. I've included the first two iterations of the loop. I'll be doing more research on this issue, but if you have any suggestions, please let me know.

#6 - 08/04/2011 07:21 AM - jeremyevans0 (Jeremy Evans)

- File thread.c.diff added

Looking at the thread debug log I uploaded earlier, I guessed that this problem might be caused because a thread was starting after

rb_thread_terminate_all was called. rb_thread_terminate_all sets vm->inhibit_thread_creation = 1, which prohibits new threads from being created. However, it doesn't prevent created-but-not-yet-started threads from starting in thread_start_func_2. My theory is that thread_start_func_2 should return without starting a thread if vm->inhibit_thread_creation = 1.

The attached patch is a crude form of what I think should happen. It fixes the Thread.new{Thread.pass} case and the related bootstrap test. It currently passes all make check tests except for one. The one test failure is:

46) Error:
test_signal_requiring(TestSignal):
EOFError: end of file reached
/usr/obj/ports/ruby-1.9.3-preview1/ruby-1.9.3-preview1/test/ruby/test_signal.rb:218:in load'
/usr/obj/ports/ruby-1.9.3-preview1/ruby-1.9.3-preview1/test/ruby/test_signal.rb:218:inblock in test_signal_requiring'
/usr/obj/ports/ruby-1.9.3-preview1/ruby-1.9.3-preview1/test/ruby/test_signal.rb:204:in popen'
/usr/obj/ports/ruby-1.9.3-preview1/ruby-1.9.3-preview1/test/ruby/test_signal.rb:204:intest_signal_requiring'

The current patch is probably missing something small in terms of thread cleanup, but as I'm not familiar with the thread code, I'm not sure what that might be.

#7 - 08/24/2011 09:31 AM - normalperson (Eric Wong)

- File thread_2.diff added

Jeremy: I think your st_delete call is incorrect, can you try my updated patch?

I also rearranged rb_register_sigaltstack() to build with SIGALTSTACK.

#8 - 08/24/2011 11:23 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> File thread_2.diff added

I think both thread*.diff are racy since inhibit_thread_creation is
checked outside of GVL and I'm not sure if the idea with these
patches is correct.

Unfortunately I cannot reproduce this issue on Linux, but I would
add debug prints around loops where native_cond_wait() is called
in gvl_yield() and also before sched_yield().

I also remember sched_yield() doesn't work on some OSes, maybe try to
change that to nanosleep()/select()?

#9 - 08/24/2011 11:25 AM - jeremyevans0 (Jeremy Evans)

Eric Wong wrote:

> Jeremy: I think your st_delete call is incorrect, can you try my updated patch?
>
> I also rearranged rb_register_sigaltstack() to build with SIGALTSTACK.

No change after replacing my patch with yours, still getting the "end of file reached" EOFError on that signal test.

#10 - 11/03/2011 09:27 AM - jeremyevans0 (Jeremy Evans)

It's been a couple of months since this issue was last updated, and about three months since this issue was last updated by a ruby committer. Can a ruby committer look at either Eric's patch or my patch and let us know if this is an appropriate fix? I can't think of a reason why you would want to start a thread if the interpreter is shutting down, so the patch makes sense to me. Eric is right that this doesn't fix the race condition, but it should at least reduce the probability of it happening.

Looking at the thread log I sent earlier, the thread that starts after rb_thread_terminate_all is called calls rb_thread_schedule_limits, but gvl_yield apparently never changes the thread pointer, so it always yields to itself.

My theory is that when rb_thread_terminate_all calls terminate_i on a thread that has been created but not started, the thread never gets the terminate signal, so it continues to run indefinitely. So I think the patch makes sense. Alternatively thread_start_func_2 could check th->thrown_errinfo or th->status instead of GET_VM()->inhibit_thread_creation. I haven't tried that, but it could be a more general fix.

#11 - 11/07/2011 03:07 PM - naruse (Yui NARUSE)

- Status changed from Feedback to Assigned

- ruby -v changed from - to ruby 2.0.0dev (2011-11-07 trunk 33648) [x86_64-openbsd5.0]

#12 - 11/10/2011 01:54 AM - kosaki (Motohiro KOSAKI)

First, we can't apply your nor Eric's patch because it's racy. Your patch care following scenario.

1) thread-A create thread-B
2) thread-A call terminate_all
3) thread-B start

but, it doesn't care following scenario.

1) thread-A create thread-B
2) thread-B start
(just after)
3) thread-A call terminate_all

Moreover, we shouldn't need a special care about thread starting. it should works as well.
I don't plan to apply your ugly bandaid patch.

Second, your analysis is not accurate. The complete opnebsd failed scenario is here.
(OMG, I needed to install openbsd).

thread-A thread-B

-> gvl_yield
vm->gvl.wait_yield = 1;
-> sched_yield
-> gvl_yield
-> cond_wait(switch_wait_cond) # wait until waking up sched_yield()
-> task-state change to COND_WAIT
<- sched_yield
vm->gvl.wait_yield = 0;
-> pthread_cond_broadcast()
-> task-state of thread-B change to RUNNING
<- gvl_yield

-> gvl_yield
vm->gvl.wait_yield = 1;
-> sched_yield
<- cond_wait(switch_wait_cond)

```
                                                *** Oh, wait_yield is 1. try to sleep again.

                                                -> cond_wait(switch_wait_cond)
```

That's all.
rb_thread_terminate_all() called rb_thread_schedule() repeatedly. and waking-up cond_wait(switch_wait_cond) failed repeatedly. OpenBSD's user land pthread library makes semi deterministic scheduling. and then, the test code can't bail out forever.

btw, sched_yield() works as expected on OpenBSD. then, I don't change sched_yield() call at least now.

#13 - 11/10/2011 02:10 AM - kosaki (Motohiro KOSAKI)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r33693.
Yui, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield(). [Bug #5130] [ruby-core:38647]

#14 - 11/10/2011 02:12 AM - kosaki (Motohiro KOSAKI)

- Tracker changed from Bug to Backport

- Project changed from Ruby master to Backport193

- Category deleted (core)

- Status changed from Closed to Assigned

- Assignee changed from kosaki (Motohiro KOSAKI) to yugui (Yuki Sonoda)

Can anyone please review this patch?

#15 - 11/10/2011 02:31 AM - jeremyevans0 (Jeremy Evans)

Motohiro KOSAKI wrote:

> First, we can't apply your nor Eric's patch because it's racy. Your patch care following scenario.
>
> 1) thread-A create thread-B
> 2) thread-A call terminate_all
> 3) thread-B start
>
> but, it doesn't care following scenario.
>
> 1) thread-A create thread-B
> 2) thread-B start
> (just after)
> 3) thread-A call terminate_all

True. As my earlier message says, the patch only reduces the chance losing the race.

> Moreover, we shouldn't need a special care about thread starting. it should works as well.
> I don't plan to apply your ugly bandaid patch.

Fair enough. I certainly agree that my patch is an ugly bandaid.

> Second, your analysis is not accurate. The complete opnebsd failed scenario is here.
> (OMG, I needed to install openbsd).
>
> **thread-A** **thread-B**
>
> -> gvl_yield
> vm->gvl.wait_yield = 1;
> -> sched_yield
> -> gvl_yield
> -> cond_wait(switch_wait_cond) # wait until waking up sched_yield()
> -> task-state change to COND_WAIT
> <- sched_yield
> vm->gvl.wait_yield = 0;
> -> pthread_cond_broadcast()
> -> task-state of thread-B change to RUNNING
> <- gvl_yield
>
> -> gvl_yield
> vm->gvl.wait_yield = 1;
> -> sched_yield
> <- cond_wait(switch_wait_cond)
>
> *** Oh, wait_yield is 1. try to sleep again.
>
> -> cond_wait(switch_wait_cond)
>
> That's all.
> rb_thread_terminate_all() called rb_thread_schedule() repeatedly. and waking-up cond_wait(switch_wait_cond) failed repeatedly. OpenBSD's user land pthread library makes semi deterministic scheduling. and then, the test code can't bail out forever.

So the underlying problem is that wait_yield is always set to 1 when thread 2 wakes up, which causes thread 2 to sleep again? Is that a bug in ruby or OpenBSD?

> btw, sched_yield() works as expected on OpenBSD. then, I don't change sched_yield() call at least now.

That's good.

Thank you for taking the time to look into the issue in more detail. If I can be of any help, please let me know.

#16 - 11/10/2011 04:22 AM - jeremyevans0 (Jeremy Evans)

Motohiro KOSAKI wrote:

> Can anyone please review this patch?

I'm not qualified to determine correctness, but I can confirm that it fixes the threading bootstrap test failure on OpenBSD amd64 and i386.

#17 - 11/10/2011 07:08 AM - kosaki (Motohiro KOSAKI)

> So the underlying problem is that wait_yield is always set to 1 when thread 2 wakes up, which causes thread 2 to sleep again? Is that a bug in ruby or OpenBSD?

Difficult question. OpenBSD has a posix compliance pthread implementation. Ruby has reasonable OS assumption. But.... the result was pretty nasty unfortunate thing. Aghh.
Anyway, I don't think OpenBSD need to fix anything.

Thank you.

#18 - 01/03/2012 09:09 AM - kosaki (Motohiro KOSAKI)

- Status changed from Assigned to Closed

This issue was solved with changeset r34179.
Yui, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

---

merge revision(s) 33693:

```
* thread_pthread.c (gvl_yield): don't prevent concurrent sched_yield().
  [Bug #5130] [ruby-core:38647]
```

Files

ruby193_thread_debug.log	3.59 KB	08/04/2011	jeremyevans0 (Jeremy Evans)
thread.c.diff	552 Bytes	08/04/2011	jeremyevans0 (Jeremy Evans)
thread_2.diff	1014 Bytes	08/24/2011	normalperson (Eric Wong)