Ruby master - Bug #14841

Very rarely IO#readpartial does not raise EOFError

06/10/2018 02:09 PM - hirura (Hiroyuki URANISHI)

Status: Closed
Priority: Normal
Assignee:
Target version:
ruby -v: ruby 2.6.0dev (2018-06-15 trunk 63671) [x86_64-linux] 
Backport: 2.3: REQUIRED, 2.4: DONE, 2.5: DONE

Description

Hi,

I'm now writing a multi thread application, which uses IO.pipe, IO#close and IO#readpartial. During running a code many times, I found that, even though calling IO#close on a write side object instantiated by IO.pipe, IO#readpartial of the read side object very rarely does not raise EOFError and keeps blocking.

The below code is working fine in most cases.

1. Thread t2 generates '0' and '1' values and writes them on w object.
2. Thread t1 read from r object with IO#readpartial
3. After thread t2 exited, close w object to finish thread t1

```
require 'logger'

logger = Logger.new 'logger.log', 1, 10000000
logger.level = Logger::DEBUG

r, w = IO.pipe

r0 = Thread.new{
  # This is a dummy thread to avoid being killed by Ruby interpreter automatically due to deadlock
  loop do
    sleep 1
    Thread.pass
  end
}

r1 = Thread.new{
  loop do
    begin
      logger.debug { "r.readpartial(10240)" }
      data_read = r.readpartial(10240)
      logger.debug { "data_read: #\{data_read\}" }
      rescue EOFError
        logger.info { "EOError" }
        break
      rescue => e
        logger.error { e.message }
      end
      logger.info { "exiting t1" }
    end
  end
}

r2 = Thread.new{
  2.times { |i|
    sleep 0.1
    w.write i.to_s
  }
  logger.info { "exiting t2" }
}
```

03/15/2020
begin
  t2.join
  logger.info { "t2 exited" }
  logger.info { "close w" }
  w.close
  logger.info { "w closed" }
rescue IOError
  logger.info { "IOError when closing w" }
end

begin
  r.close
rescue IOError
  logger.info { "IOError when closing r" }
end

And I run the above code many times like:

$ i=0; while :; do echo $i; ruby readpartial_and_eof_test.rb; i=$(expr $i + 1); done
0
1
(Snip)
3892
3893
3894
(Stopped here)

As seen above, during 3894th iteration, the code is not finished.

The output of logger is the below. Closing w object is done, but EOFError looks not being raised.

$ tail -n 32 logger.log
I, [2018-06-10T22:09:58.738087 #19893] INFO -- : t1 exited
D, [2018-06-10T22:09:58.972198 #19916] DEBUG -- : data_read: 0
D, [2018-06-10T22:09:59.072441 #19916] DEBUG -- : data_read: 1
D, [2018-06-10T22:09:59.072758 #19916] DEBUG -- : r.readpartial(10240)
I, [2018-06-10T22:09:59.073166 #19916] INFO -- : exiting t1
I, [2018-06-10T22:09:59.073268 #19916] INFO -- : t1 exited
I, [2018-06-10T22:09:59.365959 #19944] INFO -- : exiting t1
I, [2018-06-10T22:09:59.366078 #19944] INFO -- : t1 exited
D, [2018-06-10T22:09:59.572743 #19967] DEBUG -- : data_read: 0

03/15/2020
This issue occurred in CentOS 7.2.1511 and Ubuntu 16.04.4 LTS.

- **CentOS**

```bash
$ cat /etc/redhat-release
CentOS Linux release 7.2.1511 (Core)
$ uname -a
Linux mfz-remote-gw-01 3.10.0-327.18.2.el7.x86_64 #1 SMP Thu May 12 11:03:55 UTC 2016 x86_64 x86_64 GNU/Linux
$ ruby -v
ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]
$ rbenv -v
rbenv 1.1.1-30-gc8ba27f
$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
Target: x86_64-redhat-linux
```  

- **Ubuntu**

```bash
$ cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.4 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.4 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
$ uname -a
Linux epsilon 4.4.0-122-generic #146-Ubuntu SMP Mon Apr 23 15:34:04 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
$ ruby -v
ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]
$ rbenv -v
rbenv 1.1.1-37-g1c772d5
$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-linux-gnu/5/lto-wrapper
Target: x86_64-linux-gnu
```
a-1.5.0-gcj-5-amd64 --with-arch-directory=amd64 --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1.6.04.9)

Associated revisions
Revision 501069b8 - 06/22/2018 02:32 AM - normal
thread_sync.c (rb_mutex_lock): fix deadlock
  
  thread_sync.c (rb_mutex_lock): fix deadlock [ruby-core:87467] [Bug #14841]

  git-svn-id: svn+ssh://ci.ruby-lang.org/ruby@trunk@63711 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63711 - 06/22/2018 02:32 AM - normal
person (Eric Wong)
thread_sync.c (rb_mutex_lock): fix deadlock

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

Revision 63711 - 06/22/2018 02:32 AM - normal
thread_sync.c (rb_mutex_lock): fix deadlock

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

Revision 2cf3bd5b - 08/16/2018 07:59 PM - normal
person (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 501069b8a4013f2e3fdde35c50e9527e0061963 (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 - normal
person (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 501069b8a4013f2e3fdde35c50e9527e0061963 (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 c561f0db - 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 15b0b984 - 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 501069b8a4013f2e3fddde35c50e9527ef0061963 (r63711)
("thread_sync.c (rb_mutex_lock): fix deadlock") for [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 501069b8a4013f2e3fddde35c50e9527ef0061963 (r63711)
("thread_sync.c (rb_mutex_lock): fix deadlock") for [Bug #14841]

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

History

#1 - 06/19/2018 09:19 AM - hirura (Hiroyuki URANISHI)
- ruby -v changed from ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux] to ruby 2.6.0dev (2018-06-15 trunk 63671) [x86_64-linux]
- File gdb_trace_reproduced.txt added
- File gdb_trace_normal.txt added

Hi,

I tried getting gdb backtrace of a reproduced, means stopped in this case, process. Attached files are the outputs of reproduced case and normal case.

Could anyone help on this?

hirura (Hiroyuki URANISHI) wrote:

Hi,

I'm now writing a multi thread application, which uses IO.pipe, IO#close and IO#readpartial. During running a code many times, I found that, even though calling IO#close on a write side object instantiated by IO.pipe, IO#readpartial of the read side object very rarely does not raise EOFError and keeps blocking.

The below code is working fine in most cases.

1. Thread t2 generates '0' and '1' values and writes them on w object.
2. Thread t1 read from r object with IO#readpartial
3. After thread t2 exited, close w object to finish thread t1

require 'logger'

logger = Logger.new 'logger.log', 1, 10000000
logger.level = Logger::DEBUG

r, w = IO.pipe

t0 = Thread.new{
  # This is a dummy thread to avoid being killed by Ruby interpreter automatically due to deadlock
  loop do
    sleep 1
    Thread.pass
  end
}

t1 = Thread.new{
  loop do
    begin
      logger.debug { "r.readpartial(10240)" }
      data_read = r.readpartial(10240)
      logger.debug { "data_read: #{data_read}" }
      rescue EOFError
        logger.info { "EOFError" }
        break
      rescue => e
        logger.error { e.message }
    end
    end
    logger.info { "exiting t1" }
  end
}

t2 = Thread.new{
  2.times do
    sleep 0.1
    w.write i.to_s
  end
  logger.info { "exiting t2" }
end

begin
  t2.join
  logger.info { "t2 exited" }
  logger.info { "close w" }
  w.close
  logger.info { "w closed" }
  rescue IOError
    logger.info { "IOError when closing w" }
end

  t1.join
  logger.info { "t1 exited" }

begin
  r.close
  rescue IOError
    logger.info { "IOError when closing r" }
end

And I run the above code many times like:

$ i=0; while ; do echo $i; ruby readpartial_and_eof_test.rb; i=$(expr $i + 1); done
0
1
(Snip)
3892
3893
3894
(Snipped here)

As seen above, during 3894th iteration, the code is not finished.

The output of logger is the below. Closing w object is done, but EOFError looks not being raised.

$ tail -n 32 logger.log
I, [2018-06-10T22:09:58.738087 #19893] INFO -- : t1 exited
D, [2018-06-10T22:09:58.972198 #19916] DEBUG -- : data_read: 0
This issue occurred in CentOS 7.2.1511 and Ubuntu 16.04.4 LTS.

- **CentOS**

  $ cat /etc/redhat-release
  CentOS Linux release 7.2.1511 (Core)
  $ uname -a
  Linux mzf-remote-gw-01 3.10.0-327.18.2.el7.x86_64 #1 SMP Thu May 12 11:03:55 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
  $ ruby -v
  ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]
  $ rbenv -v
  rbenv 1.1.1-30-gc8ba27f
  $ gcc -v
  Using built-in specs.
  COLLECT_GCC=gcc
  COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
  Target: x86_64-redhat-linux
  Thread model: posix
  gcc version 4.8.5 20150623 (Red Hat 4.8.5-16) (GCC)

- **Ubuntu**

  $ cat /etc/os-release
  NAME="Ubuntu"
  VERSION="16.04.4 LTS (Xenial Xerus)"
  ID=ubuntu
  ID_LIKE=debian
  PRETTY_NAME="Ubuntu 16.04.4 LTS"
  VERSION_ID="16.04"
  HOME_URL="http://www.ubuntu.com/"
  SUPPORT_URL="http://help.ubuntu.com/"
  BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
  VERSION_CODENAME=xenial
 UBUNTU_CODENAME=xenial
  $ uname -a
  Linux epsilon 4.4.0-122-generic #146-Ubuntu SMP Mon Apr 23 15:34:04 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
linux
$ ruby -v
ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]
$ rbenv -v
rbenv 1.1.1-37-glc772d5
$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure --prefix=/usr --enable-shared --enable-lto --libexecdir=/usr/lib --without-headers --enable-threads=posix --enable-objc-gc --enable-multiarch --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 5.4.0 20160609

#2 - 06/19/2018 10:52 AM - normalperson (Eric Wong)
hirura@gmail.com wrote:
File gdb_trace_reproduced.txt added
ruby -v changed from ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux] to ruby 2.6.0dev (2018-06-15 trunk 63671) [x86_64-linux]
Odd, it is stuck in rb_mutex_sleep based on your backtrace (from Logger, it looks like); and you can reproduce it on recent trunk.
I'm trying to reproduce the bug, but I cannot. I will let it run while I do other things...
I suspect the problem is a bug in our deadlock detection
(vm->sleeper (1))
In your code, you have a 0 dummy thread. Is it really needed?
Curious, how many CPU cores and what model do you have?
Can you reproduce it if you use taskset or schedtool to limit it to a single core?
Do you have a different hardware to try on?
(1) I also wonder why vm->sleeper is volatile, it should not need to be.

#3 - 06/19/2018 10:52 AM - normalperson (Eric Wong)
Sorry, I left this out: If you can reproduce it again, can you print vm->sleeper and vm->living_thread_num values from gdb?
Thanks.

#4 - 06/19/2018 11:52 AM - hirura (Hiroyuki URANISHI)
Hi Eric,
Thank you for replying.
I tried and reproduced this on
* Ubuntu Linux on Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz (4cores / 8threads)
* CentOS Linux on Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz (1cores / 2threads; a VM) Unfortunately I don't have a Linux machine with AMD or other vendors'.
As for additional gdb outputs, I have a dumped core file of the
stopped process, and I tried printing the instructed. But I couldn't get "vm" variable like

```plaintext
(gdb) print vm
No symbol "vm" in current context.
```

Could you give me a bit detailed way to get the output? I'm sorry I'm not familiar with gdb.

The t0 dummy thread is used so that the process is not killed by deadlock detection and I can get the core-dump. Without the thread, the process is killed as deadlock. The output of the case is the below. I suppose deadlock detection is working as expected. (Line 45 is t1.join in this case)

```
Traceback (most recent call last):
  1: from readpartial_and_eof_test.rb:45:in `main'
readpartial_and_eof_test.rb:45:in `join': No live threads left. Deadlock? (fatal)
  2 threads, 2 sleeps current:0x00007f8c498cfc20 main thread:0x00007f8c49566470
* #<Thread:0x00007f8c498cfc20 sleep_forever>
rб_thread_t:0x00007f8c49566470 native:0x00007f8c47674740 int:0
readpartial_and_eof_test.rb:45:in `join'
readpartial_and_eof_test.rb:45:in `main'
* #<Thread:0x00007f8c498cfc20 sleep_forever>
rб_thread_t:0x00007f8c498cfc20 native:0x00007f8c3f908700 int:0
depended by: tb_thread_id:0x00007f8c49566470
/home/hirura/.rbenv/versions/2.5.1/lib/ruby/2.5.0/monitor.rb:185:in `lock'
/home/hirura/.rbenv/versions/2.5.1/lib/ruby/2.5.0/monitor.rb:185:in `mon_enter'
/home/hirura/.rbenv/versions/2.5.1/lib/ruby/2.5.0/monitor.rb:224:in `mon_synchronize'
/home/hirura/.rbenv/versions/2.5.1/lib/ruby/2.5.0/logger.rb:688:in `write'
/home/hirura/.rbenv/versions/2.5.1/lib/ruby/2.5.0/logger.rb:471:in `add'
readpartial_and_eof_test.rb:16:in `block (2 levels) in <main>'
readpartial_and_eof_test.rb:12:in `loop'
readpartial_and_eof_test.rb:12:in `block in <main>'
```

Thanks,

#5 - 06/19/2018 12:42 PM - normalperson (Eric Wong)

hirura hirura@gmail.com wrote:

I tried and reproduced this on

- Ubuntu Linux on Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz (4cores / 8threads)
- CentOS Linux on Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz (1cores / 2threads; a VM) Unfortunately I don't have a Linux machine with AMD or other vendors'.

OK, I'm on AMD (Phenom II) with 4 cores and no HT.

As for additional gdb outputs, I have a dumped core file of the stopped process, and I tried printing the instructed. But I couldn't get "vm" variable like

```
(gdb) print vm
No symbol "vm" in current context.
```

Could you give me a bit detailed way to get the output? I'm sorry I'm not familiar with gdb.

You can use "up" and "down" to move up/down stack frames until you're in a function where "vm" is accessible. (I'm not a gdb expert, either)

The t0 dummy thread is used so that the process is not killed by deadlock detection and I can get the core-dump. Without the thread, the process is killed as deadlock.

Odd, I can't reproduce a deadlock with t0 commented out, either.
The output of the case is the below. I suppose deadlock detection is working as expected.
(Line 45 is t1.join in this case)

```
Traceback (most recent call last):
  1: from readpartial_and_eof_test.rb:45:in `<main>'
readpartial_and_eof_test.rb:45:in `join': No live threads left.
Deadlock? (fatal)
2 threads, 2 sleeps current:0x00007f8c498cfc20 main thread:0x00007f8c49566470
```

OK, so that's 2 for both vm->living_thread_num and vm->sleeper; in that case that's normal.

I'm wondering if a stuck process will show some strange values (e.g. negative due to underflow).

Perhaps `volatile' is unnecessary and the compiler does something strange with sleeper... Does removing it change things?

```
--- a/vm_core.h
+++ b/vm_core.h
@@ -565,7 +565,7 @@ typedef struct rb_vm_struct {
     unsigned int safe_level_: 1;
     int trace_running;
-    volatile int sleeper;
+    int sleeper;
  */ object management */
VALUE mark_object_ary;
```

#6 - 06/19/2018 01:33 PM - hirura (Hiroyuki URANISHI)

Thank you for letting me know how to touch gdb. The below is the output, it seems that sleeper and living_thread_num have reasonable values.

```
(gdb) t 1
(Switching to thread 1 (Thread 0x7f646c51c8700 (LWP 827)))
#0 pthread_cond_wait@GLIBC_2.3.2 () at ../sysdeps/unix/sysv/linux/x86_64/pthread_cond_wait.S:185
185 in ../sysdeps/unix/sysv/linux/x86_64/pthread_cond_wait.S
(gdb) up 3
#3 0x000055bec706df9a in thread_join_sleep (arg=140725175158608) at thread.c:932
932 native_sleep(th, 0);
(gdb) p th->vm
$1 = (rb_vm_t *) 0x55bec8085f20
(gdb) p th->vm->sleeper
$2 = 2
(gdb) p th->vm->living_thread_num
$3 = 3
(gdb)
```

```
(gdb) t 3
(Switching to thread 3 (Thread 0x7f646c37aa700 (LWP 841)))
#0 pthread_cond_timedwait@GLIBC_2.3.2 () at ../sysdeps/unix/sysv/linux/x86_64/pthread_cond_timedwait.S:225
225 in ../sysdeps/unix/sysv/linux/x86_64/pthread_cond_timedwait.S: No such file or directory.
(gdb) up 3
#2 0x000055bec7069920 in native_sleep (th=0x55bec8468fa0, timeout_rel=0x7f646c37aa7b50) at thread(pthread.c:1073
1073 native_cond_timedwait(cond, lock, &timeout);
(gdb) p th->vm
$1 = (rb_vm_t *) 0x55bec8085f20
(gdb) p th->vm->sleeper
$2 = 2
(gdb) p th->vm->living_thread_num
$3 = 3
(gdb)
```

```
(gdb) t 4
```

03/15/2020 11/14
Nextly, I will try the same code with limited CPU cores, and will try patching vm_core.h.

#7 - 06/20/2018 06:33 AM - normalperson (Eric Wong)

hirura hirura@gmail.com wrote:

Nextly, I will try the same code with limited CPU cores, and will try patching vm_core.h.

Thanks. I'm failing to reproduce the problem on my end.
I also got my netbook to overheat, but that's not Ruby's fault :x

#8 - 06/20/2018 10:04 AM - normalperson (Eric Wong)

mofezilla@gmail.com wrote:

https://bugs.ruby-lang.org/issues/14841

I think the problem is mutex->th may be altered by RUBY_VM_CHECK_INTS_BLOCKING(th->ec), and we miss a point where "mutex->th = 0;"

Can you test?

--- a/thread_sync.c
+++ b/thread_sync.c
@@ -272,6 +272,7 @@ rb_mutex_lock(VALUE self)
     list_add_tail(&mutex->waitq, &w.node);
     native_sleep(th, timeout); /* release GVL */
     list_del(&w.node);
+    RUBY_VM_CHECK_INTS_BLOCKING(th->ec);
   if (!mutex->th) {
     mutex->th = th;
   }
@@ -289,8 +290,6 @@ rb_mutex_lock(VALUE self)
     th->vm->sleeper--;
   }
   if (mutex->th == th) mutex_locked(th, self);
-    -RUBY_VM_CHECK_INTS_BLOCKING(th->ec);
   }
 return self;

I think the bug existed in <2.4, but 2.5 made it easier-to-hit.

#9 - 06/20/2018 10:42 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

mofezilla@gmail.com wrote:

https://bugs.ruby-lang.org/issues/14841

Can you test?
Sorry, right idea, bad patch (broke during "make exam")
This should work:

```c
--- a/thread_sync.c
+++ b/thread_sync.c
@@ -289,8 +289,11 @@ rb_mutex_lock(VALUE self)
 th->vm->sleeper--;
                 
 if (mutex->th == th) mutex_locked(th, self);
-        
-         - RUBY_VM_CHECK_INTS_BLOCKING(th->ec);
+        - RUBY_VM_CHECK_INTS_BLOCKING(th->ec); /* may release mutex */
+        +      if (!mutex->th) {
+        +         mutex->th = th;
+        +         mutex_locked(th, self);
+        +      }
+ }
        
 return self;
```

#10 - 06/22/2018 02:32 AM - normalperson (Eric Wong)
- Status changed from Open to Closed

Applied in changeset trunk|r63711.

thread_sync.c (rb_mutex_lock): fix deadlock

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

#11 - 06/22/2018 02:41 AM - normalperson (Eric Wong)
- Backport changed from 2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN to 2.3: REQUIRED, 2.4: REQUIRED, 2.5: REQUIRED

#12 - 06/22/2018 02:41 AM - normalperson (Eric Wong)
- Status changed from Closed to Open

#13 - 06/22/2018 02:42 AM - normalperson (Eric Wong)
This affects all versions, but is most easily triggered in 2.5+

#14 - 06/22/2018 02:43 AM - Anonymous
- Status changed from Open to Closed

Applied in changeset commit:ruby-git\[501069b8a4013f2e3fdde35c50e9527ef0061963.

thread_sync.c (rb_mutex_lock): fix deadlock

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

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63711 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

#15 - 06/24/2018 05:18 AM - hirura (Hiroyuki URANISHI)

Hi,

Sorry to reply late.
Thank you for your quick help and fix.

I have tried the same code more than 1,000,000 times on "ruby 2.6.0dev (2018-06-22 trunk 63723) [x86_64-linux]" version that includes the fix, and I confirmed the issue is not reproduced.
And just for information, I tried the older version, that does not include the fix, with "taskset" tool, and the issue was not reproduced as well.

Thanks,

#16 - 10/11/2018 02:40 PM - nagachika (Tomoyuki Chikanaga)
- Backport changed from 2.3: REQUIRED, 2.4: REQUIRED, 2.5: REQUIRED to 2.3: REQUIRED, 2.4: REQUIRED, 2.5: DONE
#17 - 10/17/2018 08:40 AM - usa (Usaku NAKAMURA)

- Backport changed from 2.3: REQUIRED, 2.4: REQUIRED, 2.5: DONE to 2.3: REQUIRED, 2.4: DONE, 2.5: DONE

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_trace_normal.txt</td>
<td>15.6 KB</td>
<td>06/19/2018</td>
<td>hirura (Hiroyuki URANISHI)</td>
</tr>
<tr>
<td>gdb_trace_reproduced.txt</td>
<td>14.2 KB</td>
<td>06/19/2018</td>
<td>hirura (Hiroyuki URANISHI)</td>
</tr>
</tbody>
</table>