Ruby master - Bug #13882

Exception in `ensure` stops threads from exiting

09/08/2017 08:17 PM - zanker (Zachary Anker)

Status:	Assigned		
Priority:	Normal		
Assignee:	ko1 (Koichi Sasada)		
Target version:			
ruby -v:	2.4.1	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

When the Ruby process is gracefully exiting, if a thread has an exception during an ensure block it appears the Ruby process forgets it's attempting to exit and will keep running forever. Since there's still an alive thread, rb_thread_terminate_all doesn't finish (since vm_living_thread_num(vm) > 1 is still true), and the Ruby process never exits until you kill -9 it.

I was able to cause this going back as far as MRI 2.0.0, but didn't have a 1.9.3 install to double check with. Repo case:

```
Thread.new do
  loop do
    puts "Loop start"

    begin
      begin
        sleep
      ensure
        raise
      end
    rescue => e
      p e
    end
  end
end

sleep 1
exit
```

Will result in a two Loop start messages.

When running GDB on the process, we see Ruby is waiting on sleep_forever which is expected, but you can see the main thread is stuck on rb_thread_terminate_all:

```
(gdb) t a a bt

Thread 3 (Thread 0x7f956f36e700 (LWP 401088)):
#0  0x00000033d90df113 in poll () from /lib64/libc.so.6
#1  0x00007f9575863775 in timer_thread_sleep (p=0x7f957524c008) at thread_pthread.c:1460
#2  thread_timer (p=0x7f957524c008) at thread_pthread.c:1568
#3  0x00000033d9407aa1 in start_thread () from /lib64/libpthread.so.0
#4  0x00000033d90e893d in clone () from /lib64/libc.so.6

Thread 2 (Thread 0x7f956f363700 (LWP 401117)):
#0  0x00000033d940b68c in pthread_cond_wait@@GLIBC_2.3.2 () from /lib64/libpthread.so.0
#1  0x00007f957585d959 in native_cond_wait (cond=<value optimized out>, mutex=<value optimized out
>) at thread_pthread.c:343
#2  0x00007f95758675b8 in native_sleep (th=0x7f956c491800, timeout_tv=0x0) at thread_pthread.c:114
7
#3  0x00007f95758682f2 in sleep_forever () at thread.c:1083
#4  rb_thread_sleep_forever () at thread.c:1157
#5  0x00007f95757e1ca5 in rb_f_sleep (argc=0, argv=0x7f956bce6a78) at process.c:4393
#6  0x00007f9575890bda in vm_call_cfunc_with_frame (th=0x7f956c491800, reg_cfp=0x7f956bde6940, cal
ling=<value optimized out>, ci=0x7f956e98c910, cc=<value optimized out>)
    at vm_insnhelper.c:1752
```

```
#7  vm_call_cfunc (th=0x7f956c491800, reg_cfp=0x7f956bde6940, calling=<value optimized out>, ci=0x
7f956e98c910, cc=<value optimized out>) at vm_insnhelper.c:1847
#8  0x00007f957589f12b in vm_exec_core (th=<value optimized out>, initial=<value optimized out>) a
t insns.def:1066
#9  0x00007f95758a470b in vm_exec (th=0x7f956c491800) at vm.c:1727
#10 0x00007f95758abc04 in invoke_block () at vm.c:969
#11 invoke_iseq_block_from_c () at vm.c:1014
#12 invoke_block_from_c_splattable () at vm.c:1032
#13 vm_yield () at vm.c:1074
#14 rb_yield_0 () at vm_eval.c:1010
#15 loop_i () at vm_eval.c:1088
#16 0x00007f9575755954 in rb_rescue2 (b_proc=0x7f95758ab8b0 <loop_i>, data1=0, r_proc=0x7f957588b2
a0 <loop_stop>, data2=0) at eval.c:838
#17 0x00007f9575890bda in vm_call_cfunc_with_frame (th=0x7f956c491800, reg_cfp=0x7f956bde69a0, cal
ling=<value optimized out>, ci=0x7f956c5090f0, cc=<value optimized out>)
    at vm_insnhelper.c:1752
#18 vm_call_cfunc (th=0x7f956c491800, reg_cfp=0x7f956bde69a0, calling=<value optimized out>, ci=0x
7f956c5090f0, cc=<value optimized out>) at vm_insnhelper.c:1847
#19 0x00007f95758a726b in vm_call_method (th=0x7f956c491800, cfp=0x7f956bde69a0, calling=<value op
timized out>, ci=<value optimized out>, cc=<value optimized out>)
    at vm_insnhelper.c:2295
#20 0x00007f957589f97c in vm_exec_core (th=<value optimized out>, initial=<value optimized out>) a
t insns.def:967
#21 0x00007f95758a470b in vm_exec (th=0x7f956c491800) at vm.c:1727
#22 0x00007f95758a5771 in invoke_block (th=0x7f956c491800, captured=<value optimized out>, self=14
0279789266160, argc=<value optimized out>, argv=<value optimized out>,
    passed_block_handler=<value optimized out>, cref=0x0, splattable=0, is_lambda=0) at vm.c:969
#23 invoke_iseq_block_from_c (th=0x7f956c491800, captured=<value optimized out>, self=140279789266
160, argc=<value optimized out>, argv=<value optimized out>,
    passed_block_handler=<value optimized out>, cref=0x0, splattable=0, is_lambda=0) at vm.c:1014
#24 0x00007f95758a581f in invoke_block_from_c_unsplattable (th=<value optimized out>, block=<value
 optimized out>, self=<value optimized out>, argc=<value optimized out>,
    argv=<value optimized out>, passed_block_handler=<value optimized out>, is_lambda=<value optim
ized out>) at vm.c:1101
#25 0x00007f95758a595a in vm_invoke_proc (th=0x7f956c491800, proc=0x7f956e04da50, self=14027978926
6160, argc=0, argv=0x7f956c547c18, passed_block_handler=0) at vm.c:1126
#26 0x00007f9575864589 in thread_do_start (th=0x7f956c491800, stack_start=0x7f956f364000) at threa
d.c:577
#27 thread_start_func_2 (th=0x7f956c491800, stack_start=0x7f956f364000) at thread.c:619
#28 0x00007f95758649b6 in thread_start_func_1 (th_ptr=0x7f956c491800) at thread_pthread.c:887
#29 0x00000033d9407aa1 in start_thread () from /lib64/libpthread.so.0
#30 0x00000033d90e893d in clone () from /lib64/libc.so.6

Thread 1 (Thread 0x7f95756ab760 (LWP 401080)):
#0  0x00000033d940b68c in pthread_cond_wait@@GLIBC_2.3.2 () from /lib64/libpthread.so.0
#1  0x00007f957585d959 in native_cond_wait (cond=<value optimized out>, mutex=<value optimized out
>) at thread_pthread.c:343
#2  0x00007f95758675b8 in native_sleep (th=0x7f9575229400, timeout_tv=0x0) at thread_pthread.c:114
7
#3  0x00007f9575867e34 in rb_thread_terminate_all () at thread.c:494
#4  0x00007f9575758466 in ruby_cleanup (ex=6) at eval.c:186
#5  0x00007f9575758725 in ruby_run_node (n=0x7f956c54c6b8) at eval.c:300
#6  0x00000000004008eb in main (argc=2, argv=0x7fffccd8e7c8) at main.c:36
(gdb)
```

Looking at the thread state shows that the rb_threadptr_to_kill executed properly, because to_kill is properly set, but the errinfo was nilled out:

```
(gdb) p ruby_current_thread
$1 = (rb_thread_t *) 0x7f8cd06eb800
(gdb) p ruby_current_thread->to_kill
$2 = 1
(gdb) p ruby_current_thread->status
$3 = THREAD_STOPPED
(gdb) p ruby_current_thread->errinfo
$4 = 8
```

I'm happy to contribute a patch, but not quite sure what the appropriate fix for this would be. It looks like the issue is an exception in ensure is causing the TAG_FATAL on errinfo to be overwritten so the thread doesn't think it should exit, and the right fix is to also check if to_kill is set.

Related issues:

Has duplicate Ruby master - Bug #17164: Threads can ignore kill/interrupt/abort	**Open**

History

#1 - 09/15/2017 02:45 AM - ko1 (Koichi Sasada)

- Assignee set to ko1 (Koichi Sasada)

- Status changed from Open to Assigned

#2 - 02/21/2018 06:35 AM - ko1 (Koichi Sasada)

Sorry for long absent.

The point is "when should we check to_kill flag"?
It is not clear.

This is a issue that Ctrl-C doesn't work on it (and we need to use kill -9).
I think modify Ctrl-C issue is enough on this ticket.

Thoughts?

#3 - 02/26/2018 06:26 PM - zanker (Zachary Anker)

ko1 (Koichi Sasada) wrote:

> Sorry for long absent.
>
> The point is "when should we check to_kill flag"?
> It is not clear.
>
> This is a issue that Ctrl-C doesn't work on it (and we need to use kill -9).
> I think modify Ctrl-C issue is enough on this ticket.
>
> Thoughts?

No problem! The problem with Ctrl-C is that will fix it for user facing scripts, but not background processes. The example we ran into with this, was using Puma with forked child processes. The Puma master process was sent a kill -int, which then sent it to the child processes, but not all of them exited properly due to this bug.

To clarify the to_kill part, what is happening internally is:

1. Ruby VM receives a signal telling it to exit
2. terminate_all is called which enqueues the eTerminateSignal signal
3. Eventually hits rb_threadptr_execute_interrupts which calls rb_threadptr_to_kill
4. rb_threadptr_to_kill sets errinfo = INT2FIX(TAG_FATAL) and to_kill = 1 on the thread struct
5. Exception is caught (see example in the initial report), which causes errinfo to be nilled out
6. Ruby VM doesn't try and finish exiting the thread since the errinfo flag is now nilled out

It seems like the Ruby VM shouldn't allow you to rescue exceptions in such a way that it can block a thread from exiting.

#4 - 05/29/2020 11:42 PM - wjordan (Will Jordan)

- File http_gzip_hang.rb added

This bug still seems to be affecting all more recent versions of Ruby up to and including 2.7.1.

I encountered this in the newrelic_rpm gem which was intermittently hanging on shutdown in a short-lived script. It turns out that if Ruby exits while a thread is in the middle of receiving a gzip-encoded request via Net::HTTP, this call to #finish in an ensure block raises a Zlib::BufError exception, triggering this bug.

I've attached a minimal repro of this issue being triggered by gzip decoding in Net::HTTP, hope this is helpful.

#5 - 05/30/2020 07:10 AM - Eregon (Benoit Daloze)

In the original example, it gets stuck on the second call to sleep.
Basically the "kill Thread" exception (which cannot be rescue-d directly) was thrown after Ctrl+C to the Thread.
However, that ensure raises another (RuntimeError) exception, and that replaces the "kill Thread" exception.
I'm not sure what we could do in the VM to fix that.

Maybe raise during shutdown should always raise a "kill Thread" exception and ignore the actual exception passed?

Or the "kill Thread" exception could be set as an internal cause, and when rescue-ing the RuntimeError, that "kill Thread" exception would be re-raised at the end of the rescue block? (sounds fragile)

Retrying from the main thread to kill other threads is not great, because how to know how long we should wait until the retry?

I would argue it's at least partly the program's bug, because it swallows an exception and keeps going ignoring it.
For instance, if some other error (e.g. a SyntaxError, an OutOfMemoryError, a NoMethodError if sleep was misspelled, etc) happened in such a program, it would not terminate the program and probably go unnoticed.

#6 - 05/30/2020 07:36 AM - Eregon (Benoit Daloze)

Regarding the http/gzip example, it seems unfortunate but it's probably the same for any exception raised in ensure: that exception might replace a more "critical" exception such as "kill Thread" exception/NoMemoryError/SignalException/etc.
And that in itself can be useful, i.e., it allows to find what failed in that ensure.

In the example, the thread goes to infinite sleep after rescue, so it seems somewhat expected that program can hang though.

Related commits:
https://github.com/ruby/ruby/commit/6811973d1355ba40ee4b3fc5a43ed65b67aac9b7 by naruse (Yui NARUSE), which attemps to re-raise the original exception, but that doesn't work for "kill Thread" exceptions.
https://github.com/ruby/ruby/commit/d7bb66df2667040518186eb72928dedb4b7de6b9 which has to workaround because "kill Thread" exceptions set $! to nil

I think the only good fix for this case is to make inflate_body_io.finish not raise or have an alternative method doing similar cleanup but not raise.
Raising in ensure is problematic as seen in this issue.

One thought is we could always re-raised the exception entering the ensure block when leaving it.
But that would then completely ignore exceptions raised inside ensure, which might or not be wanted.

For this specific case, I think this is a simple fix, assuming we don't care (i.e., won't be shown anywhere, needs --debug) about the exception of finish if there is another exception:

```
      begin
        yield inflate_body_io
        success = true
      ensure
        begin
          inflate_body_io.finish
        rescue => err
          # Ignore #finish's error if there is an exception from yield
          raise err if success
        end
      end
```

Then the script terminates as expected.

#7 - 05/30/2020 08:35 AM - Eregon (Benoit Daloze)

PR to fix the Net::HTTP/gzip case with specs to verify the fix: https://github.com/ruby/ruby/pull/3164

#8 - 05/31/2020 10:52 AM - Eregon (Benoit Daloze)

I merged that PR and requested a backport in #16925.

#9 - 09/10/2020 10:43 PM - jeremyevans0 (Jeremy Evans)

- Has duplicate Bug #17164: Threads can ignore kill/interrupt/abort added

Files

http_gzip_hang.rb	832 Bytes	05/29/2020	wjordan (Will Jordan)