[PATCH] implement Timeout in VM

06/21/2018 12:21 AM - normalperson (Eric Wong)

Status: Open
Priority: Normal
Assignee: ko1 (Koichi Sasada)
Target version:

Description

implement Timeout in VM

Based on the ugliness of handling partial writes with
IO#write_nonblock and inability to use writev(2) effectively
with write timeouts in Net::HTTP in r63587-r63589, I've
decided Timeout to be the more programmer-friendly option
to use and to improve it.

Timeout is significantly faster with this patch, and stopping
the timeout before expiration (common case) is optimized to be
as fast as possible. This version relies on timer-thread to
provide wakeup interrupts.

This is a minimally intrusive patch. I also started working on
a more intrusive patch to touch all sleep/waiting function
calls, but this is easier-to-review for now. In the future,
I will try per-thread timeouts and eliminate timer-thread
for platforms with POSIX timers (timer_create/timer_settime)

Speedup ratio: compare with the result of ‘trunk’ (greater is better)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout_mt_nested</td>
<td>3.887</td>
</tr>
<tr>
<td>timeout_mt_same</td>
<td>3.843</td>
</tr>
<tr>
<td>timeout_mt_ugly</td>
<td>1.335</td>
</tr>
<tr>
<td>timeout_nested</td>
<td>7.059</td>
</tr>
<tr>
<td>timeout_same</td>
<td>5.173</td>
</tr>
<tr>
<td>timeout_zero</td>
<td>2.587</td>
</tr>
</tbody>
</table>

History

#1 - 06/21/2018 01:42 AM - normalperson (Eric Wong)

https://bugs.ruby-lang.org/issues/14859

I sometimes get failures in test_condvar_wait_deadlock_2 of
test/thread/test_cv.rb with this, but I think that test is too
dependent on thread scheduling timing...

In other words, I'm not sure it's a good test...

#2 - 06/21/2018 02:04 AM - normalperson (Eric Wong)

https://bugs.ruby-lang.org/issues/14859

I sometimes get failures in test_condvar_wait_deadlock_2 of
test/thread/test_cv.rb with this, but I think that test is too
dependent on thread scheduling timing...

Disregard, will fix...
Should `lib/timeout.rb` be removed then?
Why is it moved to core, could it stay an extension?

Note that there are pure-Ruby implementations of Timeout using a single Ruby Thread, like
https://github.com/oracle/truffleruby/blob/71df1ec4f0d9a318b5bd3998cfaeb85a96de7a8b/lib/truffle/timeout.rb (originally from Rubinius)

and that WEBrick has basically its own version of Timeout, using a single Thread:
https://github.com/ruby/ruby/blob/48efa44719d03eb067d27b32c68e821074aedce/lib/webrick/utils.rb

Something else, I would consider Timeout to be fundamentally flawed as long as it relies on Thread#raise, because it can fire in the middle of an ensure block:
http://headius.blogspot.com/2008/02/rubys-threadraise-threadkill-timeoutrb.html

Maybe IO#write and such should directly accept a timeout argument and other changes to make the API easier to use?

Right now it needs to hook into the core timer thread without needing additional threads. I already run into resource exhaustion problems with timer-thread when testing.

I'm working on making all wait functions aware of it:
`rb_wait_for_single_fd, rb_thread_fd_select, rb_thread_sleep*`, etc.

Eventually, I also want to get rid of timer thread (for POSIX) but it might not be easy

Using one extra Thread is already too much for me.

Yes, I want to get rid of that by making Timeout better.

Something else, I would consider Timeout to be fundamentally flawed as long as it relies on Thread#raise, because it can fire in the middle of an ensure block:
http://headius.blogspot.com/2008/02/rubys-threadraise-threadkill-timeoutrb.html

We have Thread.handle_interrupt, nowadays, to control when interrupts fire.

Maybe IO#write_nonblock and such should directly accept a timeout argument and other changes to make the API easier to use?

I considered that, too, but we'd need to add timeouts to every single method which can block. There are many: File.open, Queue#pop, SizedQueue#push, Mutex#lock/synchronize, Process.wait*.

IO#gets, IO#write, IO#read, IO#getc, IO.copy_stream, ...
Pretty much every IO method needs to be changed and callers need to be rewritten.

Things like File.open and Process.wait* don't have timeouts in the underlying syscall, so we'd still have to use a timer thread or POSIX timers to interrupt them on timeout.

The goal is to make all those methods aware of Timeout without changing existing user code at all.

#7 - 06/22/2018 12:42 AM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

Should lib/timeout.rb be removed then?

Also, I might keep compatibility code in timeout.rb and stop providing it. This avoids bloating the VM with deprecated stuff without breaking backwards compatibility.

#8 - 06/22/2018 01:04 AM - normalperson (Eric Wong)

stop providing it.

I meant: stop using rb_provide("timeout.rb")

#9 - 06/22/2018 10:20 AM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

eregontp@gmail.com wrote:

Something else, I would consider Timeout to be fundamentally flawed as long as it relies on Thread.raise, because it can fire in the middle of an ensure block:

http://headius.blogspot.com/2008/02/rubys-threadraise-threadkill-timeoutrb.html

We have Thread.handle_interrupt, nowadays, to control when interrupts fire.

Right, although it's very difficult to use correctly (for instance, it's incorrect to use Thread.handle_interrupt inside the ensure block) and can easily cause hangs or deadlocks.

BTW, it looks like MonitorMixin::ConditionVariable doesn't use Thread.handle_interrupt and could continue out of #wait (with an exception thrown by Thread.raise) without reacquiring the lock.

It might be nice to have a Timeout variant that only interrupts blocking IO, without relying on Thread.raise (but just SIGVTALRM).

I think that would be easier/safer to use than the current Timeout.timeout().

Not sure how to deal if there are multiple IO calls inside that Timeout block though.

And there could still be blocking IO in an ensure block, which would not work as intended.

I considered that, too, but we'd need to add timeouts to every single method which can block. There are many: File.open, Queue#pop, SizedQueue#push, Mutex#lock/synchronize, Process.wait*, IO#gets, IO#write, IO#read, IO#getc, IO.copy_stream, ...

It seems fine to me. Other implementations already have a timeout on Queue#pop IIRC.

I'm not sure we need all of them right now (what use case for Mutex#lock/synchronize ?).

I think the main need would be for standard IO like IO#read/write, especially on sockets and pipes.

#10 - 06/22/2018 10:12 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

normalperson (Eric Wong) wrote:

eregontp@gmail.com wrote:
Something else, I would consider Timeout to be fundamentally flawed as long as it relies on Thread.raise, because it can fire in the middle of an ensure block:

http://headius.blogspot.com/2008/02/rubys-threadraise-threadkill-timeoutrb.html

We have Thread.handle_interrupt, nowadays, to control when interrupts fire.

Right, although it's very difficult to use correctly (for instance, it's incorrect to use Thread.handle_interrupt inside the ensure block) and can easily cause hangs or deadlocks.

Agreed, it should be easier-to-use; but that's a separate issue and I'm trying to avoid dealing with public API design as much as possible for this.

BTW, it looks like MonitorMixin::ConditionVariable doesn't use Thread.handle_interrupt and could continue out of #wait (with an exception thrown by Thread.raise) without reacquiring the lock.

Can you report separately to shugo to be sure he sees it? I've never really understood the point of that module :x

It might be nice to have a Timeout variant that only interrupts blocking IO, without relying on Thread.raise (but just SIGVTALRM). I think that would be easier/safer to use than the current Timeout.timeout(). Not sure how to deal if there are multiple IO calls inside that Timeout block though. And there could still be blocking IO in an ensure block, which would not work as intended.

Agreed, I would welcome a "soft" timeout, without using signals/raise at all. I think my work-in-progress "intrusive" [PATCH 2/1] for this will make such a thing easier-to-implement:

https://80x24.org/spew/20180622215745.20698-1-e@80x24.org/raw

I considered that, too, but we'd need to add timeouts to every single method which can block. There are many: File.open, Queue#pop, SizedQueue#push, Mutex#lock/synchronize, Process.wait*, IO#gets, IO#write, IO#read, IO#getc, IO.copy_stream, ...

It seems fine to me. Other implementations already have a timeout on Queue#pop IIRC. I'm not sure we need all of them right now (what use case for Mutex#lock/synchronize ?). I think the main need would be for standard IO like IO#read/write, especially on sockets and pipes.

The maintenance overhead for adding timeouts to every call would be overwhelming on a human level, especially when 3rd-party libraries need to be considered.

I would much rather do the following:

```ruby
Timeout.timeout(30) do
  foo.read(...)
  foo.write(...)
  IO.copy_stream(...)
  foo.write(...)
  szqueue.push(...)
  resultq.pop
end
```

Than this:

03/14/2022
```ruby
def now
  Process.clock_gettime(Process::CLOCK_MONOTONIC)
end
```

```ruby
begin
  @stop = now + 30
  ...
  tout = @stop - now
  raise Timeout::Error if tout <= 0
  foo.read(..., tout)
  tout = @stop - now
  raise Timeout::Error if tout <= 0
  foo.write(..., tout)
  tout = @stop - now
  raise Timeout::Error if tout <= 0
  IO.copy_stream(..., tout)
  tout = @stop - now
  raise Timeout::Error if tout <= 0
  foo.write(..., tout)
  tout = @stop - now
  raise Timeout::Error if tout <= 0
  szqueue.push(..., tout)
  tout = @stop - now
  raise Timeout::Error if tout <= 0
  resultq.pop(tout)
end
```

#11 - 07/17/2018 05:42 AM - normalperson (Eric Wong)

```
Feature #14859: [PATCH] implement Timeout in VM
https://bugs.ruby-lang.org/issues/14859

Note: I'm still working on this. Feature #13618 basically has an implementation of timeouts in the VM, too, because of timeout args in rb_io_wait_able.

I would rather implement this feature first without making visible API changes for #13618.
```

#12 - 07/19/2018 01:53 AM - ko1 (Koichi Sasada)

```
Hi,

Could you explain your algorithm in pseudo code (or English)?
Current timeout method call makes a thread and use Thread#raise.

I assume that your idea is creating "timeout scheduler" in VM and it manages timeout calls and invoke Thread#raise for timeout blocks if necessary.

BTW:

I meant: stop using rb_provide("timeout.rb")

Why? Some existing codes require('timeout').
```

#13 - 07/19/2018 04:04 AM - normalperson (Eric Wong)

```
ko1@atdot.net wrote:
  > Hi,
  >
  > Could you explain your algorithm in pseudo code (or English)?
  > Current `timeout` method call makes a thread and use `Thread#raise`.
  >
  > I assume that your idea is creating "timeout scheduler" in VM and it manages `timeout` calls and invoke `Thread#raise` for timeout blocks if necessary.

Yes. The "timeout scheduler" is the same idea I used for auto-fiber.
```
It uses ccan/list to manage a sorted list of timeouts.

In my early version of the patch, I think the list_head struct is per-VM. I may make this per-thread; not sure, yet.

Either way, the idea is the same based on ccan/list and sort order.

list_del() is fast, so timer expiration (common case) is cheap.

Slowest part is insertion sort to maintain order O(n); but we can optimize for expected usage and limit traversal.

If the list_head is VM-wide; it insertion sort should walk backwards since we can assume many Threads will use the same timeout. If list_head is per-Thread, it should walk forwards; because nested Timeout only makes sense if inner timeout is smaller than outer one.

In other words, this is wrong regardless of implementation, so I won't optimize for it:

```ruby
Timeout.timeout(t+=1) do
Timeout.timeout(t+=1) do
Timeout.timeout(t+=1) do
Timeout.timeout(t+=1) do
Timeout.timeout(t+=1) do
```

This is correct, but overkill:

```ruby
Timeout.timeout(t-=1) do
Timeout.timeout(t-=1) do
Timeout.timeout(t-=1) do
Timeout.timeout(t-=1) do
Timeout.timeout(t-=1) do
```

Best is just a single-timeout per-EC:

```ruby
Timeout.timeout(t) do
...```

Worst-case insertion sort should still be faster than Thread.new :)

The list_top() check covers good blocking functions which take timeout arguments. However, we still need to rely on timer interrupt flag for functions which do not take timeout (and pure-Ruby code). So we need to set timer thread or POSIX timer to set interrupt flag (same way we do normal timeslice).

(*) Btw, I should have timer-thread removal w/ POSIX timers ready-to-publish soon.

> BTW:
> > I meant: stop using rb_provide("timeout.rb")
> > Why? Some existing codes `require('timeout')`.
>
I mean, we keep lib/timeout.rb as an empty file so `require` still works; but is a no-op. I don't feel strongly about it, though.

#14 - 07/21/2018 08:09 AM - funny_falcon (Yura Sokolov)

normalperson (Eric Wong) wrote:

Yes. The "timeout scheduler" is the same idea I used for auto-fiber.

It uses ccan/list to manage a sorted list of timeouts.

Still wonder, why you don't use binary min-heap for timers - most commonly used datastructure for this task. It has guaranteed O(log n) performance for insertion/deletion, and O(1) for check for min, and has very simple implementation. Note, that for insertion of new maximum value, binary min-heap also gives O(1) performance because item will not sift up.
normalperson (Eric Wong) wrote:

Yes. The "timeout scheduler" is the same idea I used for auto-fiber. It uses ccan/list to manage a sorted list of timeouts.

Still wonder, why you don't use binary min-heap for timers - most commonly used datastructure for this task.

It has guaranteed O(log n) performance for insertion/deletion, and O(1) for check for min, and has very simple implementation.

Note, that for insertion of new maximum value, binary min-heap also gives O(1) performance because item will not sift up.

I'll keep that in mind; I haven't looked at heap data structures in years. :-< O(log n) for delete doesn't sound appealing, though.

Most timeouts do not fire, they are deleted before the timeout is up because the task finishes on time. So I gravitate towards ccan/timer which has O(1) branchless delete (because it is just list_del from ccan/list).

However I tried ccan/timer from git://git.ozlabs.org/~ccan/ccan and making it available via "timeout_lgpl" bundled gem (LGPL); but I kept on hitting the brute_force_first() function which ended up being slow to find the first timer. So I will avoid it until brute_force_first is replaced/optimized in ccan/timer.

[ Old abandoned patch + timeout_lgpl gem using ccan/timer: https://80x24.org/spew/20180616120544.28171-1-e@80x24.org/raw git clone https://80x24.org/timeout_ext.git ]

For initial implementation, I chose naive insertion sort as it still beats Thread.new in current timeout.rb. But maybe we can try other data structures in the future (maybe build a skip list using ccan/list).