Ruby master - Feature #18630

Introduce general `IO#timeout` and `IO#timeout=` for all (non-)blocking operations.

03/14/2022 02:43 AM - ioquatix (Samuel Williams)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

I would like us to consider introducing a general timeout for all blocking operations. This timeout can be specified per IO instance. It's useful for ensuring programs don't stop responding or spend an unreasonable amount of time waiting for IO operations.

There are effectively two kinds of interfaces that we need to address:

- Those that already have a timeout argument (e.g. wait_readable) and we follow the existing semantics.
- Those that don't have a timeout argument or timeout semantics (e.g. puts, gets), and thus probably need to raise an exception on timeout.

We have three possible kinds of exceptions we could raise:

- Errno::ETIMEDOUT
- Timeout::Error (from timeout.rb)
- Introduce IO::Timeout or something similar.

Timeout isn't necessarily an error condition. There are different arguments for whether we should define:

```ruby
class IO::Timeout < Exception
end
```

or

```ruby
class IO::Timeout < StandardError
end
```

I believe the latter (StandardError) is more practical but I'm open to either option. I might have more specific arguments later why one is better than the other after testing in a practical system.

There is already a PR to try it out: [https://github.com/ruby/ruby/pull/5653](https://github.com/ruby/ruby/pull/5653)

**History**

#1 - 03/14/2022 02:43 AM - ioquatix (Samuel Williams)
- Description updated

#2 - 03/14/2022 02:44 AM - ioquatix (Samuel Williams)
- Description updated

#3 - 03/14/2022 04:28 AM - matz (Yukihiro Matsumoto)
I am positive about introducing safer timeout feature. But I have several concerns:

- time-out may happen from I/O blocking operations or other CPU bound operation. This proposal only covers I/O blocking. Is it OK?
- time-out may not be caused by a single I/O operation, but by multiple operations. This proposal should be updated to address it (maybe specifying time-bound by the specific time point).
- some I/O operation methods takes timeout keyword argument. We need to confirm they are consistent with this proposal.

Matz.

#4 - 03/14/2022 09:05 AM - Eregon (Benoit Daloze)
I'm not sure a timeout per IO instance makes sense, some IO operations might take longer e.g. reading many bytes at once and so it seems unclear whether any timeout value would be sensible there.
The proposal should also mention this can only work for non-blocking IOs (and maybe raise if called on a blocking IO?)

I thought Timeout.timeout is already good enough when there is a scheduler and has similar semantics, why would we need this? Is there a concrete example?

#5 - 03/14/2022 02:52 PM - ioquatix (Samuel Williams)

I'm not sure a timeout per IO instance makes sense, some IO operations might take longer e.g. reading many bytes at once and so it seems unclear whether any timeout value would be sensible there.

I think it's reasonable that no single IO operation should take more than, say, several seconds on a healthy system. However it's definitely a hard problem. I'm also considering whether we can have a general default, e.g. set by an environment variable or global within Ruby, e.g. IO.timeout. We could also consider adding keyword arguments to File.open and so on. A timeout is really a way of protecting code from hanging indefinitely, e.g. because of deadlocks or DoS hacks. A lot of programs monkey patch such functionality but none of it is compatible with each other. I think introducing a simple, standard interface here makes sense.

The proposal should also mention this can only work for non-blocking IOs (and maybe raise if called on a blocking IO?)

This only matters for general blocking Ruby. When using the fiber scheduler with io_uring, this limitation goes away.

We could fall back to the blocking Timeout.timeout semantics in non-scheduler blocking case, I just don't know if it's a good idea to over-complexify the implementation. The vast majority of IO in Ruby now is non-blocking by default. This mostly just applies to stdin, stdout, and stderr. I think we could even consider making stdin non-blocking by default.

In order to state this more clearly, we could document this limitation as "Timeouts are best effort and are not always guaranteed to be enforced or accurate." which is totally reasonable in my mind given the nature of timeouts.

I thought Timeout.timeout is already good enough when there is a scheduler and has similar semantics, why would we need this? Is there a concrete example?

Timeout.timeout is hard to implement and I'm not sure there is any general easy implementation. There is a fiber scheduler hook for Timeout.timeout which can be a little bit safer in practice at the expense of only interrupting non-blocking IO.

IO#timeout is more like the default timeout to use when internal wait mechanisms are invoked, like nogvl_wait_for or rb_io_wait. This is much more predictable and robust.

#6 - 03/17/2022 02:08 AM - ioquatix (Samuel Williams)

matz (Yukihiro Matsumoto) thanks for your comments.

time-out may happen from I/O blocking operations or other CPU bound operation. This proposal only covers I/O blocking. Is it OK?

I believe it's acceptable, since it only impacts IO operations.

I agree to introduce consistency here where possible. Some method like gets and puts would be hard to update, but others like read and write should be possible to add individual timeouts.

Later we could consider adding IO#read_timeout and IO#write_timeout if there is a valid use case.

#7 - 03/17/2022 12:06 PM - Eregon (Benoit Daloze)

This part wasn't answered:

Is there a concrete/practical example (for this new timeout)?

I'm not sure it's valuable to have per-IO timeout. It's kind of global state, especially if it's set on some globally-shared IO like STDIN/STDOUT/STDOUT.

Also this approach can't work when creating an IO, e.g. creating a new TCPSocket connecting to some address+port.
I think this might be redundant and add additional complexity and global state compared to a new way to do timeout that only interrupts IO and blocking operations (like Queue#pop, Mutex#lock, sleep, etc). i.e., same as Timeout.timeout except it would only interrupt blocking operations (blocking IO is considered a blocking operation too), not Thread#raise on some random Ruby line.

So we'd have Timeout.blocking_timeout(5) { ... } or so, and that's no global state, more general and more useful, isn't it?

And that should of course interrupt blocking IO via SIGVTALRM (and interruptible blocking regions in C exts) as already done for other functionality like Timeout.timeout.

#8 - 03/17/2022 03:42 PM - ioquatix (Samuel Williams)

I'm not sure it's valuable to have per-IO timeout.

async-io has used it for years successfully as a protection against slowloris attacks. It ensures that no matter who calls the IO operations, some timeout is associated with it.

There is a difference between IO which is usually externally facing and things like Queue, Thread which are internal. I personally have no problem adding timeouts to all those interfaces, and even adding Thread::Queue#timeout if that makes sense.

However, this PR is mostly just addressing the issue of making non-blocking IO safer in the presence of external malicious actors.