Ruby master - Bug #17420
Unsafe mutation of ":$" when doing non-RubyGems require in Ractor
12/21/2020 06:31 PM - Eregon (Benoit Daloze)

Status: Open
Priority: Normal
Assignee: ko1 (Koichi Sasada)
Target version:
ruby -v:
ruby 3.0.0dev (2020-12-16T10:12:48Z
master a9a7f4d6b8) [x86_64-linux]
Backport:
2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description
With an empty file a.rb:

$ ruby --disable-gems -e 'Ractor.new { puts ":$" }.take'
-e:1:in `block in <main>': can not access global variables ":$" from non-main Ractors (RuntimeError)

That is expected, given the rules for global variables.

ruby --disable-gems -e 'Ractor.new { require ":./a.rb"; }.take; p ":"
[... , ":/home/eregon/a.rb"]

Is it OK that the Ractor can do require, which does modify ":$"?

I think it's not, and it might lead to segfaults if e.g. the main Ractor mutates ":$" in parallel to some other Ractor doing require.

Probably require needs to be forbidden in non-main Ractors (it does mutate ":$", so it's logical), or there needs to be always VM-global synchronization on any access to ":$" (otherwise, segfaults are possible).
The latter doesn't seem reasonable, especially when considering the user might do ":$.each { ... }.

Note that RubyGems' require does not work on non-main Ractors (pretty much expected given it depends on a lot of global state):

$ ruby -e 'Ractor.new { require ":./a.rb"; }.take'
<internal:/home/eregon/prefix/ruby-master/lib/ruby/3.0.0/rubygems/core_ext/kernel_require.rb>:37:i
n `require': can not access non-shareable objects in constant Kernel::RUBYGEMS_ACTIVATION_MONITOR
by non-main ractor. (NameError)

This probably also has consequences for autoload.
Maybe the zeitwerk gem can help with the mode to resolve all autoload at once.

Related issues:
Related to Ruby master - Bug #17477: Ractor and pp incompatibility

History
#1 - 12/21/2020 06:32 PM - Eregon (Benoit Daloze)
- Description updated

#2 - 12/27/2020 04:19 PM - Eregon (Benoit Daloze)
- Related to Bug #17477: Ractor and pp incompatibility added

#3 - 12/27/2020 05:30 PM - marcandre (Marc-Andre Lafortune)
Maybe a solution would be to do all require in the main Reactor?

Something like whenever a new Ractor is created, the main thread could spin a private thread to do the needed require. When a require call happens, or a constant needs to be autoloaded, from within a Ractor other than main, inter-Ractor communication could be used (special tag to insure proper communication) to do the loading in the main thread.

#4 - 12/28/2020 08:43 AM - duerst (Martin Dürst)
I agree that it would be best to make require work everywhere, but always be executed in the main Reactor. That would just be part of the semantics of require (comment moved from #17477).

03/18/2022
That sounds like a good way to fix it.

marcandre (Marc-Andre Lafortune) wrote in #note-3:

the main thread could spin a private thread to do the needed require.

Whatever code is loaded by require can check Thread.current, so it should be a regular Ruby Thread, except the VM or the Ractor impl would create it.

kirs (Kir Shatrov) wrote in #note-3:

(coming to this bug from https://bugs.ruby-lang.org/issues/17477)

I think it's fine to force require to be called only from the main thread/ractor, at least for now. I can imagine that would simplify a lot of things.

I'd like to make that error very clear to developers, and not can not access non-shareable objects in constant Kernel::RUBYGEMS_ACTIVATION_MONITOR by non-main ractor that they see right now.

Current require behavior with ractors are not well considered and we need to consider it.

Maybe a solution would be to do all require in the main Ractor?

Yes, This is one good option. But not sure it is only one solution...

If we allow to require from non-main ractors, the only problem is $LOADED_FEATURES?

There are some limitations in non-main ractors, in particular no non-shareable constants, so if a gem is loaded from non-main ractor, something as trivial as VERSION = '0.0.1' in its code (without frozen string magic comment) could make the loading fail. In particular with autoload, the requiring might happen at difficult to predict times, so this behavior might happen only intermittently...

ko1 (Koichi Sasada) wrote in #note-7:

If we allow to require from non-main ractors, the only problem is $LOADED_FEATURES?

Not only, as we see above RubyGems uses a Monitor, and that doesn't work with Ractor. Making the entire logic for RubyGems' #require safe is probably not easy.

And as Marc-André says, any constant assignment with something not Ractor-shareable would fail, so basically almost every gem would fail to load in a Ractor.

So I think loading code in the main Ractor is the only realistic solution here. Either by having a clear error if require is used on non-main Ractor, or delegate the require to the main Ractor main Thread. One issue though is such a "delegation" really means interrupting the main Thread potentially at a random place (via interrupt checks I guess), which can be expensive (it might cancel no GVL code, which might need to redo a lot of work), and arbitrarily delayed (e.g., if the main Thread is in native code).

Actually having an extra Thread in the main Ractor to do the require's seems a much nicer solution, as proposed above (I forgot about it).