Ruby - Bug #17310

Closed ractors should die

11/08/2020 03:17 AM - marcandre (Marc-Andre Lafortune)

Status:	Closed		
Priority:	Normal		
Assignee:	ko1 (Koichi Sasada)		
Target version:			
ruby -v:	ruby 3.0.0dev (2020-11-07T21:47:45Z master 2f12af42f7) [x86_64-darwin18]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

While backporting Ractors, I found this issue:

```
10.times { Ractor.new { sleep(0.1) } }
sleep(1)
puts Ractor.count # => 1, ok
# but:
10.times { Ractor.new { sleep(0.1) }.close }
sleep(1)
Ractor.count # => 11, should be 1
```

Associated revisions

Revision deed21bb08170431891b65fda26f4a3557c9ffd4 - 11/11/2020 09:10 AM - ko1 (Koichi Sasada)

ignore yield_atexit if outgoing port is closed

If outgoing_port is closed, Ractor.yield never successes.
[Bug #17310]

Revision deed21bb08170431891b65fda26f4a3557c9ffd4 - 11/11/2020 09:10 AM - ko1 (Koichi Sasada)

ignore yield_atexit if outgoing port is closed

If outgoing_port is closed, Ractor.yield never successes.
[Bug #17310]

Revision deed21bb - 11/11/2020 09:10 AM - ko1 (Koichi Sasada)

ignore yield_atexit if outgoing port is closed

If outgoing_port is closed, Ractor.yield never successes.
[Bug #17310]

History

#1 - 11/08/2020 05:50 AM - marcandre (Marc-Andre Lafortune)

It also takes all the available CPU.

#2 - 11/10/2020 08:12 AM - ko1 (Koichi Sasada)

Thank you.

Flow:

1. close outgoing port
2. exit the block and try to yield the result
3. outgoing port is closed
4. raise an exception (ClosedError)
5. catch the exception, and try to yield the exception
6. goto 3

There are several options:

(1) we need to ignore the last yield if outgoing port is closed.
(2) remove close_outgoing

I have no strong motivation to provide Ractor#close_outgoing and same functionality of Ractor#close by other ractors.

I believe terminated ractors should close their own ports (incoming port and outgoing port) to tell its termination to other taking ractors.
Also I believe Ractor#close_incoming is needed to tell there is no more messages for the ractor.

However, I don't have strong opinion about close_outgoing.
This method is provided because it can be implemented.

So (2) is one idea, I guess.

#3 - 11/10/2020 06:34 PM - marcandre (Marc-Andre Lafortune)

Option 1 seems easy.

I don't have enough experience to know if Ractor#close_outgoing could be useful or not. I am assuming there will be a Ractor#kill, right?

#4 - 11/11/2020 07:29 AM - ko1 (Koichi Sasada)

At least, I merged (1) patch.

Ractor#kill is not acceptable to avoid non-deterministic behavior like introduced by Thread#kill.

I'm not sure we need Ractor#close_outgoing.
One possibility is to make a ractor detached (independent from any other ractors).
But I have no idea how to use such detached ractors.
Another possibility is notify the taking ractors to close earlier at exit phase.

```
Ractor.new do
  while msg = Ractor.recv
    Ractor.yield msg
  end
  close_outgoing # notify taking ractors before long_cleanup_code

  long_cleanup_code
end
```

I'm also not sure we can provide Ractor#close which calls close_incoming and close_outgoing.
They are different purpose, so I remove Ractor#close https://github.com/ruby/ruby/pull/3759
If they are used together frequently, we can re-introduce it.

#5 - 11/11/2020 09:11 AM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

Applied in changeset git|deed21bb08170431891b65fda26f4a3557c9ffd4.

---

ignore yield_atexit if outgoing port is closed

If outgoing_port is closed, Ractor.yield never successes.
[Bug #17310]

#6 - 11/11/2020 07:12 PM - Eregon (Benoit Daloze)

I think (2) is a better solution.
A Ractor should always be able to send messages while it's alive.

Another thought: maybe Ractor.new { 42 } should not automatically Ractor.yield the result?
Because Ractor.new {} does not wait for resume or so to start, so it seems asymmetric to yield the result (conceptually, we can see Fiber yields both before starting and when finishing).
It is convenient to Ractor.new { ... }.take, but we could have Ractor#join for that purpose, and that would also work more reliably, independent of intermediate Ractor.yield calls.

#7 - 11/11/2020 07:14 PM - Eregon (Benoit Daloze)

Also, it seems a big issue if exceptions in Ractor are silently ignored.
If we can't Ractor.yield the exception, then I think we should print it much like Thread.report_on_exception.

Not having Ractor#close_outgoing seems to solve it more cleanly though.

#8 - 11/11/2020 07:18 PM - Eregon (Benoit Daloze)

Eregon (Benoit Daloze) wrote in #note-7:

> Also, it seems a big issue if exceptions in Ractor are silently ignored.
> If we can't Ractor.yield the exception, then I think we should print it much like Thread.report_on_exception.

Sorry, I should have checked before, that's already the case.

#9 - 11/12/2020 01:07 AM - ko1 (Koichi Sasada)

> It is convenient to Ractor.new { ... }.take, but we could have Ractor#join for that purpose, and that would also work more reliably, independent of intermediate Ractor.yield calls.

When do you need Ractor#join?

#10 - 11/12/2020 01:19 AM - ko1 (Koichi Sasada)

Ractor is designed to manage blocking operations by

- receive
- yield and take

and they can be multiplex with Ractor.select. I don't want to introduce more.

I think yielder/taker can communicate on the protocol which contains notification about the end of yielder.

#11 - 11/13/2020 05:17 PM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote in #note-9:

> > It is convenient to Ractor.new { ... }.take, but we could have Ractor#join for that purpose, and that would also work more reliably, independent of intermediate Ractor.yield calls.
>
> When do you need Ractor#join?

I was thinking to the same use cases as Thread#join, I often want to wait for completion of some Ractors.
But probably we need the final value in most cases, so Ractor#value would make more sense?

Ractor.new { ... }.take might not be enough, because r=Ractor.new { n.times { Ractor.yield ... }; Ractor.yield }; while obj = r.take; ...; end; ensure r.join doesn't work.

> I don't want to introduce more.

Agreed.

> I think yielder/taker can communicate on the protocol which contains notification about the end of yielder.

Yeah, it's probably good enough.

One case I can think of where #join/value could be useful is if you want to wait for a few Ractor, ensure their cleanup runs, and there is still logic after so just waiting main thread exit is not good enough.

#12 - 11/13/2020 05:20 PM - Eregon (Benoit Daloze)

Sorry for the side discussion.

I think this is the important point to discuss:

Eregon (Benoit Daloze) wrote in #note-6:

> > I think (2) is a better solution.
> > A Ractor should always be able to send messages while it's alive.

I think in some actor models send is considered safe and never raising an exception, or only raising if the receiver is not alive.
close_outgoing seems to break that guarantee, because it might raise even if the receiver is alive.