Ruby - Bug #17678

Ractors do not restart after fork

03/08/2021 04:19 PM - ivoanjo (Ivo Anjo)

Status:	Closed		
Priority:	Normal		
Assignee:	ractor		
Target version:			
ruby -v:		**Backport:**	3.0: UNKNOWN, 3.1: UNKNOWN, 3.2: UNKNOWN

Description

Hello there! I'm working at Datadog on the ddtrace gem -- https://github.com/DataDog/dd-trace-rb and we're experimenting with using Ractors in our library but run into a few issues.

Background

When running a Ractor as a background process, the Ractor stops & does not restart when the application forks.

How to reproduce (Ruby version & script)

ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-linux]

```
r2 = Ractor.new do
 loop { puts "[#{Process.pid}] Ractor"; sleep(1) }
end

sleep(1)
puts "[#{Process.pid}] Forking..."

fork do
 sleep(5)
 puts "[#{Process.pid}] End fork."
end

loop do
 sleep(1)
end
```

Expectation and result

The application prints “Ractor” each second in the main process, but not in the fork.

Expected the Ractor (defined as r2) to run in the fork.

```
[29] Ractor
[29] Ractor
[29] Forking...
[29] Ractor
[29] Ractor
[29] Ractor
[29] Ractor
[29] Ractor
[32] End fork.
[29] Ractor
[29] Ractor
[29] Ractor
```

Additional notes

Threads do not restart across forks either, so it might not be unreasonable to expect consistent behavior. However, it’s possible to detect a dead Thread and recreate it after a fork (e.g. with #alive?, #status), but there’s no such mechanism for Ractors.

Suggested solutions

1. Auto-restart Ractors after fork
2. Add additional methods to Ractors that allow users to check & manage the status of the Ractor, similar to Thread.

History

#1 - 03/08/2021 04:21 PM - ivoanjo (Ivo Anjo)

- ruby -v set to ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-linux]

#2 - 03/09/2021 12:42 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned

- Assignee set to ko1 (Koichi Sasada)

#3 - 03/09/2021 12:42 AM - hsbt (Hiroshi SHIBATA)

- Tags set to ractor

#4 - 08/24/2023 09:12 PM - jeremyevans0 (Jeremy Evans)

- Tracker changed from Bug to Feature

- ruby -v deleted (ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-linux])

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN)

As Ractors always use separate OS threads, and fork only runs the current thread in the forked process, I don't see a way for Ractors to continue where they left off after fork. I think auto-starting would likely be a bad idea, because auto-starting would not return them to the state they were at fork.

The addition of Ractor#alive? and/or Ractor#status makes sense to me. Even in non-forked processes such methods could be useful. Note that you can get what you want already, by calling Ractor#inspect, so these methods would only need to expose information that Ractor is already storing.

#5 - 08/25/2023 08:12 AM - ivoanjo (Ivo Anjo)

> The addition of Ractor#alive? and/or Ractor#status makes sense to me. Even in non-forked processes such methods could be useful. Note that you can get what you want already, by calling Ractor#inspect, so these methods would only need to expose information that Ractor is already storing.

Thanks for looking into this!

I don't think the info is there in #inspect... At least I don't get it on stable or latest ruby-head?

Here's an updated example:

```
puts RUBY_DESCRIPTION

r2 = Ractor.new { puts "[#{Process.pid}] Ractor started!"; sleep(1000) }

puts "[#{Process.pid}] In parent process, ractor status is #{r2.inspect}"

sleep(1)
puts "[#{Process.pid}] Forking..."

fork do
  puts "[#{Process.pid}] In child process, ractor status is #{r2.inspect}"
end

Process.wait
```

and here's what I get:

```
$ ruby ractor-test.rb
ruby 3.3.0dev (2023-08-24T12:12:51Z master 5ec1fc52c1) [x86_64-linux]
ractor-test.rb:3: warning: Ractor is experimental, and the behavior may change in future versions of Ruby! Als
o there are many implementation issues.
[10] In parent process, ractor status is #<Ractor:#2 ractor-test.rb:3 blocking>
[10] Ractor started!
[10] Forking...
```

```
[12] In child process, ractor status is #<Ractor:#2 ractor-test.rb:3 blocking>
```

#6 - 08/25/2023 02:35 PM - jeremyevans0 (Jeremy Evans)

- Tracker changed from Feature to Bug

- Backport set to 3.0: UNKNOWN, 3.1: UNKNOWN, 3.2: UNKNOWN

Thanks for that information. It looks like ractors contain information about their state (shown in inspect), but that state is not updated on fork, unlike threads. So I think there is a bug, and it's that non-main ractors do not have their state to terminated upon fork (forking from non-main ractors is going to be prohibited, see #17516).

#7 - 08/25/2023 03:29 PM - ivoanjo (Ivo Anjo)

Ack, that seems a reasonable way of looking at this, having a way to detect that the ractor is dead would be enough to write some auto-restart code after fork (Erlang supervisor trees here we go :D).

#8 - 08/25/2023 04:29 PM - jeremyevans0 (Jeremy Evans)

I updated https://github.com/ruby/ruby/pull/8283 to mark non-main ractors as terminated after fork.

#9 - 05/08/2025 10:38 PM - jhawthorn (John Hawthorn)

- Assignee changed from ko1 (Koichi Sasada) to ractor

#10 - 05/08/2025 10:43 PM - jhawthorn (John Hawthorn)

- Status changed from Assigned to Closed

Sorry @jeremyevans0 (Jeremy Evans) I think Aaron and I missed that PR, but we did the same thing in https://github.com/ruby/ruby/pull/12982 which should fix this.

One thing we didn't do that's in your PR is forbid forking from within a Ractor, but that does seem like a prudent restriction (though it could be limiting for those wanting to fork, make some adjustments, exec)