Ruby - Feature #15912

Allow some reentrancy during TracePoint events

06/11/2019 01:25 PM - deivid (David Rodríguez)

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

Description

I got a report in byebug about byebug being incompatible with zeitwerk. This one: https://github.com/deivid-rodriguez/byebug/issues/564. This is a problem because zeitwerk is the default Rails code loader, and byebug is the default Rails debugger.

Both of these tools rely on the TracePoint API:

- Byebug uses a bunch of TracePoint events to stop execution at certain points in your program.
- Zeitwek uses :class events to be able to resolve some circular edge cases.

I investigated the problem and I think the issue is that while stopped at the byebug prompt, we're actually in the middle of processing a TracePoint event. That means that further TracePoint events triggered at the byebug's prompt will be ignored, because otherwise we could get into an infinite loop where the handling of events would trigger more events that trigger themselves the execution of handlers again.

I understand why the TracePoint API does this, but if we could allow some level of reentrancy here, we could probably make these tools play nice together. I figure if we kept a stack of TracePoint event handlers being run, and check that the current event type is not already in the stack, we would allow :class events to be triggered from :line events, and I think that would allow Zeitwerk to work within byebug.

What do you think about this, @ko1 (Koichi Sasada)?

Related issues:

Related to Ruby - Bug #16776: Regression in coverage library	**Assigned**

Associated revisions

Revision 9873af0b1a343dff6d1a8af4c813aa2c9ecc47d5 - 12/09/2021 03:56 PM - ko1 (Koichi Sasada)

TracePoint.allow_reentry

In general, while TracePoint callback is running,
other registerred callbacks are not called to avoid
confusion by reentrace.
This method allow the reentrace. This method should be
used carefully, otherwize the callback can be easily called
infinitely.

[Feature #15912]

Co-authored-by: Jean Boussier jean.boussier@gmail.com

Revision 9873af0b1a343dff6d1a8af4c813aa2c9ecc47d5 - 12/09/2021 03:56 PM - ko1 (Koichi Sasada)

TracePoint.allow_reentry

In general, while TracePoint callback is running,
other registerred callbacks are not called to avoid
confusion by reentrace.
This method allow the reentrace. This method should be
used carefully, otherwize the callback can be easily called
infinitely.

[Feature #15912]

Co-authored-by: Jean Boussier jean.boussier@gmail.com

Revision 9873af0b - 12/09/2021 03:56 PM - ko1 (Koichi Sasada)

TracePoint.allow_reentry

In general, while TracePoint callback is running,
other registerred callbacks are not called to avoid
confusion by reentrace.
This method allow the reentrace. This method should be
used carefully, otherwize the callback can be easily called
infinitely.

[Feature #15912]

Co-authored-by: Jean Boussier jean.boussier@gmail.com

History

#1 - 06/13/2019 11:41 PM - jeremyevans0 (Jeremy Evans)

- Tracker changed from Bug to Feature

- ruby -v deleted (ruby 2.6.3p62 (2019-04-16 revision 67580) [x86_64-linux])

- Backport deleted (2.4: UNKNOWN, 2.5: UNKNOWN, 2.6: UNKNOWN)

#2 - 07/29/2019 08:04 AM - ko1 (Koichi Sasada)

- Status changed from Open to Assigned

- Assignee set to ko1 (Koichi Sasada)

Sorry for inconvenient. Should we re-enable a hook in a block like TP.new{ ...; tp.reopen{ ... 'enable here'}; ...} ?

#3 - 07/31/2019 10:51 AM - deivid (David Rodríguez)

Hi Koichi! Thanks for answering.

So, you mean to allow the reentrancy (TracePoint events being triggered from inside TracePoint handlers) if the TracePoint API user requests so? I guess that would work, I could call tp.reopen {} before giving a prompt to the byebug user, so that zeitwerk events will trigger.

I wonder though whether it's better that this *just works* when it could work, and it ignores events when they would cause an infinite loop. I think my suggestion of keeping a stack of events being handled would allow that and would require no additions to the API.

#4 - 08/08/2019 05:50 AM - ko1 (Koichi Sasada)

In this case, your solution (allow fire different type of trace) because the problematic case :line event and :class event are different.
However, maybe it is accidentally. Someone can use :line event as usual.

reopen will allow re-occurrence explicitly by TracePoint users (like byebug), I think.
What do you think about it?

To introduce #reopen, we need to consider about:

- name (reopen? allow? unmask like signal mask idiom?).
- do we need to pass allowing events? (e.g.: all events except :line event are allowed)

#5 - 08/10/2019 03:54 PM - deivid (David Rodríguez)

Regarding your questions, reopen would work for me, and regarding passing events, I think a list of the events to be reopened like TracePoint.new(*events) would work.

If I understand what you are proposing, we would be giving full control to the user, and allowing it to cause infinite loops. For example, if I call tp.reopen(:line) during the execution of a :line event, I would get an infinite loop.

I still feel that a "just works" solution should be possible, even for handlers of the same type. Say we have line_handler1, and line_handler2 registered for :line events. Currently, when a :line event fires, we run handlers sequentially without allowing other events to fire during their execution. We could relax this restriction so that :line events could fire unless they're currently being handled, so that during execution of line_handler1, :line events could fire and run line_handler2.

```
-> Main program execution starts
-> Line event fired
   -> Execution of line_handler1 starts
   -> Line event fired
      -> Execution of line_handler2 starts
      -> Line event fired
      -> Line event fired
      -> ...
      -> Execution of line handler2 ends
  -> Line event fired
```

```
      -> Execution of line_handler2 starts
      -> Line event fired
      -> Line event fired
      -> ...
      -> Execution of line handler2 ends
  -> ...
  -> Execution of line_handler1 ends
-> ...
-> Main program ends
```

#6 - 08/13/2019 08:14 AM - ko1 (Koichi Sasada)

Your proposal is, prohibit "same" tracepoint object, right? I misreading that your suggestion is to prohibit same event (when line event handler is working, other line event handler can't be fired, but :class event handler can fire).

If my understanding is correctly, while running line_handler2, line_handler1 is invoked. right?

```
* main
  * :line event
    * line_handler1
      * :line event
        * line_handler2
      * :line event
        * line_handler2
    * line_handler2
      * :line event
        * line_handler1
      * :line event
        * line_handler1
    * finish all line hooks
* continue main
```

#7 - 08/13/2019 01:21 PM - deivid (David Rodríguez)

Yes, that was initial proposal (to prohibit other events of the same type, while a handler for a certain event type is running). But then I thought that we could even allow events of the same type, as long as we forbid handlers currently being run from being triggered.

So in my example, line_handler2 executions in lines 5 and 7 would not run line_handler1, but the execution in line 8 would run it (twice, actually).

```
 1: * main
 2:   * :line event
 3:     * line_handler1
 4:       * :line event
 5:         * line_handler2
 6:       * :line event
 7:         * line_handler2
 8:     * line_handler2
 9:       * :line event
10:         * line_handler1
11:       * :line event
12:         * line_handler1
13:   * finish all line hooks
14: * continue main
```

The idea is running any possible relevant event handler as long as it's not already in the stack of event handlers currently being run.

#8 - 08/14/2019 06:07 AM - ko1 (Koichi Sasada)

> But then I thought that we could even allow events of the same type, as long as we forbid handlers currently being run from being triggered.
> So in my example, line_handler2 executions in lines 5 and 7 would not run line_handler1, but the execution in line 8 would run it (twice, actually).
> ...
> The idea is running any possible relevant event handler as long as it's not already in the stack of event handlers currently being run.

Why line 5, line 7 would not invoked? I can't understand the rule.
And how about line 10, 12?

#9 - 08/14/2019 10:36 AM - deivid (David Rodríguez)

The idea is to avoid recursive calls to the same event, but allow other kind of reentrancy. With a real script:

```
line_handler1 = TracePoint.trace(:line) do |tp|                    # L1
```

```
  puts "Handler 1 starts (triggered from #{tp.path}:#{tp.lineno})"   # L2
  puts "Handler 1 ends (triggered from #{tp.path}:#{tp.lineno})"     # L3
end                                                                  # L4
                                                                     # L5
line_handler2 = TracePoint.trace(:line) do |tp|                      # L6
  puts "Handler 2 starts (triggered from #{tp.path}:#{tp.lineno})"   # L7
  puts "Handler 2 ends (triggered from #{tp.path}:#{tp.lineno})"     # L8
end                                                                  # L9
                                                                     # L10
puts "I'm a line"                                                    # L11
```

Current output is

> Handler 1 starts (triggered by line tp.rb:6)
> Handler 1 ends (triggered by line tp.rb:6)
> Handler 2 starts (triggered by line tp.rb:11)
> Handler 2 ends (triggered by line tp.rb:11)
> Handler 1 starts (triggered by line tp.rb:11)
> Handler 1 ends (triggered by line tp.rb:11)
> I'm a line

Proposed output would be

> Handler 1 starts (triggered by line tp.rb:6)
> Handler 1 ends (triggered by line tp.rb:6)
> Handler 2 starts (triggered by line tp:11)
> Handler 1 starts (triggered by line tp:7)
> Handler 1 ends (triggered by line tp:7)
> Handler 2 ends (triggered by line tp:11)
> Handler 1 starts (triggered by line tp:8)
> Handler 1 ends (triggered by line tp:8)
> I'm a line

By maybe the explicit solution you propose is better: allow every event to be executed via TracePoint#reopen including for code inside handlers, and let the user be in control of avoiding potential infinite loops.

#10 - 10/03/2019 03:32 PM - deivid (David Rodríguez)

I tried my idea and, while it seemed not hard to implement, it's not going to work, because many many more event would be generated and that breaks byebug and I'm guessing other TracePoint API consumers. It's also not straightforward to communicate and understand, so I think your solution of adding TracepPoint#reopen is better and more explicit.

I can try to implement it but I'm not sure how it will go.

#11 - 12/04/2021 08:32 AM - byroot (Jean Boussier)

I took the liberty to re-add this to the next developer meeting because ruby/debug appear to have the same problem:
https://github.com/ruby/debug/issues/408

#12 - 12/04/2021 10:55 AM - Eregon (Benoit Daloze)

I think the idea to explicitly reopen/allow recursive events is good.
For common usages of tracepoint I think it's a good idea to avoid any tracepoint event firing from a tracepoint handler,
but in such case such as the debugger executing arbitrary code inside a :line tracepoint then reopening seems exactly the desired semantics.

#13 - 12/06/2021 02:53 AM - mame (Yusuke Endoh)

- Related to Bug #16776: Regression in coverage library added

#14 - 12/07/2021 06:43 AM - ko1 (Koichi Sasada)

naming issue:

TracePoint.allow_reentrance do ... end ?
TracePoint.allow_reentrancy do ... end ?

#15 - 12/07/2021 08:30 AM - byroot (Jean Boussier)

What about:

```
TracePoint.trace(:line) do |tp|
  tp.reentrant do
    ...
```

```
  end
end
```

#16 - 12/08/2021 06:54 PM - ko1 (Koichi Sasada)

implementation: https://github.com/ruby/ruby/pull/5231

> tp.reentrant

It should not be an instance method of tp because it doesn't depend on current tp.
Also this is danger API, so I want to add allow_ to make clear the meaning.

#17 - 12/09/2021 08:16 AM - duerst (Martin Dürst)

ko1 (Koichi Sasada) wrote in #note-14:

> naming issue:
>
> TracePoint.allow_reentrance do ... end ?
> TracePoint.allow_reentrancy do ... end ?

I'm not completely sure about what would be at ..., but maybe allow_reentry or allow_reenter should also be considered as method names, because they are shorter.

#18 - 12/09/2021 08:47 AM - ko1 (Koichi Sasada)

Thank you. also @byroot (Jean Boussier) and colleagues proposed TracePoint.allow_reentry, so I change the name.
Matz also accepted the name and feature, so I'll merge it.

#19 - 12/09/2021 03:56 PM - ko1 (Koichi Sasada)

- Status changed from Assigned to Closed

Applied in changeset git|9873af0b1a343dff6d1a8af4c813aa2c9ecc47d5.

TracePoint.allow_reentry

In general, while TracePoint callback is running,
other registerred callbacks are not called to avoid
confusion by reentrace.
This method allow the reentrace. This method should be
used carefully, otherwize the callback can be easily called
infinitely.

[Feature #15912]

Co-authored-by: Jean Boussier jean.boussier@gmail.com