Ruby - Bug #18031

Nested TracePoint#enable with define_method target crashes

07/08/2021 11:53 PM - alanwu (Alan Wu)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN

Description

Found this crash while looking at tracing related code.
Haven't had time to dig deeper, so I'm making a ticket for now.

```
one = TracePoint.new(:call) {}
two = TracePoint.new(:call) {}

obj = Object.new
obj.define_singleton_method(:foo) {} # a bmethod

foo = obj.method(:foo)
one.enable(target: foo) do
  two.enable(target: foo) {}
end
```

It crashes on 2.6.5 and master (a7c85cc).

Associated revisions

Revision e75cb61d4645b9833b14a0cc4190cceffc9b6551 - 06/10/2022 01:10 AM - alanwu (Alan Wu)

Fix nested bmethod TracePoint and memory leak

df317151a5b4e0c5a30fcc321a9dc6abad63f7ed removed the code to free
rb_hook_list_t, so repeated targeting of the same bmethod started
to leak the hook list. You can observe how the maximum memory use
scales with input size in the following script with /usr/bin/time -v.

```
o = Object.new
o.define_singleton_method(:foo) {}
trace = TracePoint.new(:return) {}

bmethod = o.method(:foo)

ARGV.first.to_i.times { trace.enable(target:bmethod){} }
4.times {GC.start}
```

After this change the maximum doesn't grow as quickly.

To plug the leak, check whether the hook list is already allocated
when enabling the targeting TracePoint for the bmethod. This fix
also allows multiple TracePoints to target the same bmethod, similar
to other valid TracePoint targets.

Finally, free the rb_hook_list_t struct when freeing the method
definition it lives on. Freeing in the GC is a good way to avoid
lifetime problems similar to the one fixed in df31715.

[Bug #18031]

Revision e75cb61d4645b9833b14a0cc4190cceffc9b6551 - 06/10/2022 01:10 AM - alanwu (Alan Wu)

Fix nested bmethod TracePoint and memory leak

df317151a5b4e0c5a30fcc321a9dc6abad63f7ed removed the code to free
rb_hook_list_t, so repeated targeting of the same bmethod started
to leak the hook list. You can observe how the maximum memory use

scales with input size in the following script with /usr/bin/time -v.

```
o = Object.new
o.define_singleton_method(:foo) {}
trace = TracePoint.new(:return) {}

bmethod = o.method(:foo)

ARGV.first.to_i.times { trace.enable(target:bmethod){} }
4.times {GC.start}
```

After this change the maximum doesn't grow as quickly.

To plug the leak, check whether the hook list is already allocated
when enabling the targeting TracePoint for the bmethod. This fix
also allows multiple TracePoints to target the same bmethod, similar
to other valid TracePoint targets.

Finally, free the rb_hook_list_t struct when freeing the method
definition it lives on. Freeing in the GC is a good way to avoid
lifetime problems similar to the one fixed in df31715.

[Bug #18031]

Revision e75cb61d - 06/10/2022 01:10 AM - alanwu (Alan Wu)

Fix nested bmethod TracePoint and memory leak

df317151a5b4e0c5a30fcc321a9dc6abad63f7ed removed the code to free
rb_hook_list_t, so repeated targeting of the same bmethod started
to leak the hook list. You can observe how the maximum memory use
scales with input size in the following script with /usr/bin/time -v.

```
o = Object.new
o.define_singleton_method(:foo) {}
trace = TracePoint.new(:return) {}

bmethod = o.method(:foo)

ARGV.first.to_i.times { trace.enable(target:bmethod){} }
4.times {GC.start}
```

After this change the maximum doesn't grow as quickly.

To plug the leak, check whether the hook list is already allocated
when enabling the targeting TracePoint for the bmethod. This fix
also allows multiple TracePoints to target the same bmethod, similar
to other valid TracePoint targets.

Finally, free the rb_hook_list_t struct when freeing the method
definition it lives on. Freeing in the GC is a good way to avoid
lifetime problems similar to the one fixed in df31715.

[Bug #18031]

History

#1 - 07/10/2021 12:01 AM - jeremyevans0 (Jeremy Evans)

I've submitted a pull request to fix this crash: https://github.com/ruby/ruby/pull/4640

I noticed two other issues:

1. Tracepoints leak memory. I think this is definitely a bug. Example code:

   ```
   loop do
     tp = TracePoint.new(:call){}
     tp.enable
     tp.disable
   end
   ```

2. Use multiple tracepoints on the same target ends up with the second tracepoint overwriting the first. So inside the two.enable block in the original code, if you call foo, the one tracepoint isn't called, and after the two.enable block (still inside the one.enable block), neither tracepoint is called. I'm not sure whether this is a bug, but the behavior seems suprising.

#2 - 07/14/2021 11:54 PM - alanwu (Alan Wu)

I don't think the overriding behavior is intentional, since it doesn't happen for other kinds of targets.
There is an existing test case that tests this for Proc.
I also found a leak specific to targeting bmethods while looking at this bug.

Here's my PR to fix these two problems assuming my assessment is right:
https://github.com/ruby/ruby/pull/4651

#3 - 08/03/2021 09:18 PM - alanwu (Alan Wu)

- Subject changed from Nested TracePoint#enable with target crashes to Nested TracePoint#enable with define_method target crashes

#4 - 06/11/2022 05:05 AM - alanwu (Alan Wu)

- Status changed from Open to Closed

Applied in changeset git|e75cb61d4645b9833b14a0cc4190cceffc9b6551.

Fix nested bmethod TracePoint and memory leak

df317151a5b4e0c5a30fcc321a9dc6abad63f7ed removed the code to free
rb_hook_list_t, so repeated targeting of the same bmethod started
to leak the hook list. You can observe how the maximum memory use
scales with input size in the following script with /usr/bin/time -v.

```
o = Object.new
o.define_singleton_method(:foo) {}
trace = TracePoint.new(:return) {}

bmethod = o.method(:foo)

ARGV.first.to_i.times { trace.enable(target:bmethod){} }
4.times {GC.start}
```

After this change the maximum doesn't grow as quickly.

To plug the leak, check whether the hook list is already allocated
when enabling the targeting TracePoint for the bmethod. This fix
also allows multiple TracePoints to target the same bmethod, similar
to other valid TracePoint targets.

Finally, free the rb_hook_list_t struct when freeing the method
definition it lives on. Freeing in the GC is a good way to avoid
lifetime problems similar to the one fixed in df31715.

[Bug #18031]