# Ruby - Bug #7976

## TracePoint call is at call point, not call site

02/27/2013 10:25 AM - zenspider (Ryan Davis)

| | | | |
|---|---|---|---|
| **Status:** | Closed | | |
| **Priority:** | Normal | | |
| **Assignee:** | ko1 (Koichi Sasada) | | |
| **Target version:** | | | |
| **ruby -v:** | 2.0 | **Backport:** | |

**Description**

Using TracePoint to make a new tracer utility I'm finding it very difficult to actually trace where the origin is for type :call. Instead, I get the destination. This is not the case for :c_call or :b_call as they trace at the origin, not destination.

Reproduction attached. Notice how it outputs ":call wtf.rb:08 :x" where line 8 is the definition of x, not the call of x, yet the subsequent backtrace lists line 21 which is the original origin for the call to x.

### History

#### #1 - 03/12/2013 03:17 AM - tenderlovemaking (Aaron Patterson)

*- Category changed from YARV to core*

#### #2 - 06/30/2013 09:26 PM - deivid (David Rodríguez)

I find this behaviour satisfactory for my purposes.

If I need the origin of a :call event I use the previous :line event. Is this wrong?

#### #3 - 09/30/2013 08:27 PM - ko1 (Koichi Sasada)

*- Target version set to 2.1.0*

#### #4 - 10/10/2013 06:06 PM - ko1 (Koichi Sasada)

*- Status changed from Open to Feedback*

This behavior is same as backtrace.
Nearest file name and line number is used for C call,
because there are no file name and line for C methods.

```
# modify trace as follows:
tp = TracePoint.new(:call, :return, :c_call, :c_return, :raise) do |t|
  p caller(1, 1)[0].inspect + ("%-9p %s:%02d %p" % [t.event, t.path, t.lineno, t.method_id])
end

#=>
ruby 2.1.0dev (2013-10-10 trunk 43236) [i386-mswin32_110]
"\"tp_test.rb:5:in `<main>'\":c_return tp_test.rb:05 :enable"
"\"tp_test.rb:7:in `<main>'\":c_call   tp_test.rb:07 :inherited"
"\"tp_test.rb:7:in `<main>'\":c_return tp_test.rb:07 :inherited"
"\"tp_test.rb:8:in `<class:X>'\":c_call   tp_test.rb:08 :method_added"
"\"tp_test.rb:8:in `<class:X>'\":c_return tp_test.rb:08 :method_added"
"\"tp_test.rb:12:in `<class:X>'\":c_call   tp_test.rb:12 :method_added"
"\"tp_test.rb:12:in `<class:X>'\":c_return tp_test.rb:12 :method_added"
"\"tp_test.rb:16:in `<class:X>'\":c_call   tp_test.rb:16 :method_added"
"\"tp_test.rb:16:in `<class:X>'\":c_return tp_test.rb:16 :method_added"
"\"tp_test.rb:21:in `<main>'\":c_call   tp_test.rb:21 :new"
"\"tp_test.rb:21:in `new'\":c_call   tp_test.rb:21 :initialize"
"\"tp_test.rb:21:in `new'\":c_return tp_test.rb:21 :initialize"
"\"tp_test.rb:21:in `<main>'\":c_return tp_test.rb:21 :new"
"\"tp_test.rb:8:in `x'\":call     tp_test.rb:08 :x"
"\"tp_test.rb:12:in `y'\":call     tp_test.rb:12 :y"
"\"tp_test.rb:16:in `z'\":call     tp_test.rb:16 :z"
"\"tp_test.rb:17:in `z'\":c_call   tp_test.rb:17 :raise"
"\"tp_test.rb:17:in `raise'\":c_call   tp_test.rb:17 :new"
"\"tp_test.rb:17:in `new'\":c_call   tp_test.rb:17 :initialize"
"\"tp_test.rb:17:in `new'\":c_return tp_test.rb:17 :initialize"
"\"tp_test.rb:17:in `raise'\":c_return tp_test.rb:17 :new"
```

```
"\"tp_test.rb:17:in `z'\":c_call   tp_test.rb:17 :backtrace"
"\"tp_test.rb:17:in `z'\":c_return tp_test.rb:17 :backtrace"
"\"tp_test.rb:17:in `z'\":raise    tp_test.rb:17 :z"
"\"tp_test.rb:17:in `z'\":c_return tp_test.rb:17 :raise"
"\"tp_test.rb:17:in `z'\":return   tp_test.rb:17 :z"
"\"tp_test.rb:13:in `y'\":return   tp_test.rb:13 :y"
"\"tp_test.rb:9:in `x'\":return   tp_test.rb:09 :x"
tp_test.rb:17:in `z': no (RuntimeError)
 from tp_test.rb:13:in `y'
 from tp_test.rb:9:in `x'
 from tp_test.rb:21:in `<main>'
```

And binding is also nearest Ruby level binding.

```
  tp = TracePoint.trace(:call, :return, :c_call, :c_return, :raise) do |t|
    p [t, t.self, t.binding.eval('self')]
  end

  tp.enable

#=>
ruby 2.1.0dev (2013-10-10 trunk 43236) [i386-mswin32_110]
[#<TracePoint:c_return `trace'@tp_test.rb:1>, TracePoint, main]
[#<TracePoint:c_call `enable'@tp_test.rb:5>, #<TracePoint:c_call `enable'@tp_test.rb:5>, main]
[#<TracePoint:c_return `enable'@tp_test.rb:5>, #<TracePoint:c_return `enable'@tp_test.rb:5>, main]
[#<TracePoint:c_return `enable'@tp_test.rb:5>, #<TracePoint:c_return `enable'@tp_test.rb:5>, main]
[#<TracePoint:c_call `times'@tp_test.rb:7>, 1, main]
[#<TracePoint:c_call `times'@tp_test.rb:7>, 1, main]
[#<TracePoint:c_return `times'@tp_test.rb:7>, 1, main]
[#<TracePoint:c_return `times'@tp_test.rb:7>, 1, main]
```

self of times' method is 1'. However, binding.eval('self') returns main. This means that binding is also used Ruby level.
This is limitation of current TracePoint.

What should be happen on such case?

#### #5 - 01/30/2014 06:16 AM - hsbt (Hiroshi SHIBATA)

*- Target version changed from 2.1.0 to 2.2.0*

#### #6 - 03/02/2017 02:56 PM - jahfer (Jahfer Husain)

I recently found this behaviour while attempting to build a gem that outputs the full call graph for our Rails application.

Using TracePoint, we can capture all calls/returns with the object the method is invoked upon, but finding the actual site of the invocation is a daunting task. My gem is a C-extension, and after much experimentation, the only reliable way to get the correct caller is through using caller_locations, which computes the full backtrace from the VM and is a significant slowdown when compared to using rb_tracearg_path.

As a point of comparison, we can run 63k tests (parallelized) while invoking rb_tracearg_path in roughly 10 minutes, whereas using caller_locations takes about 1.5 hours to complete.

Regardless of how the file paths are computed, their accuracy is significant for the performance of our gem because we are able to use a blacklist to ignore logging all gem method invocations as well as Rails core or other infrastructural pieces that we do not need to trace.

I also attempted to use :line events as David Rodríguez suggested above, but on returns the value is always pointing to the deepest part of the stack most recently touched. The method does work well for calls though (albeit expensive to trace every line event).

You can see the different results (along with my personal expectation of what *should* be returned) in this gist:
https://gist.github.com/jahfer/092e44df6f15298bae3cf47056301708#gistcomment-2015893

#### #7 - 04/17/2017 07:59 AM - shyouhei (Shyouhei Urabe)

*- Status changed from Feedback to Assigned*

#### #8 - 01/05/2018 09:00 PM - naruse (Yui NARUSE)

*- Target version deleted (2.2.0)*

#### #9 - 05/10/2023 01:00 AM - jeremyevans0 (Jeremy Evans)

*- Status changed from Assigned to Closed*

@headius (Charles Nutter) and I agree this is not a bug. JRuby has similar behavior for Java extensions as CRuby does for C extensions.

**Files**

| | | | |
|---|---|---|---|
| tp_test.rb | 1.21 KB | 02/27/2013 | zenspider (Ryan Davis) |