Ruby master - Bug #15303
Return tracepoint doesn't fire when tailcall optimization is applied

11/14/2018 07:13 PM - alanwu (Alan Wu)

Status: Closed
Priority: Normal
Assignee:
Target version:
ruby -v: ruby 2.6.0dev (2018-11-14 trunk 65727) [x86_64-darwin17]
Backport: 2.4: UNKNOWN, 2.5: REQUIRED

Description
After a tailcall, the "return" tracepoint event is only fired once. Normally, after a call at the end of a method, the return event is fired twice, once for the callee returning and once for the caller returning.

The following script outputs

```
call
call
call
:return
```

```
method_source = <<-RB
def toy(n)
  return if n == 2
  toy(n+1)
end
RB

iseq = RubyVM::InstructionSequence.compile(method_source, tailcall_optimization: true)
#puts iseq.disasm
iseq.eval

trace = TracePoint.new(:call, :return) do |tp|
  p tp.event
end

trace.enable
toy(0)
```

The "return" event behaves more like a "stack frame pop" event currently. I don't think it's feasible/desirable to have the same behavior when TCO is applied, but it would be nice if there was some way for the tracepoint to know a tail call is going to happen.

I'm raising this issue because the popular debugger "byebug" relies on these events to track execution in various stack frames.

https://github.com/deivid-rodriguez/byebug/issues/481
Forwardable explicitly uses TCO which triggers this issue.

Related issues:
Has duplicate Ruby master - Bug #15604: Backport r66349 to 2.5

Associated revisions
Revision 241dced6 - 12/12/2018 05:46 AM - nobu (Nobuyoshi Nakada)
Disable tailcall optimization [Bug #15303]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@66349 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 66349 - 12/12/2018 05:46 AM - nobu (Nobuyoshi Nakada)
Disable tailcall optimization [Bug #15303]

Revision 66349 - 12/12/2018 05:46 AM - nobu (Nobuyoshi Nakada)
Disable tailcall optimization [Bug #15303]

History
#1 - 11/14/2018 07:56 PM - alanwu (Alan Wu)
- ruby -v set to ruby 2.6.0dev (2018-11-14 trunk 65727) [x86_64-darwin17]

#2 - 12/05/2018 07:03 AM - ko1 (Koichi Sasada)

hi,

how about to call return event hook at tailcall? There is no need to introduce new API proposed at [Bug #15313].

def bar
end

iseq = RubyVM::InstructionSequence.new("def foo; bar; end", __FILE__, __FILE__, __LINE__, tailcall_optimization: true)
iseq.eval

TracePoint.new(:call, :return) { |tp| p tp }.enable{
  foo
}

#<TracePoint:call `foo'@/home/ko1/src/ruby/gitruby/test.rb:4>
#<TracePoint:return `foo'@/home/ko1/src/ruby/gitruby/test.rb:4>
#<TracePoint:call `bar'@/home/ko1/src/ruby/gitruby/test.rb:3>
#<TracePoint:return `bar'@/home/ko1/src/ruby/gitruby/test.rb:2>

patch:

diff --git a/compile.c b/compile.c
index d97cf4e8c6..696e127b9a 100644
--- a/compile.c
+++ b/compile.c
@@ -3144,13 +3144,15 @@
     ci->flag |= VM_CALL_TAILCALL;
     goto change_to_tailcall;
   }
+}
  } else {
    change_to_tailcall:
    ci->flag |= VM_CALL_TAILCALL;
+    iobj->insn_info.events = RUBY_EVENT_RETURN;
  }
}

#3 - 12/05/2018 03:50 PM - alanwu (Alan Wu)

The main concern I have with this approach is that there is no reasonable value we can make Tracepoint#return_value return when we trigger it due to a tailcall:

def bar
  "bar's return value"
end

iseq = RubyVM::InstructionSequence.new("def foo; bar; end", __FILE__, __FILE__, __LINE__, tailcall_optimization: true)
iseq.eval

TracePoint.new(:return) { |tp| p tp |p tp.return_value |.enable{
  foo
}

Outputs the following with the patch from [https://bugs.ruby-lang.org/issues/15303#note-2](https://bugs.ruby-lang.org/issues/15303#note-2) applied:

#<TracePoint:return `foo'@./test.rb:5>
Clearly, foo doesn't return main; there is no meaningful value we can put for #return_value since semantically, foo hasn't returned yet. With this patch applied, #return_value is not reliable anymore for return events. I made #15313 because I felt it's the most backward compatible approach.

**#4 - 12/05/2018 04:33 PM - ko1 (Koichi Sasada)**

Yes, the patch should return nil (I forgot to support it).

I think we can change the semantic of tailcall as a method call/return flow I proposed.

What kind of issue on this change?

**#5 - 12/05/2018 04:50 PM - alanwu (Alan Wu)**

If we return nil for #return_value, users can't tell whether the nil is from the method actually returning nil, or a nil from tailcall.

I think the return event should be reserved for only real returns, because of its name.

I don't think this is a huge issue but I'm uncomfortable with ignoring it.

**#6 - 12/05/2018 05:02 PM - ko1 (Koichi Sasada)**

or raise on return_value?

it is irregular case. I think tailcall is not normal call (it is same as a method call by lexical).

foo() # tailcall

we can see:

```
return_and_call(:foo)
```

and nobody think it is strange.

in other words, using tailcall, we can define it as "transformation from former syntax to latter syntax~ we can think.

I want to know the real issue by this spec, from real users such as debugger creators.

**#7 - 12/05/2018 05:19 PM - alanwu (Alan Wu)**

I personally don't see an issue with this spec, besides that it would introduce a breaking change one way or the other. I think it would fix the next command in byebug, though. (it would have to rescue when it calls #return_value, though)

I've asked David Rodríguez, the creator of Byebug to join the discussion on Github. Hopefully we get to hear from him.

**#8 - 12/08/2018 01:31 AM - ko1 (Koichi Sasada)**

Ah, I have another strong idea (forgot to propose).

Cancel tailcall opt when return event is specified.

**#9 - 12/08/2018 06:33 AM - alanwu (Alan Wu)**

- File no-tco-no-problem.patch added

ko1 (Koichi Sasada) wrote:

```
Cancel tailcall opt when return event is specified.
```

I think anyone who goes through all the trouble to enable tailcall elimination right now relies on it to avoid stack overflow. Why else would someone use it? It doesn't make your program faster, it's cumbersome to enable, and it erases part of your stack trace which makes debugging harder.

Making this change seems to me replacing a problem with a different one. It is a problem that affects way less people admittedly, but I feel that it would be punishing functional programmers for the benefit of everyone else.

If we just want to fix the problem most people have today, i.e., byebug's next command's breakage, making Forwardable not use TCO is good enough. It will still be broken for tailcall optimized code, but at least most people won't run into this problem. That might be the most pragmatic "fix", and it's a simple one. With it, we can keep ignoring this problem and wait till someone complains.

**#10 - 12/11/2018 10:20 PM - deivid (David Rodríguez)**
Hello! I have very similar views to alanwu.

The original proposal of extending the API to be able to distinguish between regular method calls and tail calls sounds good to me in principle and looks like it would solve the problem in the general case. It would be a niche feature, but tailcall optimization is a niche feature too, right? It just happens to be used in a standard lib method, that's why it's biting byebug.

Regarding koichi's first proposal, it's not bad either. The return_value method in return events is only marginally used in byebug, so it's not a big deal to lose it on these edge cases.

Regarding koichi's last proposal.. Again I agree with alanwu. Couldn't using the tracepoint API over programs relying on tailcall optimization actually break those programs?

| #11 - 12/12/2018 05:46 AM - nobu (Nobuyoshi Nakada) | Status changed from Open to Closed |
| Applied in changeset trunk\r66349. |

Disable tailcall optimization [Bug #15303]

| #12 - 12/12/2018 06:06 AM - ko1 (Koichi Sasada) |
| Matz decided to remove tailcall opt from core libraries. |
| r66349 was committed by nobu. |

| #13 - 12/12/2018 12:07 PM - deivid (David Rodriguez) |
| Thank you! That works for me too, and will solve the majority of the cases :) |

| #14 - 04/23/2019 07:14 AM - nagachika (Tomoyuki Chikanaga) |
| Backport changed from 2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN to 2.4: UNKNOWN, 2.5: REQUIRED |

| #15 - 04/23/2019 07:14 AM - nagachika (Tomoyuki Chikanaga) |
| Has duplicate Bug #15604: Backport r66349 to 2.5 added |

| Files | |
| no-tco-no-problem.patch | 520 Bytes | 12/08/2018 | alanwu (Alan Wu) |