Ruby master - Bug #13774

for methods defined from procs, the binding of the resulting bound_method.to_proc does not have access to the original proc's closure environment

07/27/2017 05:39 PM - bughit (bug hit)

Status:	Rejected		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.5.0p0 (2017-12-25 revision 61468) [x86_64-linux]	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

```
def foo
  o = Object.new
  local_var = 'local_var'
  method_lambda = ->{local_var}
  o.define_singleton_method(:lambda_method, &method_lambda)
  method_proc = o.method(:lambda_method).to_proc
  puts method_proc.()
  puts eval('local_var', method_lambda.binding)
  puts o.lambda_method
  puts eval('local_var', method_proc.binding) # undefined local variable or method `local_var'
end

foo
```

when executed as a method it has access to local_var as expected
but the binding of the bound method proc does not

in ruby 2.2.2 the above code crashes the vm

History

#1 - 07/27/2017 05:55 PM - bughit (bug hit)

bughit (bug hit) wrote:

> ```
> def foo
>   o = Object.new
>   local_var = 'local_var'
>   method_lambda = ->{local_var}
>   o.define_singleton_method(:lambda_method, &method_lambda)
>   method_proc = o.method(:lambda_method).to_proc
>   puts method_proc.()
>   puts eval('local_var', method_lambda.binding)
>   puts o.lambda_method
>   puts eval('local_var', method_proc.binding) # undefined local variable or method `local_var'
> end
>
> foo
> ```
>
> when executed as a method it has access to local_var as expected
> but the binding of the bound method proc does not
>
> in ruby 2.2.2 the above code crashes the vm

should also underscore that the bound method proc has access to local_var: method_proc.()
but its binding does not eval('local_var', method_proc.binding), so there's a clear discrepancy between a proc and its binding

#2 - 12/27/2017 05:27 PM - bughit (bug hit)

- ruby -v changed from ruby 2.4.1p111 (2017-03-22 revision 58053) [x86_64-linux] to ruby 2.5.0p0 (2017-12-25 revision 61468) [x86_64-linux]

- Subject changed from for methods defined from procs, the binding of the resulting bound method proc does not have access to the original proc's closure environment to for methods defined from procs, the binding of the resulting bound_method.to_proc does not have access to the original proc's

closure environment

```
def foo
  o = Object.new
  local_var = 'has access to captured variable'

  original_proc = ->{local_var}

  o.define_singleton_method(:method_from_proc, &original_proc)
  method_proc = o.method(:method_from_proc).to_proc

  puts "original_proc.call #{original_proc.()}"
  puts "original_proc.binding #{original_proc.binding.eval('local_var')}"

  puts "#method_from_proc #{o.method_from_proc}"

  puts "method(:method_from_proc).to_proc.call #{method_proc.()}"
  puts "method(:method_from_proc).to_proc.binding #{method_proc.binding.eval('local_var') rescue 
"does not have access to captured variable: #{$!.inspect}"}"

end

foo
```

method(:method_from_proc).to_proc.binding does not have access to captured variable: #<NameError: undefined local variable or methodlocal_var'`

This is a bug?

#3 - 12/28/2017 08:21 AM - nobu (Nobuyoshi Nakada)

I think it is not a bug.

Local variables are not set up in the context of a binding from a method.

```
$ ruby -e 'def m(v=12);v;end; p method(:m).to_proc.binding.eval("v")'
Traceback (most recent call last):
    2: from -e:1:in `<main>'
    1: from -e:1:in `eval'
-e:1:in `<empty iseq>': undefined local variable or method `v' for main:Object (NameError)
```

#4 - 12/28/2017 04:40 PM - bughit (bug hit)

nobu (Nobuyoshi Nakada) wrote:

> I think it is not a bug.
>
> Local variables are not set up in the context of a binding from a method.
>
> ```
> $ ruby -e 'def m(v=12);v;end; p method(:m).to_proc.binding.eval("v")'
> Traceback (most recent call last):
>   2: from -e:1:in `<main>'
>   1: from -e:1:in `eval'
> -e:1:in `<empty iseq>': undefined local variable or method `v' for main:Object (NameError)
> ```

You are accessing a local variable in the method (which does not exist until the method is invoked), whereas I am accessing a variable outside the method, captured by the proc that defined the method, which does exist outside the invocation of the method. So your example does not explain the behavior that I am highlighting.

To try answer my own question, I can see how this might not be a bug if method.to_proc returns a proc that is no way related to the proc that defined the method. Such a proc simply forwards the args to the method and knows nothing about the original proc binding. I initially thought that it would somehow inherit the binding of the original proc.

#5 - 01/07/2018 07:43 AM - nobu (Nobuyoshi Nakada)

Once a Proc is bound as a method, the context of that method is not same as the based Proc, receiver for instance.

```
x = ->{}
o = Object.new
o.define_singleton_method(:x, x)
p x.binding.receiver #=> main
p o.method(:x).to_proc.binding.receiver #=> o
```

#6 - 01/07/2018 07:43 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected