

Ruby - Feature #8215

Support accessing Fiber-locals and backtraces for a Fiber

04/04/2013 08:47 AM - halorgium (Tim Carey-Smith)

Status:	Closed	
Priority:	Normal	
Assignee:	ioquatix (Samuel Williams)	
Target version:		
Description =begin As part of debugging celluloid, I have been wanting to diagnose where the Fibers are running and their various locals. I would expect the following to work. Thread.current[:key] = "outside" fiber = Fiber.new do Thread.current[:key] = "inside" Fiber.yield end fiber.resume fiber[:key] == "inside" # true fiber.backtrace # ... I also wonder whether (({Fiber#[]})) should be implemented, so (({Fiber.current[:key]})) is possible. For reference, here is the issue on the rubinius issue tracker: (<"github/rubinius/rubinius/2200" URL: https://github.com/rubinius/rubinius/issues/2200>)) =end		

History

#1 - 04/04/2013 09:50 PM - Eregon (Benoit Daloze)

- Category set to core
- Status changed from Open to Closed

Thread#[] and friends access *Fiber*-local variables, as the doc says:

(Attribute Reference---Returns the value of a fiber-local variable (current thread's root fiber if not explicitly inside a Fiber), using either a symbol or a string name.)

This is unfortunate, as indeed one might expect it to be *thread*-local, but it has been made fiber-local for safety.

In ruby >= 2.0.0, you have thread_variable_{get,set} and such for manipulating *thread*-local variables. Not as nice, but at least the API is there. See [#7097](#).

#2 - 04/05/2013 06:27 PM - nobu (Nobuyoshi Nakada)

- Description updated
- Status changed from Closed to Open

According to that rubinius issue tracker, it seems a feature request for Fiber#[] and Fiber#[]=.

#3 - 04/05/2013 08:06 PM - Eregon (Benoit Daloze)

Ah, sorry, I thought it was only misunderstanding of Thread#[] and such.

#4 - 04/07/2013 07:13 PM - halorgium (Tim Carey-Smith)

Yup, I am sorry if that was not clear!

I also am very interested in being able to get the backtrace of a Fiber too.

Should I open a new issue for Fiber#backtrace?

#5 - 04/09/2013 04:33 PM - nobu (Nobuyoshi Nakada)

One ticket, one issue, please.

#6 - 04/16/2013 05:03 PM - nobu (Nobuyoshi Nakada)

- File 0001-cont.c-fiber-local-accessors.patch added

More tests may be needed.

#7 - 04/19/2013 03:50 PM - halorgium (Tim Carey-Smith)

I realised that this might be better in common-ruby.
I originally proposed the idea with RBX.
Could someone move it?

#8 - 04/19/2013 04:23 PM - duerst (Martin Dürst)

On 2013/04/19 15:50, halorgium (Tim Carey-Smith) wrote:

Issue [#8215](#) has been updated by halorgium (Tim Carey-Smith).

I realised that this might be better in common-ruby.

This isn't issue specific: I propose that just for the moment, issues stay where they are. Once the overall directions are sorted out, we can organize a general campaign to move issues wherever necessary. If we can avoid it, we don't want to pollute each and every issue with individual move requests.

Regards, Martin.

#9 - 06/02/2013 03:45 PM - zzak (zzak _)

- Status changed from Open to Assigned
- Assignee set to nobu (Nobuyoshi Nakada)

#10 - 01/05/2020 10:51 PM - ioquatix (Samuel Williams)

[@matz \(Yukihiko Matsumoto\)](#) I agree with adding all three APIs, Fiber#[], Fiber#[]= and Fiber#backtrace. Can you let me know if you are happy with these additions?

#11 - 01/05/2020 10:51 PM - ioquatix (Samuel Williams)

- Assignee changed from nobu (Nobuyoshi Nakada) to ioquatix (Samuel Williams)

#12 - 01/06/2020 06:22 PM - Eregon (Benoit Daloze)

Fiber#[] and Fiber#[]= sounds fine, but what if somebody does:

```
some_fiber = Fiber.new { ... }; Thread.new { some_fiber[:fiber_local] }?
```

I think that should raise or not be possible.
If it would return the value, it would imply synchronization on every access to fiber locals which seems unfortunate.

By making the API Fiber[], we can avoid that entirely and have true fiber-locals, which can only be accessed by that Fiber:

```
Fiber[:my_fiber_local] = 42
value = Fiber[:my_fiber_local]
# no way to access fiber locals of another Fiber
```

I think for new APIs we should take the chance to make them only possible to use correctly.
We could even finally have Fiber and Thread local in a consistent way:

```
# access Fiber-local
Fiber[:my_fiber_local]
# access Thread-local
Thread[:my_thread_local]
```

#13 - 01/06/2020 08:50 PM - ioquatix (Samuel Williams)

[@Eregon \(Benoit Daloze\)](#) I agree with your points. I respect that you've studied a lot in your thesis so ultimately I'll defer to your judgement. But let me explain a bit more.

The reason for expanding the Fiber# interface is so that tools like async can show better debugging of all fibers.

Ideally it can show the backtrace, and allow the user to check fiber locals (and maybe even get a list of keys for debugging purposes).

I'd also be okay with adding Fiber.[] and so on, but that's kind of a separate issue.

Thread safety is not my concern, the function can be marked as thread unsafe, and maybe we can add detection of this and warn against it.

#14 - 04/19/2020 05:21 AM - ioquatix (Samuel Williams)

[@matz \(Yukihiro Matsumoto\)](#) do you mind checking this when you have time? Especially Fiber#backtrace is super important for debugging issues with fibers.

#15 - 04/21/2020 02:18 AM - ioquatix (Samuel Williams)

Here is a quick hack I used to add Fiber.backtrace to aid in debugging. I'd say, don't use it in production, but I did :p

```
module Fiber::Backtrace
  def yield
    Fiber.current.backtrace = caller
    super
  end
end

class Fiber
  attr_accessor :backtrace

  class << self
    prepend Fiber::Backtrace
  end
end
```

#16 - 04/24/2020 08:37 AM - Eregon (Benoit Daloze)

[@ioquatix \(Samuel Williams\)](#) Please create a separate feature ticket for Fiber#backtrace, as already asked by nobu in <https://bugs.ruby-lang.org/issues/8215#note-5>

I'd like to keep the discussion here about Fiber locals, not mix it with an unrelated method.

A separate ticket is better so e.g. the discussion is clear and focused, that ticket can be closed when implemented, the reference from NEWS will be much less confusing, etc.

Back to the fiber locals discussion:

Thread safety is not my concern, the function can be marked as thread unsafe, and maybe we can add detection of this and warn against it.

It's of course unacceptable if a Ruby implementation behaves unsafely (e.g., crashes or loses fiber local variable) for this, so it is a relevant concern for this issue.

Another concern is this needs Fiber.current, which is only available after require "fiber".

I think that could be confusing, so I'd suggest making Fiber.current always available if we go that way.

I think Fiber[:foo] and Fiber[:foo] = ... is a better API because it avoids this problem entirely, and gives the possibility to have clean Thread and Fiber locals APIs.

We could also add a check in Fiber#[] but that would read the current Fiber a second time, and move it to a runtime error instead of simply not being possible with Fiber[:foo].

Maybe the check is not so important since I guess Thread#[] will still need to work for a long while (but maybe we could deprecate cross-thread/cross-fiber access), but for new APIs I think a good safe design is worth considering.

If you think we need the ability to access another Fiber's locals, could you show some examples?

#17 - 04/24/2020 09:05 AM - Eregon (Benoit Daloze)

To make it clearer about "clean API": if we have Fiber#[] and Thread#[] then Thread#[] is just deceptive and doesn't do anything different. OTOH, with Fiber[:foo] and Thread[:foo] we can actually have the intuitive semantics.

#18 - 04/24/2020 10:23 AM - ioquatix (Samuel Williams)

It's of course unacceptable if a Ruby implementation behaves unsafely (e.g., crashes or loses fiber local variable) for this, so it is a relevant concern for this issue.

We already have situations in JRuby and TruffleRuby where unsynchronized access to shared mutable state can cause a crash or data corruption. So why is this different?

If you believe it's important, then I defer to your judgement. However, from my point of view:

- This method should not be invoked across threads.
- This method should not need to be thread safe.
- The GVL might make it safe but it shouldn't be guaranteed.
- Maybe we need annotations or other documentation to explain this.

If you think we need the ability to access another Fiber's locals, could you show some examples?

The reason to access other fiber locals is mostly for debugging purposes. In Async, we make a list of tasks:

```
4874m43s      info: Falcon::Controller::Proxy [oid=0x2b28b0f0d558] [pid=324886] [2020-04-24 09:55:55 +0000]
              | #<Async::Reactor:0x2b28b10316c8 1 children running>
              | #<Async::Task:0x2b28b1031498 (running)>
              | #<Async::Task:0x2b28b10312e0 waiting for signal USR1 (running)>
              | #<Async::Task:0x2b28b1030930 accepting secure connection #<Addrinfo: [::ff
ff:34.228.115.75]:26564 TCP> (running)>
              | #<Async::Task:0x2b28b1079b1c connected to #<Addrinfo: /srv/http/ww
w.oriontransfer.co.nz/application.ipc SOCK_STREAM> [fd=17] (complete)>
              | #<Async::Task:0x2b28b109a7b8 h2 reading data for Async::HT
TP::Protocol::HTTP2::Client. (running)>
              | #<Async::Task:0x2b28b10869e8 Reading h2 requests for Async::HTTP::
Protocol::HTTP2::Server. (complete)>
              | #<Async::Task:0x2b28b13a2dc0 connected to #<Addrinfo: /srv
/http/www.codeotaku.com/application.ipc SOCK_STREAM> [fd=16] (complete)>
              | #<Async::Task:0x2b28b12fb8f4 h2 reading data for A
sync::HTTP::Protocol::HTTP2::Client. (running)>
              | #<Async::Task:0x2b28b0fefe30 Reading h2 requests for Async::HTTP::
Protocol::HTTP2::Server. (running)>
              | #<Async::Task:0x2b28b1004b64 h2 reading data for Async::HT
TP::Protocol::HTTP2::Server. (running)>
```

When something is going wrong, the ability to dig into the fiber state is very useful. My plan is to make debugging tools to expose this more easily but I can't do it if there are no methods to access the state.

#19 - 04/24/2020 12:08 PM - Eregon (Benoit Daloze)

We already have situations in JRuby and TruffleRuby where unsynchronized access to shared mutable state can cause a crash or data corruption. So why is this different?

Let's not introduce more bugs (and IMHO those bugs should be fixed), especially if these methods are supposed to access *local* state, which means state you may not access from any other Thread.

It's probably fine to access locals of another Fiber of the same Thread though, that's not racy since Fibers of a Thread don't run simultaneously. So I'd be OK with that, if we raise when trying to access locals of a Fiber belonging to a different Thread.

Is there a reason you want to store this state in Fiber locals and not e.g., in instance variables of the Fiber object?

#20 - 08/08/2020 11:11 PM - ioquatix (Samuel Williams)

- Status changed from Assigned to Closed

It looks like we can implement `Fiber#backtrace`:

<https://bugs.ruby-lang.org/issues/16815>

and in addition we should continue the discussion about Fiber locals:

<https://bugs.ruby-lang.org/issues/13893>

Files

0001-cont.c-fiber-local-accessors.patch	2.94 KB	04/16/2013	nobu (Nobuyoshi Nakada)
---	---------	------------	-------------------------