Ruby - Feature #19333

Setting (Fiber Local|Thread Local|Fiber Storage) to nil should delete value in order to avoid memory leaks.

01/11/2023 10:21 PM - ioquatix (Samuel Williams)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

As it stands, Fiber Locals, Thread Locals and Fiber Storage have no way of deleting key-value associations.

```
100.times do |i|
  name = :"variable-#{i}"
  Thread.current[name] = 10
end
```

Because of this, dynamically generated associations can leak over time. This is worse for things like Threads that might be pooled (or maybe an argument against user-space pooling).

In any case, having a way to delete those associations would allow application code to at least delete the associations when they no longer make sense.

I propose that assigning nil to "locals" or "storage" should effectively delete them.

e.g.

```
100.times do |i|
  name = :"variable-#{i}"
  Thread.current[name] = 10
  Thread.current[name] = nil # delete association
end
```

A more invasive alternative would be to define new interfaces like Thread::Local, Fiber::Local and Fiber::Storage::Local (or something) which correctly clean up on GC.

History

#1 - 01/16/2023 09:53 AM - ioquatix (Samuel Williams)

While the proposed idea is a good move in the right direction, it doesn't handle cases where several (a large number) threads all have the SAME thread local, and that thread local goes out of scope, e.g.

```
name = ":local-#{object_id}"

100.times do
  Thread.new do
    thing = ... something that consumes memory ...
    Thread.current.thread_variable_set(name, thing)
    # Thread lives for a long time
    while job = jobs.pop
      job.call
    end
  end
end
```

Even when name is no longer useful for any purpose, all those threads sill still have that thread local set. There is no global way to say "For all threads that have a thread local set for name, please clear it".

In order to do that, we'd need to introduce a Thread::Local object which uniquely identifies an association in 0 or more threads, and when that Thread::Local instance is GCed, all threads which have it set, have it cleared. This ensures that threads don't leak memory. Consider a work pool that allocates a new Thread::Local every now and then... eventually threads will have lots of locals which are potentially no longer useful.

That being said, the other option is just not to encourage people to have long running threads that accumulate cruft over time.

#2 - 01/16/2023 11:53 AM - Eregon (Benoit Daloze)

The current solution/workaround to both of these memory leaks is to use concurrent-ruby's ThreadLocalVar (and upcoming FiberLocalVar), which is the reason that exists.

The idea to remove on assigning nil looks elegant at first sight but has multiple problems:

- The user might use nil as a "valid value" and with this change it will be more expensive to remove the fiber local than to just assign, also more expensive if later that fiber local is set again to non-nil (delete+insert vs set+set).
- This is problematic when implementing fiber locals with Shapes, which TruffleRuby currently does, because it causes a lot more Shape polymorphism if fiber locals can be removed, so makes fiber locals slower.
- It doesn't solve the leak itself, it would need a finalizer per fiber local, which is pretty heavy for this feature AND we can't put a finalizer on the Symbol since the Symbol is never collected. So this seems unusable for fiber/thread locals, it would need an extra wrapping object for every usage, or explicitly clearing the fiber local in a begin/ensure/end pattern.

For those reasons I'm rather against this.
I think it's a rare enough problem that it's OK to rely on concurrent-ruby for this feature.
Many fiber/thread local usages are fine because they don't use a random part in the Symbol and so it's a small number of them.
It is a problem when there are many fiber/thread locals, IMO those should use concurrent-ruby's ThreadLocalVar/FiberLocalVar.

> That being said, the other option is just not to encourage people to have long running threads that accumulate cruft over time.

That's a hard sell, at least the not having long running threads part. On all current Rubies the time to create a thread is significant and so reusing threads e.g. via a thread pool is a necessity for performance (e.g., in threaded web servers like Puma). (maybe CRuby can do M-N threading in the future, but this is not possible on JVM AFAIK because e.g. ReentrantLock relies on java.lang.Thread identity and not RubyThread/RubyFiber identity)

#3 - 01/17/2023 06:40 AM - ioquatix (Samuel Williams)

> The user might use nil as a "valid value"

Have you seen any examples of this?

#4 - 01/17/2023 11:41 AM - Eregon (Benoit Daloze)

There are some, yes, see
https://github.com/search?q=%22Thread.current%5B%22+%22%5D+%3D+nil%22+language%3ARuby&type=code&l=Ruby
A better search with a Regexp:
https://github.com/search?q=%2FThread.current%5C%5B.*%5C%5D+%3D+nil%2F+language%3ARuby&type=code&l=Ruby
Some are about cleaning after usage in ensure, but not all.
And even then it's not clear if it's never used again after.
For example I saw a = nil to reset a connection.

#5 - 01/17/2023 10:13 PM - ioquatix (Samuel Williams)

Thanks for the code search link.

Regarding the usage, wouldn't it only matter if the variable was checked with thread_variable? or equivalent? Because thread_variable_get(:x) is nil regardless of whether it's set (to nil) or not (deleted on nil assignment). The same applies to Thread.current[] IIUC.

Assignment to nil, e.g. thread_variable_set(:x) = nil, and the subsequent deletion of that variable, only has a side effect iif later behaviour is controlled by thread_variable?(:x) AND that behaviour would be different in the presence of a nil value.

Also, thread_variable? was actually pretty broken until recently, so I would be surprised if anyone was using it in practice:
https://bugs.ruby-lang.org/issues/16906

#6 - 01/18/2023 03:58 PM - Eregon (Benoit Daloze)

There is also Thread#key? for Fiber locals, so not just thread_variable?.

Above I am talking more about the performance issue than the semantic incompatibility.
Doing delete+insert is more expensive than set+set, and even much more so if Fiber/Thread locals are implemented with Shape (which is already the case of TruffleRuby, so not theoretical).

#7 - 01/19/2023 08:58 AM - matz (Yukihiro Matsumoto)

Accepted, although we foresee some possible issues. One example is that people may use nil as a value when they only care about existence of keys (set-like usage). They should use non-nil values (e.g., true) for the case.

Matz.

#8 - 12/28/2023 08:55 AM - ioquatix (Samuel Williams)

This is implemented for Fiber#storage in Ruby 3.3.

I'll consider expanding it to Thread and Fiber locals as it was agreed on but has a bigger blast radius.

#9 - 08/07/2025 12:30 AM - ioquatix (Samuel Williams)

See https://github.com/socketry/async-http-faraday/pull/57 for a valid use case.