Ruby master - Feature #16792

Make Mutex held per Fiber instead of per Thread

04/16/2020 09:01 AM - Eregon (Benoit Daloze)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

Currently, Mutex in CRuby is held per Thread.
In JRuby and TruffleRuby, Mutex is held per Fiber (because it's simply easier implementation-wise).

While a user could theoretically notice the difference, it seems extremely uncommon in practice (probably incorrect synchronization).

The usage pattern for a Mutex is using #synchronize or lock+unlock.
Such a pattern protects/surrounds a region of code using some resource, and such a region of code is always on the same Fiber since it's on a given Ruby "stack".

With #16786 it becomes more relevant to have Mutex held per Fiber, otherwise Mutex#lock will hurt scalability of that proposal significantly.
This means, if a Fiber does Mutex#lock and it's already held by another Fiber of the same Thread, and the Thread#scheduler is enabled, instead of just raising an error (which made sense before, because it would be a deadlock, but no longer the case with scheduler),
or disabling fiber scheduling entirely until #unlock (current state in #16786, makes Mutex#lock special and hurts scalability),
we would just go to the scheduler and schedule another Fiber (for instance, the one holding that Mutex, or any other ready to be run Fiber).

This is not a new idea and in fact Crystal already does this with its non-blocking Fibers, which is very similar with #16786:
https://github.com/crystal-lang/crystal/blob/612825a53c831ce7d17368c8211342b199ca02ff/src/mutex.cr#L72

Mutex#lock is just like other blocking operations, so let's make it so building on #16786.
I believe it's the natural and intuitive thing to do for Fiber concurrency with a scheduler.

Queue#pop and SizedQueue#push could be other candidates to handle in a similar way.

Here is an early commit to make Mutex held per Fiber, it's quite trivial as you can see:
https://github.com/ruby/ruby/compare/master...eregon:mutex-per-fiber
It passes test-all and test-spec.

Related issues:

Related to Ruby master - Feature #16786: Light-weight scheduler for improved ...	**Closed**

Associated revisions

Revision 178c1b09 - 09/14/2020 04:44 AM - Eregon (Benoit Daloze)

Make Mutex per-Fiber instead of per-Thread

- Enables Mutex to be used as synchronization between multiple Fibers of the same Thread.
- With a Fiber scheduler we can yield to another Fiber on contended Mutex#lock instead of blocking the entire thread.
- This also makes the behavior of Mutex consistent across CRuby, JRuby and TruffleRuby.
- [Feature #16792]

Revision ce888bfa - 09/17/2020 09:17 AM - Eregon (Benoit Daloze)

Add NEWS entry for [Feature #16792]

History

#1 - 04/16/2020 09:02 AM - Eregon (Benoit Daloze)

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN)

- Tracker changed from Bug to Feature

#2 - 04/16/2020 09:02 AM - Eregon (Benoit Daloze)

- Related to Feature #16786: Light-weight scheduler for improved concurrency. added

#3 - 04/16/2020 09:18 AM - Eregon (Benoit Daloze)

- Description updated

#4 - 04/16/2020 09:19 AM - Eregon (Benoit Daloze)

- Description updated

#5 - 04/16/2020 09:24 AM - ioquatix (Samuel Williams)

Characterising this as trivial hides the impact of this both on the scheduler design and application code which expects per-thread mutex.

Once the scheduler lands into master, we will need to scope out something like wait_mutex and all the associated scheduling that needs to happen.

Can you check the Crystal implementation? How do they implement fair scheduling?

Finally, as this breaks assumptions about application code, I proposed Mutex.new(blocking: false/true). I'm on the fence regarding this interface, but at least it should be clear that such a change has a track record of breaking application code. So if we decide to make the mutex per-fibre, we need to anticipate this problem, either through analysis of existing source code, issuing warnings, or something else.

Ultimately, I think keeping it simple would be great. But I'm hesitant to do so since it may break existing code. headius (Charles Nutter) maybe you can comment on what user code was affected so we can see if there is some other way to mitigate it.

#6 - 04/16/2020 09:32 AM - Eregon (Benoit Daloze)

> Application code which expects per-thread mutex

I'm happy to see examples of that.
I've never seen it in 5+ more years of bug reports on TruffleRuby, and JRuby AFAIK always had Mutex per Fiber and yet no real issue because of it.
The only cases that would rely on this seems broken synchronization
(e.g., taking the lock in one Fiber and unlocking in another Fiber, which I believe is always a bug, as lock/unlock needs to be used like lock; begin; ...; ensure; unlock; end, which is always on the same stack so on the same Fiber).

#7 - 04/16/2020 09:37 AM - Eregon (Benoit Daloze)

ioquatix (Samuel Williams) wrote in #note-5:

> Can you check the Crystal implementation? How do they implement fair scheduling?

It's linked in the description.
Mutex is already not fair so I don't think it matters.

> Finally, as this breaks assumptions about user code, I proposed Mutex.new(blocking: false/true).

It doesn't.

> I'm on the fence regarding this interface, but at least it should be clear that such a change has a track record of breaking application code.

Any example? Please don't claim such things if there is no example.

#8 - 04/16/2020 09:57 AM - ioquatix (Samuel Williams)

> Any example? Please don't claim such things if there is no example.

I'll have to defer to headius (Charles Nutter) for examples, since he was the one that mentioned it. I can only assume he is telling the truth when he said some application code was broken.

The only reason to look at it is to check if it's a valid use case or not. If we can't find any valid use case, it confirms your assessment, which is a good thing.

> It doesn't.

I did play around with this a bit, and I also wondered if we should add it to Ruby spec before making changes.

https://github.com/ruby/ruby/pull/3032/files#diff-1a99f5621b805d95b67228708b652ff9R25

That is the current semantic and if we introduce non-blocking fiber, it can cause deadlock, as shown in the test, if you disable the mutex -> blocking relationship.

#9 - 04/16/2020 10:26 AM - ioquatix (Samuel Williams)

My initial reply might have come across as overly critical and that was not my intention.

I want to say, I agree with this proposal, and I think it's a good idea.

I would like to see that we merge the scheduler proposal first, which preserves the current semantics of Mutex. Then, shortly afterwards and before Ruby 3 is released, we should implement this.

One more point:

> will hurt scalability of that proposal significantly.

There is no evidence to suggest this. I do agree it *can* hurt the scalability of a system but it's entirely possible to build such a system without a Mutex.

#10 - 04/21/2020 03:10 AM - ioquatix (Samuel Williams)

In order to experiment with this, I'd like to propose the following hooks to the scheduler:

```
class Scheduler
  # A fiber has tried to lock a mutex, but it failed.
  def wait_mutex(mutex)
  end

  # A mutex which has previously called `wait_mutex` may now be available to lock.
  def notify_mutex(mutex)
  end
end
```

Eregon (Benoit Daloze) do you think this is sufficient? I'm a little bit concerned about the reentrancy guarantees of notify_mutex. Should we define that method to be reentrant/thread safe? Ideally, we have a way to notify the scheduler to reschedule the fiber to try and lock the mutex again.

cc jeremyevans0 (Jeremy Evans)

#11 - 05/12/2020 10:21 PM - ko1 (Koichi Sasada)

I think it seems difficult to implement it.
If an interpreter manages everything, it is easy (at least I can image how to implement it).

(1) API

I'm not sure we can implement Mutex scheduling with the hooks introduced at #10.
If there is only one thread, maybe it is easy to implement.

Maybe notify_mutex(mutex) will be called with locked mutex by the interpreter. It means we need to introduce lock_nonblock or similar API for Mutex (maybe _nonblock is not good name because it is different from IO#read_nonblock). Mutex#notice_when_locked?

Also we need to wait this notification with wait for the ready of IO operations. how to write it? Use pipe trick or interrupt mechanism?

(2) Queue/SizedQueue/CV

there are several blocking operations because of the thread synchronization, how to treat them?
Introduce unified hook method like wait_synchronization(sync_object) and notify_synchronization(sync_object)?
Introduce pop_nonblock similar to Mutex#sync_nonblock?

(3) implemented by all schedulers?

maybe most of code are same between scheduler implementations. can we provide a framework to implement it?

#12 - 08/17/2020 03:17 AM - ioquatix (Samuel Williams)

I have played around with this and I see the most basic operation is a way to tell the scheduler that a fiber can make progress. We can use a generic approach - in a loop, you may call Fiber.yield. When that fiber is ready to proceed (e.g. mutex is unlocked), you need to notify scheduler, e.g. either of the following interfaces:

```
class Scheduler

# Inform the scheduler that the fiber can be resumed. If urgent, the scheduler will wake up as soon as possibl
e.
```

```
  # @parameter fiber [Object] Must respond to `#resume`.
  def ready(fiber, urgent = false)
  end

  # Execute the block in the run loop. If urgent, the scheduler will wake up as soon as possible.
  def invoke(urgent = false, &block)
  end
end
```

Here is the proof of concept:

https://github.com/socketry/async/pull/72

See the hacks required to async/mutex.rb to make it work. This approach should work for Queue/SizedQueue/ConditionVariable.

The implementation of a blocking operation looks like this:

```
until [operation succeeded]
  self.waiting << Fiber.current
  Fiber.yield
end
```

On the other side, where the resource becomes available:

```
if fiber = self.waiting.pop
  if fiber.scheduler # helper
    fiber.scheduler.ready(fiber)
  else
    # existing logic
  end
end
```

I don't mind how we call the methods. Crystal had reschedule. I'm open to ideas that make sense. ready maybe not so great.

Also, maybe this is crazy idea, but in order to unify the interface, maybe we should introduce Fiber#call as an alias for Fiber#resume (or Fiber#transfer). That way, we could make it valid to substitute a proc for a fiber.

#13 - 09/14/2020 04:44 AM - Eregon (Benoit Daloze)

- Status changed from Open to Closed

Applied in changeset git|178c1b0922dc727897d81d7cfe9c97d5ffa97fd9.

Make Mutex per-Fiber instead of per-Thread

- Enables Mutex to be used as synchronization between multiple Fibers of the same Thread.
- With a Fiber scheduler we can yield to another Fiber on contended Mutex#lock instead of blocking the entire thread.
- This also makes the behavior of Mutex consistent across CRuby, JRuby and TruffleRuby.
- [Feature #16792]