In some cases, it may be beneficial to disable preemptibility to have more predictable behavior than increasing Thread#priority. Threads with preempt disabled will run until they either:

a) encounter a thread blocking region
b) call Thread.pass

This allows native threads to act cooperatively without being interrupted by other threads.

Thread#preemptible? => true or false
Thread#preemptible = (true|false)

I plan to implement timer-based switching to "auto-fiber/threadlet/thriber/whatever-name" [Feature #13618] to minimize migration costs from Thread.

However, I think based on the discussion in [Feature #13618]; having predictability of non-preemptible threads is beneficial. So implement it for native Thread for now.

I will unify the behavior of [Feature #13618] with existing Thread. I think green/native threads can be unified under Thread class similar to how Fixnum/Bignum are both "Integer".

No, this is different from that old method because thread switching still happens from blocking regions.

Is this the same as Thread.new { Thread.handle_interrupt(Exception => :never) { ... } }?
What's the difference?

Is preemption as in threads giving control (and the GIL) to another still possible with Thread#preemptible = true or not? If not, then I think this feature requires a GIL to be implemented (because Thread#preemptible = true is called after Thread creation) and I am very strongly against it.
Is this the same as Thread.new { Thread.handle_interrupt(Exception => :never) { ... } }?

What's the difference?

No, current thread switching does not use interrupts in the same sense (no Exception objects are created). However, I suppose it could be implemented internally using ec->interrupt_mask.

Is preemption as in threads giving control (and the GIL) to another is still possible with Thread#preemptible = true or not? If not, then I think this feature requires a GIL to be implemented (because Thread#preemptible = true is called after Thread creation) and I am very strongly against it.

Giving control to other threads still happens with Thread.pass or anything which currently releases GVL (including IO#read, File.open, etc...).

For platforms without GVL, it can be a no-op. I understand why this can be a bad feature from that perspective (I hate GVL, too). I mainly wanted this feature to give equivalence for proposed auto-Fiber [Feature #13618] behavior.

Also, maybe disabling preempt less important since I am redoing [Misc #14937] to eliminate timer-thread completely for pthreads platforms; so there won't be a need to spawn extra threads even under contention.

#6 - 07/28/2018 06:35 PM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

Giving control to other threads still happens with Thread.pass or anything which currently releases GVL (including IO#read, File.open, etc...).

At that point I wouldn't call them threads anymore. I think threads usually imply preemption (not just on blocking actions but also timer-based).

Also, we'd be mixing two very different kinds of Threads, preemptive threads and auto-fibers.

For platforms without GVL, it can be a no-op. I understand why this can be a bad feature from that perspective (I hate GVL, too). I mainly wanted this feature to give equivalence for proposed auto-Fiber [Feature #13618] behavior.

No, because if it's a no-op it has different semantics. Other Ruby implementations without GVL will want to support something like auto-fibers too, and have compatible API. Designing this way seems to make it a MRI/GVL-only feature which is wrong.

However, I think making preemption an argument of the constructor, or a special Thread class/factory method would help implementing it on Ruby implementations without a GVL.

Still, it's unclear to me how multiple "non-preemptive threads" would work (when there is no GVL). I think auto-fibers need to be at a different level than Thread, so we can reason about such questions.

Being at Fiber level clarifies this: all fibers of a Thread never execute in parallel, but fibers of different Threads can.

I think it wouldn't make any sense in a Ruby implementation without GVL to have "a running non-preemptive thread/auto-fiber" prevent all other (preemptive) Threads to execute concurrently.

#7 - 08/08/2018 05:39 AM - ko1 (Koichi Sasada)

The following is the fact I want to clear

(fix me if my understanding is wrong)

I believe this proposal is equals to auto-fiber from user perspective except "how to create" and implementation.

The created concurrent entity (CE) is same:
CE does not support preemption by timer.
CE only switches other CE when explicit action (Thread.pass) or implicit blocking operations.

The CE is not same:

- implementation
  - Using native-thread (OS resource) (by preemptible=false)
  - Using only memory (by auto-fiber).
- creation.
  - we can switch preemptible by preemptible= method.
  - auto-fiber should be created from special constructor(s)

**This is my opinion**

I don't think it is good idea to support Thread#preemptible=false because someone can misuse this feature as "locking" like Thread.exclusive on Ruby 1.8 (this method prevented thread switching. From 1.9, this method changed meaning).

I agree with Eregon:

```
However, I think making preemption an argument of the constructor, or a special Thread class/factory method would help implementing it on Ruby implementations without a GVL.
```

and this is what auto-fiber do. I don't against to introduce auto-fiber if the name is related to Thread (not related to Fiber. The implementation is based on Fiber, but the semantics is not Fiber. They should not have #resume method and so on).

---

#8 - 08/08/2018 08:04 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

```
I don't think it is good idea to supportThread#preemptible=false because someone can misuse this feature as "locking" like Thread.exclusive on Ruby 1.8 (this method prevented thread switching. From 1.9, this method changed meaning).
```

Fair enough. We can close this issue

```
and this is what auto-fiber do. I don't against to introduceauto-fiber if the name is related to Thread (not related toFiber. The implementation is based on Fiber, but the semanticsis not Fiber. They should not have #resume method and soon).
```

OK, I will call [Feature #13618] "Thread::Green" or something similar. I should be done with it once timer-thread-elimination is fixed and Timeout-in-VM is done (so maybe one more week or two).

rb_thread_sleep_* for green threads requires sharing data structures with Timeout.

#9 - 08/09/2018 07:10 AM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

---

**Files**

| File Name                                      | Size   | Date       | Username | Email
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0001-thread-allow-disabling-preempt.patch</td>
<td>4.16 KB</td>
<td>04/27/2018</td>
<td>normalperson</td>
<td>(Eric Wong)</td>
</tr>
</tbody>
</table>