Fix collision of ConditionVariable#wait timeout and #signal (+ other cosmetic changes)

03/19/2012 12:40 AM - funny_falcon (Yura Sokolov)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>kosaki (Motohiro KOSAKI)</td>
</tr>
<tr>
<td>Target version:</td>
<td>2.0.0</td>
</tr>
<tr>
<td>ruby -v:</td>
<td>ruby 2.0.0dev (2012-03-17 trunk 35075) [i686-linux]</td>
</tr>
</tbody>
</table>

**Description**

1. Currently, when Thread wakes on timeout, it could not remove itself from ConditionVariable waiters until it acquires lock. So that, when ConditionVariable#signal is called, it will try to wakeup such thread, instead of some one else.
   
   [1](https://github.com/funny-falcon/ruby/commit/24a9b6112477b2107ac9a19d0189a11fb97aa891)
   [2](https://github.com/funny-falcon/ruby/commit/24a9b6112477b2107ac9a19d0189a11fb97aa891.patch)

   Simple way to avoid it, is to allow Mutex#sleep to recieve a block, which will be called right after Thread will awake, but before Mutex will be tried to lock.
   
   [1](https://github.com/funny-falcon/ruby/commit/9e9157c5318926331dbee883416b69d38a58f0a5d)
   [2](https://github.com/funny-falcon/ruby/commit/9e9157c5318926331dbee883416b69d38a58f0a5d.patch)

2. Since MatzRuby use GVL for thread isolation, and native method could not be interrupted (unless it will), we could remove couple of calles to Mutex#synchronize.
   
   [1](https://github.com/funny-falcon/ruby/commit/9e9157c5318926331dbee883416b69d38a58f0a5d)
   [2](https://github.com/funny-falcon/ruby/commit/9e9157c5318926331dbee883416b69d38a58f0a5d.patch)

3. Usage of hash with compare_by_identity allows remove call to Array#include? in a Queue. Also it allows to remove other call Mutex#synchronize from ConditionVariable#wait in case when we rely on GVL.
   
   [1](https://github.com/funny-falcon/ruby/commit/0da1887a04f7a0e4f9289d2167c2a60073651e1)
   [2](https://github.com/funny-falcon/ruby/commit/0da1887a04f7a0e4f9289d2167c2a60073651e1.patch)

3. And cosmetic changes to SizedQueue
   
   [1](https://github.com/funny-falcon/ruby/commit/60ed97557c8178bc78edf670f3d53d661e627bf0)
   [2](https://github.com/funny-falcon/ruby/commit/60ed97557c8178bc78edf670f3d53d661e627bf0.patch)

Pull request at once:

[1](https://github.com/ruby/ruby/pull/104)
[2](https://github.com/ruby/ruby/pull/104.diff)
[3](https://github.com/ruby/ruby/pull/104.patch)

**Related issues:**

Related to Ruby master - Feature #6762: Control interrupt timing

Closed 07/21/2012

**History**

**#1 - 03/19/2012 12:41 AM - funny_falcon (Yura Sokolov)**

and native method could not be interrupted (unless it will)

Sorry, it should be read as "(unless it wish)"

**#2 - 04/10/2012 02:07 PM - mame (Yusuke Endoh)**

- Status changed from Open to Assigned
- Assignee set to ko1 (Koichi Sasada)

**#3 - 04/11/2012 03:30 AM - kosaki (Motohiro KOSAKI)**

Hi Yura,
If nobody reviews months, I'll review this. The description is completely sane. :)

#4 - 06/26/2012 04:33 AM - ko1 (Koichi Sasada)
- Assignee changed from ko1 (Koichi Sasada) to kosaki (Motohiro KOSAKI)

Thanks Kosaki-san!

#5 - 09/14/2012 03:36 AM - kosaki (Motohiro KOSAKI)
- Status changed from Assigned to Rejected

I've reviewed your code. Unfortunately your testcase is not correct.

- def test_condvar_wait_timeout
- serialize = Serializer.new +
- mutex = Mutex.new
- condvar = ConditionVariable.new
- def condvar.waiters
- @waiters
- end
- thread = Thread.new do
- serialize.signal
- mutex.synchronize do
- condvar.wait(mutex, 0.001)
- end
- end
- serialize.wait{ thread.run }
- mutex.synchronize do
- sleep(0.01)
- assert_not_includes(condvar.waiters, thread)
- end
- end

this code behave as following.

1. main thread wait at serialize.wait
2. sub thread wake main thread up by serialize signal
3. sub thread wait at condvar.wait(mutex, 0.001)
4. main thread wake sub thread up by thread.run
5. main thread take 'mutex'
6. main thread sleep 0.01 seconds. but sub thread can't wake up because it can't take a 'mutex'.
7. main thread inspect condvar.waiters and it found sub thread.

But this is not a bug. main thread shouldn't take a mutex.

Moreover, your callback approach have a race. Think, if trap is happen when waiting in mutex.sleep.
your callback don't have any guarantee to be called.

I'm sorry. we can't take this patch.

#6 - 09/14/2012 04:48 PM - funny_falcon (Yura Sokolov)

- serialize.wait{ thread.run }

This could be

- serialize.wait[ ]

Test is failing as well, so that, step 4 doesn't matter.
In fact, step 4 runs before step 2 in every run, you could easily check it by putting "puts".
so that thread.run wakes up sub thread at the beginning, not at the condvar.wait !!!!

Then steps are looks like:

1. main thread wait at serialize.wait
2. sub thread wake main thread up by serialize signal
3. sub thread wait at condvar.wait(mutex, 0.001)
4. main thread take 'mutex'
5. main thread sleep 0.01 seconds (imitating some work).
6. sub thread tries to wake up, so that it should not be present in waiters. But he can't wake up because it can't take a 'mutex'.

09/23/2021
7. main thread inspect condvar.waiters and it found sub thread.

I can't understand, why "main thread shouldn't take a mutex"? In real application it could be very other thread, not main.
Holding mutex by main thread were intentionally to expose this race. Race could happen without it, but it would be hard to catch.

I see, that callback should be wrapped in ensure inside of mutex.sleep. Thank you for this point, I'll try to fix it.

#7 - 09/14/2012 06:01 PM - funny_falcon (Yura Sokolov)
Good day, Kosaki-san.

Ok, now callback is wrapped to rb_ensure inside of rb_mutex_sleep_forever and rb_mutex_wait_for.
Also, I replace serialize.wait{ thread.run } to serialize.wait inside of test to be less confusing.

Would you mind recheck patch and make consideration again, please?
(it is at the same place: 
https://github.com/ruby/ruby/pull/104
https://github.com/ruby/ruby/pull/104.patch
)

With regards,
Sokolov Yura aka funny-falcon.

#8 - 09/15/2012 03:13 AM - kosaki (Motohiro KOSAKI)

I can't understand, why "main thread shouldn't take a mutex"? In real application it could be very other thread, not main.
Holding mutex by main thread were intentionally to expose this race. Race could happen without it, but it would be hard to catch.

While the mutex is held by any thread, condvar.wait can't be return. It's one of basic condvar semantics. and sleep(0.01) don't release mutex at all. Thus, sleep(0.01) of main thread can't help to wakeup and finish sub thread.

#9 - 09/15/2012 03:21 AM - kosaki (Motohiro KOSAKI)

Ok, now callback is wrapped to rb_ensure inside of rb_mutex_sleep_forever and rb_mutex_wait_for.

Unfortunately, this doesn't help us. Even if using rb_ensure, still your callback can be interruptible. and we have no guarantee to execute a callback.
I believe we need to implement [Feature #6762].