Ruby master - Bug #11178

possible bug: a thread killed when waiting on ConditionVariable locks the mutex again before real exit

05/24/2015 09:35 PM - cvss (Kirill Vechera)

Status: Closed
Priority: Normal
Assignee:
Target version:
ruby -v:
Backport: 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

Description
Probably it's not a bug, but just a hidden behaviour. Please write your comments.

When a thread waits on ConditionVariable, it releases the mutex. And killing the thread during this state assumes no need to lock the mutex again because there's no ruby code to be executed. It results to problems:

- the killed thread remains alive until the mutex is released by another thread
- joining the killed thread from another thread locked on the mutex brings to deadlock

```ruby
@m = Mutex.new
@r = ConditionVariable.new

t = Thread.new {
  @m.synchronize do
    @r.wait(@m)
    # some ruby code here
  end
}
sleep 0.01

@m.synchronize do
  t.kill
  sleep 0.01
  puts "alive?: #{t.alive?}" # the thread should not be alive here
  t.join # at this point we're getting deadlock
end
```

History

#1 - 07/07/2019 05:09 AM - jeremyevans0 (Jeremy Evans)
- Status changed from Open to Closed

I don't think this is a bug. Killing a thread requires running ensure blocks until the thread exits, the ensure block for ConditionVariable#wait needs to reacquire the mutex. If it didn't, the method would exit with the mutex not locked, which would cause a ThreadError: Attempt to unlock a mutex which is not locked in Mutex#synchronize. Even if we added code to both ConditionVariable#wait and Mutex#synchronize to recognize killed threads and try to avoid this issue, it wouldn't result in correct behavior for manual ensure blocks. Consider:

```ruby
t = Thread.new {
  @m.synchronize do
    begin
      @r.wait(@m)
      ensure
        some_operation_assuming_mutex_is_locked
      end
    end
  end
}
```

This issue is not specific to ConditionVariable, it can happen anytime a thread tries to lock a mutex in an ensure block. For example, replacing the definition of t in your example with the following code will also trigger it:

```ruby
t = Thread.new {
  begin
    sleep 1
  end
```
def synchronize
  @m.synchronize do
    end
  end

Files
1.rb  240 Bytes  05/24/2015  cvss (Kirill Vechera)