# Ruby - Bug #9754

## Ruby refuses to run source with Mutex locks though there is no hazard..

04/17/2014 10:48 AM - ThomasWLynch (Thomas Lynch)

| | | | |
|---|---|---|---|
| **Status:** | Closed | | |
| **Priority:** | Normal | | |
| **Assignee:** | | | |
| **Target version:** | | | |
| **ruby -v:** | ruby 2.0.0p353 (2013-11-22 revision 43784) [x86_64-linux] | **Backport:** | 2.0.0: UNKNOWN, 2.1: UNKNOWN |

**Description**

I put this up on a forum for a week in case there was something I didn't know about Ruby Mutexes ..
but no one commented. So apologies in advance if some discussion is needed.

In this code, the locks are used as latches, so there is no hazard. There may be a hazard if they toggled, though locking and unlocking in itself is not a hazard. This latch form is useful for multiple threads performing separate searches while blocking processing until something is found. Any thread can flip the go latch.

(The second mutex is not needed but I didn't want to get into a discussion about that, so I added it to keep things clean. ... any thread could call .unlock and let the latch open, a second coming and doing the same just reconfirms the unlock.)

> ./mutex_brokenq.rb
> two threads are searching ..
> internal:prelude:8:in lock': deadlock; recursive locking (ThreadError) from <internal:prelude>:8:in synchronize'
> from ./mutex_brokenq.rb:33:in `'

---

here is the code:
#!/bin/ruby

puts "two threads are searching .."

```
wait_for_finish = Mutex.new
wait_for_finish.lock

wait_for_finish_mutex = Mutex.new

thread_one = Thread.new {
  puts "thread_one working!"
  count=0
  while( count < 10000)
    count += 1
  end
  wait_for_finish_mutex.synchronize do
    if wait_for_finish.locked? then wait_for_finish.unlock end
  end
}

thread_two = Thread.new {
  puts "thread_two working!"
  count=0
  while( count < 9000)
    count += 1
  end
  wait_for_finish_mutex.synchronize do
    if wait_for_finish.locked? then wait_for_finish.unlock end
  end
}
```

wait_for_finish.synchronize do

```
puts "Eureka!"
end
```

### History

#### #1 - 04/17/2014 07:12 PM - vjoel (Joel VanderWerf)

A Mutex is not reentrant.

This deadlocks:

```
m = Mutex.new
m.lock
m.synchronize {}
```

So does this:

```
m = Mutex.new
m.synchronize {m.synchronize {}}
```

But Monitor is reentrant, so the following does not deadlock:

```
m = Monitor.new
m.synchronize {m.synchronize {}}
```

There is no Monitor#lock, but there is MonitorMixin::ConditionVariable, which will do what you want.

(Btw, ruby-talk might be better for these questions than the forum.)

#### #2 - 07/11/2019 10:22 PM - jeremyevans0 (Jeremy Evans)

*- Status changed from Open to Closed*