Ruby master - Bug #6822
Race Condition with Fiber and Process
08/02/2012 10:38 AM - MartinBosslet (Martin Bosslet)

Status:  Closed
Priority:  Normal
Assignee:  ko1 (Koichi Sasada)
Target version:  2.0.0
ruby -v:  ruby 2.0.0dev (2012-05-07 trunk 35550) [x86_64-linux]

Backport:

Description

If I run the following code

```ruby
$stdout.sync = true
objects = [1, 2, 3]

fiber = Fiber.new do
  loop do
    objects.each { |obj| Fiber.yield(obj) }
  end
end

def run(obj)
  fork do
    puts obj
  end
end

def on_child_exit(obj)
  begin
    while Process.wait(-1, Process::WNOHANG)
      run(obj)
    end
    rescue Errno::ECHILD
  end
end

trap{:CHLD} { on_child_exit(fiber.resume) }
4.times { run(fiber.resume) }
sleep

I get

fiber_process.rb:26:in `resume': double resume (FiberError)

or

fiber_process.rb:26:in `resume': fiber called across stack rewinding barrier (FiberError)
```

There is a race condition when two or more children exit. Now I know I can implement this differently, but this still made me curious. Is this a bug? Let's say I would need to use a Fiber, then there is no way how I can do the synchronization manually, or is there? Using a Mutex to synchronize the Fiber#resume will fail due to the non-reentrant behaviour of Mutex#lock ("I'll get "in 'lock': deadlock; recursive locking (ThreadError")"). Is there a way to do this or should Fibers not be used in this context?

History

#1 - 08/02/2012 02:02 PM - shyouhei (Shyouhei Urabe)
- Category changed from core to YARV

08/07/2021 1/3
In general, you can sync with variables because Fibers are not changed automatically. In other words, you can completely control Fiber transition.

Thanks for looking into this. With your input, I found a way to safely synchronize the exiting childs by using Mutex#try_lock. Thank you!

No. You don't need Mutex at all. 
You only need to use variables (such as global variables).

Now I'm confused. How would I write the example code without getting the FiberErrors? Since I have no control over when a child process exits, I can't control when the 'trap(:CHLD)' block calls Fiber#resume, no? I thought I would have to do some form of manual synchronization there, to avoid the race condition. Sorry to bug you :)

Now, I understand your issue. This is not a Fiber problem, but concurrency problem with signal.

I recommend that you shouldn't use Fiber.resume in a trap handler. In the trap handler, you should only set a flag and make flag sense in main.

--
// SASADA Koichi at dot dot net
ko1 (Koichi Sasada) wrote:

Now, I understand your issue. This is not a Fiber problem, but concurrency problem with signal.

I recommend that you shouldn't use Fiber.resume in a trap handler. In the trap handler, you should only set a flag and make flag sense in main.

Thanks for the advice, I will do that! Thanks for bearing with me ;)}