Ruby trunk - Bug #15262

WeakRef::RefError for object that is still in use

10/27/2018 05:12 PM - larskanis (Lars Kanis)

Status:	Feedback		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.6.0dev (2018-10-27 trunk 65390) [x86_64-linux]	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

Given the following program:

```
require "weakref"

Thread.abort_on_exception = true

class Adder
  def self.start_adder(obj, oid)
    obj.add
  end

  def initialize
    @qu = Queue.new
    count = 10
    count.times do
      Thread.new(WeakRef.new(self), object_id, &self.class.method(:start_adder))
    end
    count.times do
      @qu.pop
    end
  end

  def add
    @qu.push true
  end
end

def test_adder
  10.times.map do
    Thread.new do
      Adder.new
    end
  end.each(&:join)
end

1000.times do
  test_adder
end
```

Expected behaviour:

The program should simply execute without error. This is the case on JRuby but not on MRI.

Actual behavior:

The program stops with a probability of approximately 50% with the following error:

```
$ ruby -W2 adder-test.rb
#<Thread:0x0000556e9e06f3f8@adder-test2.rb:6 run> terminated with exception (report_on_exception i
s true):
```

```
Traceback (most recent call last):
adder-test2.rb:7:in `start_adder': Invalid Reference - probably recycled (WeakRef::RefError)
```

Although start_adder works with a WeakRef, the Adder object should still be GC marked and therefore kept usable per WeakRef until Adder.new exited. This is because Adder.new waits for completion of all Threads to have called start_adder.

If Adder.start_adder is changed to catch the WeakRef error like so:

```
  def self.start_adder(obj, oid)
    obj.add
  rescue WeakRef::RefError
  end
```

... then the ruby VM detects a deadlock, which shows that @qu.pop is still executed within initialize. Therefore WeakRef#add should not raise a WeakRef::RefError to that point in time, but allow access to the object:

```
Traceback (most recent call last):
        5: from adder-test2.rb:35:in `<main>'
        4: from adder-test2.rb:35:in `times'
        3: from adder-test2.rb:36:in `block in <main>'
        2: from adder-test2.rb:32:in `test_adder'
        1: from adder-test2.rb:32:in `each'
adder-test2.rb:32:in `join': No live threads left. Deadlock? (fatal)
2 threads, 2 sleeps current:0x000056520aa3cee0 main thread:0x000056520a5f2ff0
* #<Thread:0x000056520a644b48 sleep_forever>
   rb_thread_t:0x000056520a5f2ff0 native:0x00007f7ccf535740 int:0
   adder-test2.rb:32:in `join'
   adder-test2.rb:32:in `each'
   adder-test2.rb:32:in `test_adder'
   adder-test2.rb:36:in `block in <main>'
   adder-test2.rb:35:in `times'
   adder-test2.rb:35:in `<main>'
* #<Thread:0x000056520a913040@adder-test2.rb:29 sleep_forever>
   rb_thread_t:0x000056520a609fc0 native:0x00007f7ca54d4700 int:0
    depended by: tb_thread_id:0x000056520a5f2ff0
   adder-test2.rb:18:in `pop'
   adder-test2.rb:18:in `block in initialize'
   adder-test2.rb:17:in `times'
   adder-test2.rb:17:in `initialize'
   adder-test2.rb:30:in `new'
   adder-test2.rb:30:in `block (2 levels) in test_adder'
```

I verified this on ruby-trunk, but get the same behavior on all older MRI versions.

History

#1 - 10/28/2018 10:39 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Feedback

- Description updated

larskanis (Lars Kanis) wrote:

> Although start_adder works with a WeakRef, the Adder object should still be GC marked, since Adder.new doesn't return before all calls to start_adder finished.
>
> ```
>       @count.times do
>         Thread.new(WeakRef.new(self), &self.class.method(:start_adder))
>       end
> ```

This method doesn't wait these threads which run start_adder.

Actual behavior:

The program stops with a probability of approximately 80% with the following error:

So this behavior seems expected.

#2 - 10/28/2018 11:52 AM - larskanis (Lars Kanis)

Thanks nobu (Nobuyoshi Nakada) for looking at the issue!

> This method doesn't wait these threads which run start_adder.

It does: initialize waits for all threads to have called Adder#add at the last line at @qu.deq . It only returns after this event.

#3 - 10/29/2018 01:13 AM - nobu (Nobuyoshi Nakada)

larskanis (Lars Kanis) wrote:

> It does: initialize waits for all threads to have called Adder#add at the last line at @qu.deq . It only returns after this event.

It waits just once.

#4 - 10/29/2018 09:00 AM - larskanis (Lars Kanis)

- File adder-test2.rb added

> It waits just once.

Yes, but the one event is sent after all 10 threads have been called, so that @count is decremented to 0.

The same error is present, when we wait for 10 calls to the queue. So the class can actually be simplified like so:

```
class Adder
  def self.start_adder(obj)
    obj.add
  end

  def initialize
    @qu = Queue.new
    count = 10
    count.times do
      Thread.new(WeakRef.new(self), &self.class.method(:start_adder))
    end
    count.times do
      @qu.deq
    end
  end

  def add
    @qu.enq true
  end
end
```

This version raises Invalid Reference - probably recycled (WeakRef::RefError) with a probability of around 50% on my Linux systems. The whole file is attached.

#5 - 10/30/2018 01:34 AM - nobu (Nobuyoshi Nakada)

larskanis (Lars Kanis) wrote:

> > It waits just once.
>
> Yes, but the one event is sent after all 10 threads have been called, so that @count is decremented to 0.

Why do you think so?

#6 - 10/30/2018 03:05 PM - larskanis (Lars Kanis)

- Description updated

- Subject changed from GCing of object in use to WeakRef::RefError for object that is still in use

#7 - 10/30/2018 03:08 PM - larskanis (Lars Kanis)

nobu (Nobuyoshi Nakada) I updated the bug report, so that it should be more understandable now, what's going wrong.

#8 - 10/30/2018 04:16 PM - jeremyevans0 (Jeremy Evans)

I agree, this appears to a be a bug in WeakRef. The Adder instance is not garbage collected, but you still get a WeakRef::RefError when referencing it through the WeakRef instance. Here's a slightly modified example that just prints when the WeakRef::RefError is raised and prints inside Adder#initialize showing that the Adder instance has not yet been garbage collected after the WeakRef::RefError is raised (and the object ids for the instance and @qu match).

```
require "weakref"

class Adder
  def self.start_adder(obj, oid, q)
    obj.add
  rescue WeakRef::RefError
    p [WeakRef::RefError, oid, q.object_id]
    q.push [oid, q.object_id]
  end

  def initialize
    @qu = Queue.new
    count = 10
    count.times do
      Thread.new(WeakRef.new(self), object_id, @qu, &self.class.method(:start_adder))
    end
    count.times do
      oid, q_oid = @qu.pop
      if oid
        p [:initialize, self, oid, object_id, oid == object_id, q_oid == @qu.object_id]
      end
    end
  end

  def add
    @qu.push nil
  end
end

def test_adder
  oids = 10.times.map do
    Thread.new do
      Adder.new
    end
  end.map(&:join)
end

1000.times do
  test_adder
end
```

Output on ruby 2.5.3:

```
[WeakRef::RefError, 16073782609840, 16073782609800]
[:initialize, #<Adder:0x00001d3cf0348f60 @qu=#<Thread::Queue:0x00001d3cf0348f10>>, 16073782609840, 16073782609
840, true, true]
[WeakRef::RefError, 16073295767400, 16073295767360]
[:initialize, #<Adder:0x00001d3cb62b4ed0 @qu=#<Thread::Queue:0x00001d3cb62b4e80>>, 16073295767400, 16073295767
400, true, true]
```

#9 - 10/31/2018 10:07 AM - larskanis (Lars Kanis)

The reference error doesn't appear when WeakRef is replaced by _id2ref like so:

```
  def self.start_adder(oid)
    ObjectSpace._id2ref(oid).add
  end
```

So it appears to be a bug in ObjectSpace::WeakMap.

Files

adder-test2.rb	476 Bytes	10/29/2018	larskanis (Lars Kanis)