Ruby trunk - Bug #15598

Deadlock on mutual reference of autoloaded constants

02/11/2019 12:39 PM - akr (Akira Tanaka)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.7.0dev (2019-02-11 trunk 67049) [x86_64-linux]	**Backport:**	2.4: UNKNOWN, 2.5: UNKNOWN, 2.6: UNKNOWN

Description

Mutual reference of autoloaded constants can cause deadlock sporadically.

Assume A is defined in a.rb and it uses B at loading time.
Also, B is defined in b.rb and it uses A at loading time.

```
% cat a.rb
class A
  def a1() end
  p [__FILE__, __LINE__, B.instance_methods(false)]
  def a2() end
end
% cat b.rb
class B
  def b1() end
  p [__FILE__, __LINE__, A.instance_methods(false)]
  def b2() end
end
```

If they are loaded via autoload and constants are referenced sequentially,
it works (no error, at least).

However, incomplete A (which a2 is not defined) is appear in b.rb, though.

```
% cat base_seq.rb
autoload :A, "./a"
autoload :B, "./b"
A
B
% ruby base_seq.rb
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, [:b1, :b2]]
```

However, the constants are referenced in multi threads,
deadlock can occur, or works like sequential version, sporadically.

```
% cat base_thread_const.rb
autoload :A, "./a"
autoload :B, "./b"
t1 = Thread.new { A }
t2 = Thread.new { B }
t1.join
t2.join
% ruby base_thread_const.rb
Traceback (most recent call last):
    1: from base_thread_const.rb:5:in `<main>'
base_thread_const.rb:5:in `join': No live threads left. Deadlock? (fatal)
3 threads, 3 sleeps current:0x000055f9e2fa1b00 main thread:0x000055f9e2ec14b0
* #<Thread:0x000055f9e2eef188 sleep_forever>
   rb_thread_t:0x000055f9e2ec14b0 native:0x00007f259bc54b40 int:0
   base_thread_const.rb:5:in `join'
   base_thread_const.rb:5:in `<main>'
```

```
* #<Thread:0x000055f9e31ece30@base_thread_const.rb:3 sleep_forever>
   rb_thread_t:0x000055f9e31403c0 native:0x00007f2597e99700 int:0
    depended by: tb_thread_id:0x000055f9e2ec14b0
   /tmp/h/a.rb:3:in `<class:A>'
   /tmp/h/a.rb:1:in `<top (required)>'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   base_thread_const.rb:3:in `block in <main>'
* #<Thread:0x000055f9e31ecbb0@base_thread_const.rb:4 sleep_forever>
   rb_thread_t:0x000055f9e2fa1b00 native:0x00007f258ffff700 int:0
   /tmp/h/b.rb:3:in `<class:B>'
   /tmp/h/b.rb:1:in `<top (required)>'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   base_thread_const.rb:4:in `block in <main>'
% ruby base_thread_const.rb
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, [:b1, :b2]]
```

Also, if "require" is used instead of constant references in the threads,
deadlock can occur (sporadically) too.

Note that incomplete A can appear in b.rb and
incomplete B can appear in a.rb.
The incompleteness vary.

```
% cat base_thread_require.rb
autoload :A, "./a"
autoload :B, "./b"
t1 = Thread.new { require './a' }
t2 = Thread.new { require './b' }
t1.join
t2.join
% ruby base_thread_require.rb
Traceback (most recent call last):
    1: from base_thread_require.rb:5:in `<main>'
base_thread_require.rb:5:in `join': No live threads left. Deadlock? (fatal)
3 threads, 3 sleeps current:0x00005591a27f5190 main thread:0x00005591a24264b0
* #<Thread:0x00005591a24531a0 sleep_forever>
   rb_thread_t:0x00005591a24264b0 native:0x00007feced36ab40 int:0
   base_thread_require.rb:5:in `join'
   base_thread_require.rb:5:in `<main>'
* #<Thread:0x00005591a2754cc8@base_thread_require.rb:3 sleep_forever>
   rb_thread_t:0x00005591a27f5190 native:0x00007fece95af700 int:0
    depended by: tb_thread_id:0x00005591a24264b0
   /tmp/h/a.rb:1:in `<top (required)>'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   base_thread_require.rb:3:in `block in <main>'
* #<Thread:0x00005591a2754a98@base_thread_require.rb:4 sleep_forever>
   rb_thread_t:0x00005591a2506b00 native:0x00007fece13ad700 int:0 mutex:0x00005591a27f5190 cond:1
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   /tmp/h/b.rb:3:in `<class:B>'
   /tmp/h/b.rb:1:in `<top (required)>'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   /home/akr/ruby/o0/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:54:in `require'
   base_thread_require.rb:4:in `block in <main>'
% ruby base_thread_require.rb
["/tmp/h/b.rb", 3, []]
["/tmp/h/a.rb", 3, [:b1]]
% repeat 100 (ruby base_thread_require.rb >& /tmp/z && cat /tmp/z)
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
```

```
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/b.rb", 3, []]
["/tmp/h/a.rb", 3, [:b1]]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, [:b1, :b2]]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, [:b1, :b2]]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, [:b1, :b2]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, [:b1, :b2]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, [:b1]]
["/tmp/h/b.rb", 3, [:a1, :a2]]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, [:b1, :b2]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
["/tmp/h/a.rb", 3, []]
["/tmp/h/b.rb", 3, [:a1]]
```

I think there are several ways to solve this issue.

- Prohibit mutual reference. I.e. raise an error at autoload constant reference currently loading. Since mutual reference causes incomplete definition, it is dangerous even with single thread. However, if real application uses such code, this is incompatible.
- More coarse locking. Since the deadlock is caused because two threads lock the constants in different order: A to B and B to A. I think it is possible to fix this issue by locking whole autoloading procedure by single lock, namely "global autoload lock". Note that it should also be locked by "require" method if it load a file for autoload.

Related issues:

Related to Ruby trunk - Bug #15599: Mixing autoload and require causes deadlo...	**Open**

History

#1 - 03/20/2019 10:47 AM - Eregon (Benoit Daloze)

Should this "global autoload lock" also be locked for normal non-autoload require?
Otherwise I think it could deadlock:

```
T1: require "foo"; AutoloadC;
T2: AutoloadC; require "foo";
```

#2 - 03/20/2019 10:58 AM - akr (Akira Tanaka)

Eregon (Benoit Daloze) wrote:

> Should this "global autoload lock" also be locked for normal non-autoload require?

I think requiring a library which is configured for autoload should lock "global autoload lock".
Currently I think requiring a library which is not configured for autoload should not lock it
because NaHi-san said that some library would contain infinite loop at load time.

Anyway, deadlock reported in [Bug #15599] is a bug.

#3 - 03/20/2019 10:59 AM - akr (Akira Tanaka)

- Related to Bug #15599: Mixing autoload and require causes deadlock and incomplete definition. added