Ruby - Bug #21315

Finalizers violate the `rb_ractor_confirm_belonging` assertion

05/09/2025 04:19 PM - byroot (Jean Boussier)

Status:	Assigned		
Priority:	Normal		
Assignee:	ractor		
Target version:			
ruby -v:		**Backport:**	3.2: UNKNOWN, 3.3: UNKNOWN, 3.4: UNKNOWN

Description

Reproduction (with assertions enabled):

```
require "tempfile"

Ractor.new do
  10.times do
    Tempfile.new
  end
end.take

GC.start
```

```
  <OBJ_INFO:rb_ractor_confirm_belonging@../src/ractor_core.h:375> (File)
  /home/runner/work/ruby/ruby/src/lib/tempfile.rb:387: [BUG] rb_ractor_confirm_belonging object-ra
ctor id:1, current-ractor id:2
  ruby 3.5.0dev (2025-05-09T15:51:24Z pull/13291/merge e289a1dc58) +PRISM [x86_64-linux]

  -- Control frame information -----------------------------------------------
  c:0005 p:0007 s:0025 e:000024 METHOD /home/runner/work/ruby/ruby/src/lib/tempfile.rb:387 [FINISH
]
  c:0004 p:0022 s:0020 e:000019 METHOD /home/runner/work/ruby/ruby/src/lib/prism/parse_result.rb:8
33 [FINISH]
  c:0003 p:---- s:0012 e:000011 CFUNC  :parse_lex_file
  c:0002 p:0006 s:0007 e:000006 BLOCK  /home/runner/work/ruby/ruby/src/test/prism/ractor_test.rb:2
6 [FINISH]
  c:0001 p:---- s:0003 e:000002 DUMMY  [FINISH]

  -- Ruby level backtrace information ----------------------------------------
  /home/runner/work/ruby/ruby/src/test/prism/ractor_test.rb:26:in 'block in test_parse_lex_file'
  /home/runner/work/ruby/ruby/src/test/prism/ractor_test.rb:26:in 'parse_lex_file'
  /home/runner/work/ruby/ruby/src/lib/prism/parse_result.rb:833:in 'initialize'
  /home/runner/work/ruby/ruby/src/lib/tempfile.rb:387:in 'call'

  -- Threading information ---------------------------------------------------
  Total ractor count: 2
  Ruby thread count for this ractor: 1

  -- C level backtrace information -------------------------------------------
  /home/runner/work/ruby/ruby/build/ruby(rb_print_backtrace+0x14) [0x561441aeedbf] ../src/vm_dump.
c:843
  /home/runner/work/ruby/ruby/build/ruby(rb_vm_bugreport) ../src/vm_dump.c:1175
  /home/runner/work/ruby/ruby/build/ruby(bug_report_end+0x0) [0x561441aa3f39] ../src/error.c:1097
  /home/runner/work/ruby/ruby/build/ruby(rb_bug_without_die_internal) ../src/error.c:1097
  /home/runner/work/ruby/ruby/build/ruby(die+0x0) [0x5614416e237f] ../src/error.c:1115
  /home/runner/work/ruby/ruby/build/ruby(rb_bug) ../src/error.c:1117
  /home/runner/work/ruby/ruby/build/ruby(rb_ractor_confirm_belonging+0xde) [0x56144188c6be] ../src
/ractor_core.h:376
  /home/runner/work/ruby/ruby/build/ruby(vm_exec_core+0xef) [0x5614418b23ef] /home/runner/work/rub
y/ruby/build/vm.inc:2500
  /home/runner/work/ruby/ruby/build/ruby(rb_vm_exec+0x156) [0x5614418a4676] ../src/vm.c:2618
  /home/runner/work/ruby/ruby/build/ruby(vm_call0_cc+0x13d) [0x5614418aa1dd] ../src/vm_eval.c:101
```

```
  /home/runner/work/ruby/ruby/build/ruby(rb_vm_call0+0x4c) [0x5614418b1cc5] ../src/vm_eval.c:61
  /home/runner/work/ruby/ruby/build/ruby(rb_vm_call_kw) ../src/vm_eval.c:326
  /home/runner/work/ruby/ruby/build/ruby(rb_check_funcall_default_kw) ../src/vm_eval.c:694
  /home/runner/work/ruby/ruby/build/ruby(rb_gc_run_obj_finalizer+0x1cf) [0x56144170220f] ../src/gc
.c:306
  /home/runner/work/ruby/ruby/build/ruby(rb_multi_ractor_p+0x0) [0x56144170246f] ../src/gc/default
/default.c:2820
  /home/runner/work/ruby/ruby/build/ruby(rb_vm_lock_enter) ../src/vm_sync.h:74
  /home/runner/work/ruby/ruby/build/ruby(rb_gc_vm_lock) ../src/gc.c:137
  /home/runner/work/ruby/ruby/build/ruby(finalize_list) ../src/gc/default/default.c:2843
  /home/runner/work/ruby/ruby/build/ruby(finalize_deferred_heap_pages) ../src/gc/default/default.c
:2866
```

I think it wouldn't be hard to disable that check when running a finalizer, but I don't know if it's the correct thing to do.

Related issues:

Has duplicate Ruby - Bug #21355: `csv/test/csv/interface/test_read.rb` at tes...	**Closed**

History

#1 - 05/09/2025 06:51 PM - jhawthorn (John Hawthorn)

It does seem like it's a problem that this is a way to have access to an unshareable object from multiple ractors

```
def build_finalizer(object)
  previous_ractor = Ractor.current
  proc { puts "finalizing on #{Ractor.current} (was: #{previous_ractor}). Passed object: #{object}" }
end

r = Ractor.new {
  obj1 = Object.new
  obj2 = Object.new
  ObjectSpace.define_finalizer(obj1, build_finalizer(obj2))
  obj1 = nil # allow obj1 to be GC'd
  Ractor.yield :ok
  sleep 1 # We're still running and could access obj2
}

r.take
GC.start
r.take
```

prints

```
finalizing on #<Ractor:#1 running> (was: #<Ractor:#2 ractor_finalizer.rb:6 running>). Passed object: #<Object:
0x000000010451cde0>
```

So we've given access to an object to the main Ractor even though it should not be shareable. It seems like finalizers run on an arbitrary thread so this could also be a problem in the other direction (unshareable object from main Ractor used as part of finalizer on another).

It seems like we should either:

- Run each finalizer in the Ractor that created it (what do we do about terminated ractors?)
- Disallow finalizers in non-main-Ractors (seems restrictive)
- Only allow shareable procs for finalizers in Ractors (seems difficult for users)

I wonder if @ko1's Ractor-local GC will help with this

#2 - 05/09/2025 06:56 PM - byroot (Jean Boussier)

> Run each finalizer in the Ractor that created it (what do we do about terminated ractors?)

That one I think would be doable, we could record the current ractor in with the finalizer.

For terminated Ractors I don't think it's a big deal, as long as we do what is necessary so that Ractor.current and Ractor local storage still work as expected.

#3 - 05/09/2025 07:13 PM - Eregon (Benoit Daloze)

When a Ractor terminates just before it actually does so it should probably run the remaining finalizers, AFAIK, this is what CRuby does when the process is about to terminate and there are finalizers not run yet (but I might be wrong, though it clearly seems to exhibit that behavior).

3 is a no-no, it's already quite difficult to write correct finalizers (which don't capture the object to finalize), it's basically impossible to write them as shareable procs.

#4 - 05/09/2025 07:45 PM - jhawthorn (John Hawthorn)

> When a Ractor terminates just before it actually does so it should probably run the remaining finalizers

Objects aren't necessarily unreferenced/garbage collectable at this point (they could have been made shareable, or be the return value taken by a another Ractor). I guess we could do this, but it does seem extremely surprising behaviour to run a finalizer on an accessible live object.

> For terminated Ractors I don't think it's a big deal, as long as we do what is necessary so that Ractor.current and Ractor local storage still work as expected.

The final return value of a Ractor is not made shareable, so the take-ing Ractor may have access with whatever objects were captured in the finalizer proc. I think we probably need to track that and forward the finalizer to the "next" Ractor (possibly following that several levels until we find a still-running Ractor).

#5 - 05/09/2025 08:04 PM - Eregon (Benoit Daloze)

Eregon (Benoit Daloze) wrote in #note-3:

> it clearly seems to exhibit that behavior

To clarify what I mean:

```
$ ruby -e 'OBJ = Object.new; ObjectSpace.define_finalizer(OBJ) { puts "finalizer ran" }'
finalizer ran
```

But clearly that object is always reachable, so the finalizer is run on a live object in this case.
We can even print the object in its finalizer:

```
$ ruby -e 'OBJ = Object.new; ObjectSpace.define_finalizer(OBJ) { p OBJ }'
#<Object:0x00007f48f9d970a8>
```

> it does seem extremely surprising behaviour to run a finalizer on an accessible live object.

I agree, I haven't yet fully understood how this works in CRuby.
We tried to replicate the behavior in TruffleRuby but it leds to issues because some objects might depend on other objects for finalization, or rely on finalizers of objects they reference to not run before, and so the order can matter and potentially cause segfaults if run in the wrong order.

#6 - 05/12/2025 11:16 PM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned

#7 - 05/14/2025 05:21 PM - luke-gru (Luke Gruber)

I agree with John, I think we need to find the next eligible ractor for finalizers after ractor termination.

#8 - 05/15/2025 07:40 PM - ko1 (Koichi Sasada)

it will be fixed with ractor-local GC patch.

#9 - 06/24/2025 10:51 AM - byroot (Jean Boussier)

- Has duplicate Bug #21355: `csv/test/csv/interface/test_read.rb` at test-bundled-gems is flaky recent days added