Ruby - Bug #21090

SEGV from require in Thread in Ractor

01/26/2025 03:21 PM - wanabe (_ wanabe)

Status:	Closed		
Priority:	Normal		
Assignee:	ractor		
Target version:			
ruby -v:	ruby 3.5.0dev (2025-01-18T00:19:17Z origin/master 65a7c69188) +PRISM [x86_64-linux]	**Backport:**	3.1: UNKNOWN, 3.2: UNKNOWN, 3.3: UNKNOWN, 3.4: UNKNOWN

Description

When ruby calls 'require' in Thread in non-main Ractor, it can cause SEGV sometimes.

```
$ ruby -e '1000.times { Ractor.new { th = Thread.new { require "rbconfig" }; Thread.pass }.take }'
 > segv.log 2>&1
Segmentation fault (core dumped)
```

segv.log is too large to paste in this description, so I attached as a file.

Associated revisions

Revision b28f3443 - 06/12/2025 08:13 PM - jhawthorn (John Hawthorn)

Use a T_DATA for cross_ractor_require

[Bug #21090]

The struct was previously allocated on the stack, which could be freed
if the Thread is terminated. Moving this to a T_DATA on the heap should
mean this is no longer an issue.

1000.times { Ractor.new { th = Thread.new { require "rbconfig" }; Thread.pass }.take }

Co-authored-by: Luke Gruber luke.gruber@shopify.com

History

#1 - 01/27/2025 10:13 PM - luke-gru (Luke Gruber)

I couldn't track down the exact cause of the issue, but I do have a PR coming that solves it.

Edit: PR here https://github.com/ruby/ruby/pull/12646

If you wanted to call th.join in your script, this will still fail. You need another patch that is not yet merged, this one:
https://github.com/ruby/ruby/pull/12520

#2 - 02/02/2025 11:20 PM - luke-gru (Luke Gruber)

I managed to track down the issues, as there was more than 1.

I'll copy my PR message below:

stack struct memory was receiving weird values with lots of ractors when
calling Ractor#require in a thread that wasn't joined.
This was due to the script exiting before the ractor channel yields to the taker.
In that case, the taker can receive a FATAL interrupt and jump to its end, but
the channel might still try to use that stack object from the taker when its thread
starts. For this reason, we allocate on the heap and free after the require ends.

There were 2 more issues as well:

We must close the incoming port of the taker before raising, or another
ractor might try to yield to us. This would use the basket object on the
stack of the taker, but it will be corrupted after a raise.

There was an issue with ractor barriers during GC, any thread calling
ractor_sched_barrier_join_wait_locked must not lock TH_SCHED(th) of

calling thread. Otherwise this could result in a deadlock if another thread tries to lock our sched.

#3 - 05/08/2025 10:38 PM - jhawthorn (John Hawthorn)

- Assignee set to ractor

#4 - 05/12/2025 11:16 PM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned

#5 - 06/12/2025 09:03 PM - jhawthorn (John Hawthorn)

- Status changed from Assigned to Closed

Applied in changeset git|b28f3443122c4e5461877d704618c752e56ca8b0.

Use a T_DATA for cross_ractor_require

[Bug #21090]

The struct was previously allocated on the stack, which could be freed if the Thread is terminated. Moving this to a T_DATA on the heap should mean this is no longer an issue.

1000.times { Ractor.new { th = Thread.new { require "rbconfig" }; Thread.pass }.take }

Co-authored-by: Luke Gruber luke.gruber@shopify.com

Files

segv.log	36.9 KB	01/26/2025	wanabe (_ wanabe)