Ruby master - Bug #13887

test/ruby/test_io.rb may get stuck with FIBER_USE_NATIVE=0 on Linux

09/10/2017 01:19 PM - wanabe (_wanabe)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>ko1 (Koichi Sasada)</td>
</tr>
<tr>
<td>Target version:</td>
<td>ruby -v:</td>
</tr>
</tbody>
</table>

| Backport:    | 2.2: REQUIRED, 2.3: DONE, 2.4: DONE |

**Description**

Eric Wong wrote in https://bugs.ruby-lang.org/issues/13875#note-5 [ruby-core:82708]

However, test/ruby/test_io.rb seems stuck when FIBER_USE_NATIVE is 0 on my system...

I can reproduce with make optflags="-DFIBER_USE_NATIVE=0 -O0" test-all TESTOPTS="$(git rev-parse --show-cdup)/test/ruby/test_io.rb" on my Ubuntu 17.10, glibc 2.24-12.

Here is a reduced code.

This code don't stop with SIGINT (CTRL+C) so I had to use SIGKILL.

```ruby
1000.times do |i|
  p i
  r, w = IO.pipe
  w.puts "foo"
  w.close
  rt = Thread.new do
    r.each_char.next
    r.close
  end
  Thread.new {}.join # <= stuck
  rt.join
end
```

GDB showed ALLOCATE_STACK() had entered in infinity loop at get_cached_stack(), static function of glibc. I guess stack_cache local variable of allocatestack.c of glibc got corrupted.

```
$ gdb --args ./miniruby -v a.rb
(snip)
(gdb) r
Starting program: /home/takira/work/prog/ruby/ruby/tmp/.out.tmp/miniruby -v a.rb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7f2865b5e700 (LWP 5568)]
ruby 2.5.0dev (2017-09-10 trunk 59457) [x86_64-linux]
0
(snip)
[Thread 0x7f8fe2ba4700 (LWP 27769) exited]
51
[Thread 0x7f8fe2da6700 (LWP 27768) exited]
^C
```

Thread 1 "miniruby" received signal SIGINT, Interrupt.
```
0x00007ff8fe4014bc3 in get_cached_stack (memp=<synthetic pointer>, sizep=<synthetic pointer>) at allocatestack.c:194
warning: Source file is more recent than executable.
194 if (FREE_P (curr) && curr->stackblock_size >= size)
(gdb) bt 5
#0 0x00007ff8fe4014bc3 in get_cached_stack (memp=<synthetic pointer>, sizep=<synthetic pointer>) at allocatestack.c:194
#1 allocate_stack (stack=<synthetic pointer>, pdp=<synthetic pointer>, attr=0x7fffffff350) at allocatestack.c:496
```
I've reproduced it with ruby_2_4, ruby_2_3 and ruby_2_2.

Related issues:
Related to Ruby master - Bug #13875: segfault in Enumerable#zip after GC Closed

Associated revisions
Revision 60384 - 10/23/2017 09:50 PM - normalperson (Eric Wong)
thread_pthread: do not corrupt stack

This fixes stuck test/ruby/test_io.rb with FIBER_USE_NATIVE=0 on GNU/Linux because linked-list pointers used by glibc get corrupted when fiber stacks are copied.

Thanks to wanabe for finding the bug and original patch.

- thread_pthread (native_thread_init_stack): fix stack corruption [ruby-core:82737] [Bug #13387]

Revision 12fc8129 - 03/10/2018 02:46 AM - nagachika (Tomoyuki Chikanaga)
merge revision(s) 60384: [Backport #13878]

thread_pthread: do not corrupt stack

This fixes stuck test/ruby/test_io.rb with FIBER_USE_NATIVE=0 on GNU/Linux because linked-list pointers used by glibc get corrupted when fiber stacks are copied.

Thanks to wanabe for finding the bug and original patch.

- * thread_pthread (native_thread_init_stack): fix stack corruption [ruby-core:82737] [Bug #13387]

Revision 62712 - 03/10/2018 02:46 AM - nagachika (Tomoyuki Chikanaga)
merge revision(s) 60384: [Backport #13878]

thread_pthread: do not corrupt stack

This fixes stuck test/ruby/test_io.rb with FIBER_USE_NATIVE=0 on GNU/Linux because linked-list pointers used by glibc get corrupted when fiber stacks are copied.

Thanks to wanabe for finding the bug and original patch.

- * thread_pthread (native_thread_init_stack): fix stack corruption [ruby-core:82737] [Bug #13387]

Revision ca310ba6 - 03/18/2018 03:27 PM - usa (Usaku NAKAMURA)
merge revision(s) 60384: [Backport #13878]

thread_pthread: do not corrupt stack
This fixes stuck test/ruby/test_io.rb with FIBER_USE_NATIVE=0 on GNU/Linux because linked-list pointers used by glibc get corrupted when fiber stacks are copied.

Thanks to wanabe for finding the bug and original patch.

* thread_pthread (native_thread_init_stack): fix stack corruption
[ruby-core:82737] [Bug #13387]

Revision 62825 - 03/18/2018 03:27 PM - usa (Usaku NAKAMURA)
merge revision(s) 60384: [Backport #13887]

thread_pthread: do not corrupt stack

This fixes stuck test/ruby/test_io.rb with FIBER_USE_NATIVE=0 on GNU/Linux because linked-list pointers used by glibc get corrupted when fiber stacks are copied.

Thanks to wanabe for finding the bug and original patch.

* thread_pthread [native_thread_init_stack]: fix stack corruption
[ruby-core:82737] [Bug #13387]

History
#1 - 09/11/2017 05:51 AM - wanabe (_ wanabe)
- Related to Bug #13875: segfault in Enumerable#zip after GC added

#2 - 09/12/2017 06:08 AM - normalperson (Eric Wong)
Another note, FIBER_USE_NATIVE segfaults with my work-in-progress "thriber" implementation:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/82756

I guess I'll investigate tomorrow.

#3 - 09/14/2017 08:23 AM - wanabe (_ wanabe)
Another reproduction code is here.

```ruby
10.times do |i|
  p i
  t = Thread.new { Fiber.new { sleep }.resume } # t.run.join
end
```

#4 - 09/15/2017 04:38 PM - wanabe (_ wanabe)

I guess cont->machine.stack_src should not contain the under thread_start_func_1() stack. Because this area will be used and overwritten by pthread, especially stack_list_{add,del} in nptl/allocatestack.c. If cont_restore_1() copies cont->machine.stack to cont->machine.stack_src, it may be reverted above changes and corrupted stack_cache list structures.

```diff
diff --git a/thread_pthread.c b/thread_pthread.c
index 242b48f15d..50ac751763 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
@@ -841,8 +841,8 @@ native_thread_init_stack(rb_thread_t *th)
   size_t size;
   
 -   if (get_stack(&start, &size) == 0) {
-     th->machine.stack_start = start;
-     th->machine.stack_maxsize = size;
-     th->machine.stack_start = &curr;
-     th->machine.stack_maxsize = size - ((char*)start - (char*)&curr);
+   } #elif defined get_stack_of
     if (!th->machine.stack_maxsize) {
       
 Above patch can prevent the issue.
But I believe it is never correct.
((char*) cast is ugly, I want change cont->machine.stack_src but not th->machine.stack_start, ruby should change the behaviour only when
FIBER_USE_NATIVE == 0, and so on.)

#5 - 09/21/2017 10:28 PM - normalperson (Eric Wong)
s_wanabe@gmail.com wrote:

But I believe it is never correct.
((char*) cast is ugly, I want change cont->machine.stack_src but not th->machine.stack_start, ruby should change the behaviour only when
FIBER_USE_NATIVE == 0, and so on.)

I guess replacing "char *" with "uintptr_t" is appropriate for
pointer arithmetic:

```
diff --git a/thread_pthread.c b/thread_pthread.c
index 96723d4b17..9f959e095 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
@@ -841,8 +841,9 @@
 size_t size;
 if (get_stack(&start, &size) == 0) {
-       th->ec.machine.stack_start = start;
-       th->ec.machine.stack_maxsize = size;
+       uintptr_t diff = (uintptr_t)start - (uintptr_t)&curr;
+       th->ec.machine.stack_start = &curr;
+       th->ec.machine.stack_maxsize = size - diff;
 }
#endif
```

The above works for me as far as test_io.rb goes, but my Thriber
patch for [Feature 
97x466][13618] still segfaults with
FIBER_USE_NATIVE==0, so I guess that is a separate bug I need
to fix...

#6 - 09/23/2017 06:31 PM - normalperson (Eric Wong)
s_wanabe@gmail.com wrote:

I guess cont->machine.stack_src should not contain the under thread_start_func_1() stack.
Because this area will be used and overwritten by pthread, especially stack_list_[add,del] in nptl/allocatestack.c.
If cont_restore_1() copies cont->machine.stack to cont->machine.stack_src, it may be reverted above changes and corrupted stack_cache list
structures.

I think this makes your patch necessary for callcc use
regardless of FIBER_USE_NATIVE value.

Shall I commit [ruby-core:82925]?

My Thriber patch has a different problem with FIBER_USE_NATIVE==0
and I need to use heap alloccation for platforms without native fiber.

#7 - 09/24/2017 12:10 PM - wanabe (_ wanabe)
normalperson (Eric Wong) wrote:

s_wanabe@gmail.com wrote:

I guess cont->machine.stack_src should not contain the under thread_start_func_1() stack.
Because this area will be used and overwritten by pthread, especially stack_list_[add,del] in nptl/allocatestack.c.
If cont_restore_1() copies cont->machine.stack to cont->machine.stack_src, it may be reverted above changes and corrupted stack_cache list
structures.

I think this makes your patch necessary for callcc use
regardless of FIBER_USE_NATIVE value.

Shall I commit [ruby-core:82925]?

Thank you for your advice. I didn't care about callcc.

05/10/2020
Your refined patch [ruby-core:82925] looks good for me, but I want to hear the opinion of Evaluator Maintainer just in case.

Sasada-san, how do you think about the issue?

#8 - 09/24/2017 12:10 PM - wanabe (_ wanabe)
- Assignee set to ko1 (Koichi Sasada)

#9 - 10/17/2017 07:32 PM - normalperson (Eric Wong)
s.wanabe@gmail.com wrote:

Assignee set to ko1 (Koichi Sasada)
https://bugs.ruby-lang.org/issues/13887#change-66853

ko1: ping?

#10 - 10/21/2017 02:51 PM - ko1 (Koichi Sasada)
I can't understand it is correct, but I believe you guys. Please commit it.
Thank you so much.

I'm not sure why previous versions doesn't have same problem.

#11 - 10/23/2017 09:07 PM - normalperson (Eric Wong)
- Backport changed from 2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN to 2.2: REQUIRED, 2.3: REQUIRED, 2.4: REQUIRED

Previous versions (2.2, 2.3, 2.4) do have the same problem.

#12 - 10/24/2017 02:54 PM - wanabe (_ wanabe)
- Status changed from Open to Closed

I'm happy to close this issue because of r60384. Thank you.

#13 - 03/10/2018 02:46 AM - nagachika (Tomoyuki Chikanaga)
- Backport changed from 2.2: REQUIRED, 2.3: REQUIRED, 2.4: REQUIRED to 2.2: REQUIRED, 2.3: REQUIRED, 2.4: DONE

ruby_2_4 r62712 merged revision(s) 60384.

#14 - 03/18/2018 03:27 PM - usa (Usaku NAKAMURA)
- Backport changed from 2.2: REQUIRED, 2.3: REQUIRED, 2.4: DONE to 2.2: REQUIRED, 2.3: DONE, 2.4: DONE

ruby_2_3 r62825 merged revision(s) 60384.