ruby -v:
ruby 1.9.2p290 (2011-07-09 revision 32553) [x86_64-linux]

thread.c says:

- If another thread interrupts this thread (Thread#kill, signal delivery, VM-shutdown request, and so on), ubf() is called (ubf()' means "un-blocking function"). ubf() should interruptfunc()' execution.
- NOTE: You can not execute most of Ruby C API and touch Ruby objects in `func()' and `ubf()', including raising an exception, because current thread doesn't acquire GVL (cause synchronization problem).

VALUE

rb_thread_blocking_region(
rb_blocking_function_t *func, void *data1,
rb_unblock_function_t *ubf, void *data2)

I've created my ubf() function which is called when the Ruby thread in which rb_thread_blocking_region() was called is killed or interrupted. Within my ubf() function I expect not to have the GVL (as the doc says) and I need to run Ruby code, so I use:

ruby_thread_call_with_gvl(terminate_my_C_reactor_with_gvl, NULL);

and I get an error:

[BUG] rb_thread_call_with_gvl: called by a thread which has GVL.

So... is ubf() called with the GVL or not??

Associated revisions

Revision 37394 - 10/31/2012 12:29 AM - drbrain (Eric Hodel)

- thread.c (rb_thread_call_without_gvl2): Note that ubf() may or may not be called with the GVL. Hinted that rb_thread_call_with_gvl() can be used to access ruby functionality. [ruby-trunk - #6433]
used to access ruby functionality. [ruby-trunk - #6433]

Revision 37394 - 10/31/2012 12:29 AM - drbrain (Eric Hodel)

- thread.c (rb_thread_call_without_gvl2): Note that ubf() may or may not be called with the GVL. Hinted that rb_thread_call_with_gvl() can be used to access ruby functionality. [ruby-trunk - #6433]

Revision 37394 - 10/31/2012 12:29 AM - drbrain (Eric Hodel)

- thread.c (rb_thread_call_without_gvl2): Note that ubf() may or may not be called with the GVL. Hinted that rb_thread_call_with_gvl() can be used to access ruby functionality. [ruby-trunk - #6433]

Revision 37394 - 10/31/2012 12:29 AM - drbrain (Eric Hodel)

- thread.c (rb_thread_call_without_gvl2): Note that ubf() may or may not be called with the GVL. Hinted that rb_thread_call_with_gvl() can be used to access ruby functionality. [ruby-trunk - #6433]

History

#1 - 05/15/2012 03:23 AM - ko1 (Koichi Sasada)
(2012/05/14 21:57), ibc (Iñaki Baz Castillo) wrote:

So... is ubf() called with the GVL or not??

No. ubf() will call without GVL. I'm not sure why your code said such error. Can you show us a reproduce-able small code?

PS1: I'm surprising that you want to invoke Ruby code. I only imagine ubf() as a small code such as "toggle cancel flag" or "kick some API to cancel invoking situation". Maybe you shouldn't use Ruby code (Ruby code can be run after blocking process is canceled).

PS2: ubf() is acronym of un-block function. "ubf() function" is redundant :)

--
// SASADA Koichi at atdot dot net

#2 - 05/15/2012 04:15 AM - ibc (Iñaki Baz Castillo)

ko1 (Koichi Sasada) wrote:

(2012/05/14 21:57), ibc (Iñaki Baz Castillo) wrote:

No. ubf() will call without GVL. I'm not sure why your code said such error. Can you show us a reproduce-able small code?

My code clearly says "[BUG] rb_thread_call_with_gvl: called by a thread which has GVL.".

You can reproduce it by downloading it (Ruby 1.9.2 or perhaps 1.9.3 required):

~# git clone git://github.com/ibc/AsyncEngine.git
~# cd AsyncEngine/
~# rake (it should success)
~# cd test/others/
~# ruby crash_ubf_with_gvl.rb

C DBG: run_uv_without_gvl() starts
----------C NOTICE: ae_thread_killed() starts, running rb_thread_call_with_gvl(terminate_uv_with_gvl, NULL)...

~# ruby crash_ubf_with_gvl.rb

CDBG: run_uv_with_gvl() starts
----------C NOTICE: ae_thread_killed() starts, running rb_thread_call_with_gvl(terminate_uv_with_gvl, NULL)...

-- control frame ----------
c:0004 p:---- s:0010 b:0010 l:000009 d:000009 CFUNC .kill
C:0002 p:---- s:0004 b:0004 l:000003 d:000003 FINISH
-- Ruby level backtrace information ----------------------------------------
crash_ubf_with_gvl.rb:15:in `<main>'
crash_ubf_with_gvl.rb:15:in`kill'
-- C level backtrace information ------------------------------------------
/usr/lib/libruby-1.9.1.so.1.9((rb_vm_bugreport+0x5f) [0x7fdcffe6f4b1f])
/usr/lib/libruby-1.9.1.so.1.9(+0x5b304) [0x7fdff70304]
/usr/lib/libruby-1.9.1.so.1.9((rb_bug+0xb3) [0x7fdff70473])
/usr/lib/libruby-1.9.1.so.1.9((rb_thread_call_with_gvl+0x160) [0x7fdff68680])
/tmp/kk/AsyncEngine/lib/asyncengine/asyncengine_ext.so(+0x0f4) [0x7fdff88904]
/usr/lib/libruby-1.9.1.so.1.9((rb_threadptr_interrupt+0x3c) [0x7fdff659bc])
/usr/lib/libruby-1.9.1.so.1.9((rb_thread_kill+0x48) [0x7fdff680a8])
/usr/lib/libruby-1.9.1.so.1.9(+0x148be8) [0x7fdff5db8e]
/usr/lib/libruby-1.9.1.so.1.9(+0x140dd0) [0x7fdff55dd0]
/usr/lib/libruby-1.9.1.so.1.9(+0x1457f1) [0x7fdff5a7f1]
/usr/lib/libruby-1.9.1.so.1.9((rb_iseq_eval_main+0xb2) [0x7fdff5aa42])
/usr/lib/libruby-1.9.1.so.1.9(+0x5e4c2) [0x7fdff73ac2]
/usr/lib/libruby-1.9.1.so.1.9((ruby_exec_node+0x1d) [0x7fdff734ed])
/usr/lib/libruby-1.9.1.so.1.9((ruby_run_node+0x1e) [0x7fdff74ade])
ruby(main+0x4b) [0x40099b]
/lib/libc.so.6(__libc_start_main+0xfd) [0x7fdff0c8d]
ruby() [0x400889]

The main C file is ext/asyncengine/asyncengine_ruby.c.

lib/asyncengine.rb contains the method AsyncEngine.run (AE.run) which invokes AE._c_run defined in asyncengine_ruby.c. This method uses rb_thread_call_without_gvl() to start the UV loop, which blocks. When an event occurs, the Ruby callback is called with rb_thread_call_with_gvl() (this works ok). When a signal (i.e. Interrupted) is received in the thread running AE, the ubf function ae_thread_killed() is called, and that function uses rb_thread_call_with_gvl(terminate_uv_with_gvl, NULL).

The function terminate_uv_with_gvl() runs the Ruby method AE.stop, which iterates over all the existing UV handles (in a @_handles Hash) and invokes #destroy on each of them, so they are closed by UV and the C blocking function uv_run() exits.

So I need to run AE.stop from the ubf function since it's hard to clean a Hash from C (rb_hash_clear is not public). The problem is that, when the ubf function calls to rb_thread_call_with_gvl(terminate_uv_with_gvl, NULL), the above error occurs.

PS1: I'm surprising that you want to invoke Ruby code. I only imagine ubf() as a small code such as "toggle cancel flag" or "kick some API to cancel invoking situation". Maybe you shouldn't use Ruby code (Ruby code can be run after blocking process is canceled).

uv_run() (from the C library UV) just exits when all its handles have been closed with the uv_closer() function. I store the uv handles within C structures in Ruby AsyncEngine classes (i.e. AsyncEngine::Timer), so I need to iterate over all the Ruby AsyncEngine::XXXXX instances (stored in AE.@_handles) and invoke the #destroy method on them.

#3 - 05/15/2012 04:53 AM - ko1 (Koichi Sasada)

Hi,

(2012/05/15 4:15), ibc (Iñaki Baz Castillo) wrote:

You can reproduce it by downloading it (Ruby 1.9.2 or perhaps 1.9.3 required):

Thank you. But I want to read a small code....

Okay, maybe I understand your problem.

Let us clear them:
(1) There are several threads and main threads
(2) main thread kills another blocking thread

Right?

In this case, ubf() will be called by a thread "Thread#kill" (or signal, etc). Main thread has GVL, and call ubf() to cancel another thread. This is why the error occurs.

FYI:
Maybe you expect that ubf() was called by blocking thread. However, it is not.

In last comment, I made a mistake that "ubf() is not called with GVL". It's my mistake. Correct answer is "ubf() is possible to be called with/without GVL".

We can't use "call_with_gvl()" in ubf().

You need call shutdown/destruct codes after unblocking.

Regards,
Koichi

--
// SASADA Koichi at atdot dot net

#4 - 05/15/2012 07:59 AM - ibc (Iñaki Baz Castillo)

ko1 (Koichi Sasada) wrote:

Okay, maybe I understand your problem.

Let us clear them:
(1) There are several threads and main threads
(2) main thread kills another blocking thread

Right?

In this case, ubf() will be called by a thread “Thread#kill” (or signal, etc). Main thread has GVL, and call ubf() to cancel another thread. This is why the error occurs.

FYI:

/usr/lib/libruby-1.9.1.so.1.9(rb_thread_kill+0x48) [0x7fdcfff680a8]
is Thread#kill code.

Maybe you expect that ubf() was called by blocking thread. However, it is not.

Hi, within my ubf() function I printf the value of ruby_thread_has_gvl_p() and I get 1.

In last comment, I made a mistake that "ubf() is not called with GVL". It's my mistake. Correct answer is "ubf() is possible to be called with/without GVL".

We can't use "call_with_gvl()" in ubf().

Clear.

You need call shutdown/destruct codes after unblocking.

That is what I don't understand. How to call those shutdown/destruct codes after my ubf() function? And what should then my ubf() function do?

I've changed my code now. My ubf() function does not call to rb_thread_call_with_gvl() (since it already has the GVL). Then it "safely" invokes my AE stop method which terminates all the UV handles. But doing that, uv_run() exits so the C code within the run_uv_without_gvl() continues and terminates (of course without the GVL). After that the program gets frozen. If I press Ctrl+C (Interrupt signal) then I see my ubf() being calling again! Also with GVL! and then the program gets really frozen (kill -9 needed).

Before you said: “Maybe you shouldn't use Ruby code (Ruby code can be run after blocking process is canceled)”. What do you exactly mean? Where to put that Ruby code?

Thanks a lot.
ibc (Iñaki Baz Castillo) wrote:

Maybe you expect that ubf() was called by blocking thread. However, it is not.

Hi, within my ubf() function I print the value of ruby_thread_has_gvl_pl() and I get 1.

My fault, the blocking thread is the one without GVL :)

I've re-read the doc:

- If another thread interrupts this thread (Thread#kill, signal delivery, VM-shutdown request, and so on), ubf() is called (ubf() means *un-blocking function*).
- ubf() should interrupt func()'s execution.

But my ubf() does not do that. It closes all the UV handles so uv_run() (which was blocking) terminates and the rest of func() is executed until func() terminates and returns Qtrue.

Honestly I don't understand what "ubf() should interrupt func()'s execution" means :(

Simple example (pseudo-code):

```ruby
$interrupted_flag = false

Thread1:
begin
call_without_gvl(blocking_process: ->{
while $interrupted_flag == false
# do something
end
},
ubf: -> {
$interrupted_flag = true
}
)
ensure
if $interrupted_flag == true
# can detect interrupt
# you can do clean up process such as #destroy
end
end

Thread2:
thread1.raise("interrupt")

Thread2 invoke ubf() and then, blocking_process() can cancel because blocking_process() checks $interrupted_flag. You can also detect interruption after that.

--
// SASADA Koichi at atdot dot net
```

Thanks, I was suggested to look at bignum.c which makes use of a ubf similar to your pseudo-code. I will try to apply it.

Thanks a lot.

PS: BTW this bug can be closed, but still the doc says that the ubf() is called without GVL while my code demonstrates that it's called with the GVL.
#9 - 05/16/2012 10:08 PM - mame (Yusuke Endoh)
  - Status changed from Open to Assigned
  - Assignee set to ko1 (Koichi Sasada)

#10 - 10/30/2012 09:14 AM - ko1 (Koichi Sasada)
  - Target version changed from 1.9.2 to 2.0.0

Need to check document (someone's helps are welcome!)

#11 - 10/31/2012 09:31 AM - drbrain (Eric Hodel)
  - Status changed from Assigned to Closed

I updated the documentation in r37393 and r37394 for natural english and to mention that the ubf() may not be called from the GVL.