Ruby - Bug #21270

init_fast_fallback_inetsock_internal (default for TCPSocket.new/TCPSocket.open) isn't fiber aware

04/16/2025 06:19 PM - drahosj (Jake Drahos)

Status:	Assigned		
Priority:	Normal		
Assignee:	ioquatix (Samuel Williams)		
Target version:			
ruby -v:	ruby 3.4.2	**Backport:**	3.2: UNKNOWN, 3.3: UNKNOWN, 3.4: UNKNOWN

Description

Sockets created via init_fast_fallback_inetsock_internal() don't get a chance to call the scheduler hook(s). This is the default for connections created with TCPSocket.new unless fast_fallback is specified as false. Unfortunately, this has the effect of blocking all fibers in the thread if the initial connect() call hangs. TCPSocket.open() is used in Net::HTTP, which means the issue is present there.

Here's a quick proof of concept. It works as intended as written (starts both fibers even though the first fiber hangs). Switching to the default TCPSocket.new call causes the whole thread to block.

NOTE: Uses the "toy" scheduler from test/fiber/scheduler.rb, but the Socketry async gem is also affected (should affect all schedulers since the hook isn't called).

```
require 'socket'
require_relative 'scheduler'

Fiber.set_scheduler(Scheduler.new)
puts "#{Fiber.current.object_id}: Main fiber"

Fiber.schedule do
  puts "#{Fiber.current.object_id}: Creating socket"
  # Assuming that attempting to connect to example.com on port 12345 hangs
  # Default causes the scheduler to hang and never create second fiber
  # TCPSocket.new("example.com", 12345)
  TCPSocket.new("example.com", 12345, fast_fallback: false)
  puts "#{Fiber.current.object_id}: Connected"
end

Fiber.schedule do
  puts "#{Fiber.current.object_id}: Sleeping"
  sleep 2
  puts "#{Fiber.current.object_id}: Done sleeping"
end

puts "#{Fiber.current.object_id}: Both fibers started"
```

Running the working PoC:

```
$ ruby async-connect.rb
16: Main fiber
24: Creating socket
32: Sleeping
16: Both fibers started
32: Done sleeping
[Hangs here until the socket connection eventually times out]
```

However, without fast_fallback: false, the TCPSocket.new call will block the entire scheduler, never creating the second fiber:

```
$ ruby async-request.rb
16: Main fiber
24: Creating socket
[Hangs here until the connect times out]
```

Here's a stack dump of the hung version:

```
#0  __syscall_cancel_arch () at ../sysdeps/unix/sysv/linux/x86_64/syscall_cancel.S:56
#1  0x00007fc9e4c9581c in __internal_syscall_cancel (a1=<optimized out>, a2=<optimized out>,
    a3=<optimized out>, a4=<optimized out>, a5=a5@entry=0, a6=a6@entry=0, nr=270) at cancellation.
c:49
#2  0x00007fc9e4c95871 in __syscall_cancel (a1=<optimized out>, a2=<optimized out>, a3=<optimized
out>,
    a4=<optimized out>, a5=a5@entry=0, a6=a6@entry=0, nr=270) at cancellation.c:75
#3  0x00007fc9e4d1af07 in __GI___select (nfds=<optimized out>, readfds=<optimized out>,
    writefds=<optimized out>, exceptfds=<optimized out>, timeout=<optimized out>)
    at ../sysdeps/unix/sysv/linux/select.c:69
#4  0x00007fc9e5233c9f in rb_fd_select (n=<optimized out>, readfds=<optimized out>,
    writefds=<optimized out>, exceptfds=<optimized out>, timeout=<optimized out>)
    at /usr/src/debug/ruby-3.4.2/thread.c:4163
#5  0x00007fc9e5236cc6 in native_fd_select (n=<optimized out>, readfds=<optimized out>,
    writefds=<optimized out>, exceptfds=<optimized out>, timeout=<optimized out>, th=<optimized ou
t>)
    at /usr/src/debug/ruby-3.4.2/thread_pthread.c:2380
#6  do_select (p=p@entry=140504668626880) at /usr/src/debug/ruby-3.4.2/thread.c:4314
#7  0x00007fc9e50e7416 in rb_ensure (b_proc=0x7fc9e52368b0 <do_select>, data1=140504668626880,
    e_proc=0x7fc9e5232440 <select_set_free>, data2=140504668626880)
    at /usr/src/debug/ruby-3.4.2/eval.c:1074
#8  0x00007fc9e523700f in rb_thread_fd_select (max=max@entry=21, read=read@entry=0x7fc9cadce430,
    write=write@entry=0x7fc9cadce440, except=except@entry=0x0, timeout=timeout@entry=0x0)
    at /usr/src/debug/ruby-3.4.2/thread.c:4374
#9  0x00007fc9c97b5050 in init_fast_fallback_inetsock_internal (v=v@entry=140504668627872)
    at /usr/src/debug/ruby-3.4.2/ext/socket/ipsocket.c:894
#10 0x00007fc9e50e7416 in rb_ensure (
    b_proc=b_proc@entry=0x7fc9c97b4950 <init_fast_fallback_inetsock_internal>,
    data1=data1@entry=140504668627872,
    e_proc=e_proc@entry=0x7fc9c97b7880 <fast_fallback_inetsock_cleanup>,
    data2=data2@entry=140504668627872) at /usr/src/debug/ruby-3.4.2/eval.c:1074
#11 0x00007fc9c97b7f3c in rsock_init_inetsock (self=self@entry=140504646018560,
    remote_host=remote_host@entry=140504646018640, remote_serv=remote_serv@entry=24691,
    local_host=local_host@entry=4, local_serv=local_serv@entry=4, type=type@entry=0,
    resolv_timeout=<optimized out>, connect_timeout=<optimized out>, fast_fallback=<optimized out>
,
    test_mode_settings=<optimized out>) at /usr/src/debug/ruby-3.4.2/ext/socket/ipsocket.c:1285
#12 0x00007fc9c97b8216 in tcp_init (argc=<optimized out>, argv=<optimized out>, sock=1405046460185
60)
    at /usr/src/debug/ruby-3.4.2/ext/socket/tcpsocket.c:76
```

tcp_init is the TCPSocket#initialize defined in tcpsocket.c, which just calls into the pure-C stack without ever backing out to check io_wait as is done in other socket-creation code path. It looks like the main difference is that init_fast_fallback_inetsock_internal hasn't been updated to be Fiber-aware (eg. init_inetsock_internal calls rsock_connect which uses the Fiber-aware wait_connectable), but init_fast_fallback_inetsock_internal is hard-coded to use thread stuff.

In case anybody stumbles across this wondering why Net::HTTP sometimes blocks the Fiber scheduler during connect(), here's a quick workaround by hacking default fast_fallback: false into TCPSocket.open:

```
orig_open = TCPSocket.method(:open)
TCPSocket.define_singleton_method(:open) do |*args, **kwargs, &block|
  kwargs[:fast_fallback] ||= false
  orig_open.call(*args, **kwargs, &block)
end
```

A slightly fancier workaround would be to have TCPSocket#initialize default fast_fallback to false if the current scheduler is non-nil. That would get rid of unexpected scheduler blocking without requiring a rewrite of the fast fallback code to be fiber-aware. I'm not sure if that's as trivial to implement in a c-defined method as it is in a native Ruby definition.

History

#1 - 04/16/2025 06:33 PM - drahosj (Jake Drahos)

Ignore the hack workaround above; the default for fast fallback is exposed and can be set back to false per 3.4.0 release notes.
https://www.ruby-lang.org/en/news/2024/12/25/ruby-3-4-0-released/

```
require 'async' # or any other fiber scheduler implementation
require 'net/http'

Socket.tcp_fast_fallback = false

# Net::HTTP is now happy and Fiber-aware
```

#2 - 04/16/2025 11:20 PM - byroot (Jean Boussier)

- Assignee set to ioquatix (Samuel Williams)

#3 - 05/12/2025 11:16 PM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned