Ruby - Bug #19624

Backticks - IO object leakage

04/30/2023 02:56 PM - pineman (João Pinheiro)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 3.2.2 (2023-03-30 revision e51014f9c0) [arm64-darwin22]	**Backport:**	3.0: UNKNOWN, 3.1: UNKNOWN, 3.2: UNKNOWN

Description

Hi,

This code works on ruby 3.0.6:

```
`echo`
ObjectSpace.each_object(IO) do |io|
  if ![STDIN, STDOUT, STDERR].include?(io)
    io.close
  end
end
```

but raises IOError on 3.2.2:

```
minimal-repro-case.rb:8:in `close': uninitialized stream (IOError)
```

I found it started failing on ruby 3.1.0 and after, on macOS and Linux.
This code is useful for closing unneeded IO objects in forked processes.
It looks like backticks is 'leaking' IO objects, waiting for GC, and it didn't used to before 3.1.0.
In ruby 3.1.0, inside rb_f_backquote in io.c, rb_gc_force_recycle was removed in favor of RB_GC_GUARD (commit aeae6e2842e). I wonder if this has something to do with the problem.
Is this code incorrect since ruby 3.1.0 or is it a bug in ruby?
Thanks.

Associated revisions

Revision 814f52a9ebd035ec6e20641c602fa42f64b5dbe0 - 04/30/2023 08:10 PM - nobu (Nobuyoshi Nakada)

[Bug #19624] Hide internal IO for backquote

Revision 814f52a9ebd035ec6e20641c602fa42f64b5dbe0 - 04/30/2023 08:10 PM - nobu (Nobuyoshi Nakada)

[Bug #19624] Hide internal IO for backquote

Revision 814f52a9 - 04/30/2023 08:10 PM - nobu (Nobuyoshi Nakada)

[Bug #19624] Hide internal IO for backquote

Revision ab637cad2b582e8247bafd87a3b0f6323d564f64 - 09/21/2023 01:23 AM - nobu (Nobuyoshi Nakada)

[Bug #19624] Clean up backquote IO

It should not be hidden, since it can be grabbed by a fiber scheduler.

Revision ab637cad2b582e8247bafd87a3b0f6323d564f64 - 09/21/2023 01:23 AM - nobu (Nobuyoshi Nakada)

[Bug #19624] Clean up backquote IO

It should not be hidden, since it can be grabbed by a fiber scheduler.

Revision ab637cad - 09/21/2023 01:23 AM - nobu (Nobuyoshi Nakada)

[Bug #19624] Clean up backquote IO

It should not be hidden, since it can be grabbed by a fiber scheduler.

History

#1 - 04/30/2023 02:58 PM - pineman (João Pinheiro)

- Description updated

#2 - 04/30/2023 03:02 PM - pineman (João Pinheiro)

- Description updated

#3 - 04/30/2023 03:02 PM - pineman (João Pinheiro)

- Description updated

#4 - 04/30/2023 08:10 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset git|814f52a9ebd035ec6e20641c602fa42f64b5dbe0.

[Bug #19624] Hide internal IO for backquote

#5 - 09/20/2023 09:14 AM - byroot (Jean Boussier)

Note: hiding the IO created a regression with the fiber scheduler, since this instance can end up being passed to Ruby code, and it's invalid for Ruby code to touch an hidden object: https://github.com/ruby/ruby/pull/8485

#6 - 09/20/2023 11:16 AM - Eregon (Benoit Daloze)

I think these IO objects should not be hidden. They are not leaked.
ObjectSpace.each_object(IO) do |io| ... io.close is incorrect, it needs to account for possible IOError's.

#7 - 09/20/2023 11:18 AM - byroot (Jean Boussier)

Agreed.

@nobu (Nobuyoshi Nakada) any objection to reverting 814f52a9ebd035ec6e20641c602fa42f64b5dbe0 ?

#8 - 09/20/2023 11:57 AM - nobu (Nobuyoshi Nakada)

Is this OK?

```
diff --git a/io.c b/io.c
index 712dce3ceb8..48cdc5b9a7a 100644
--- a/io.c
+++ b/io.c
@@ -5574,12 +5574,9 @@ clear_codeconv(rb_io_t *fptr)
     clear_writeconv(fptr);
 }

-void
-rb_io_fptr_finalize_internal(void *ptr)
+static void
+rb_io_fptr_cleanup_all(rb_io_t *fptr)
 {
-    rb_io_t *fptr = ptr;
-
-    if (!ptr) return;
     fptr->pathv = Qnil;
     if (0 <= fptr->fd)
         rb_io_fptr_cleanup(fptr, TRUE);
@@ -5587,7 +5584,14 @@ rb_io_fptr_finalize_internal(void *ptr)
     free_io_buffer(&fptr->rbuf);
     free_io_buffer(&fptr->wbuf);
     clear_codeconv(fptr);
-    free(fptr);
+}
+
+void
+rb_io_fptr_finalize_internal(void *ptr)
+{
+    if (!ptr) return;
+    rb_io_fptr_cleanup_all(ptr);
+    free(ptr);
 }
```

```
 #undef rb_io_fptr_finalize
@@ -10467,11 +10471,9 @@ rb_f_backquote(VALUE obj, VALUE str)
     if (NIL_P(port)) return rb_str_new(0,0);

     GetOpenFile(port, fptr);
-    rb_obj_hide(port);
     result = read_all(fptr, remain_size(fptr), Qnil);
     rb_io_close(port);
-    RFILE(port)->fptr = NULL;
-    rb_io_fptr_finalize(fptr);
+    rb_io_fptr_cleanup_all(fptr);
     RB_GC_GUARD(port);

     return result;
```

#9 - 09/20/2023 12:00 PM - byroot (Jean Boussier)

@nobu (Nobuyoshi Nakada) Interesting. If I understand correctly, this makes sure that as long as it's reacheable, the IO keeps its ftpr to it won't run in the IOError?

If so yes that seems like a much cleaner solution than hiding it.

#10 - 09/20/2023 12:34 PM - nobu (Nobuyoshi Nakada)

- Status changed from Closed to Open

#11 - 09/20/2023 12:52 PM - nobu (Nobuyoshi Nakada)

@byroot (Jean Boussier) Do you have any short reproducible code?

#12 - 09/20/2023 01:01 PM - byroot (Jean Boussier)

> Do you have any short reproducible code?

No sorry, I don't understand the fiber scheduler enough to reduce the test suite.

I discovered the bug via: https://github.com/rmosolgo/graphql-ruby/issues/4640#issuecomment-1727220148

```
GraphQL::Dataloader::AsyncDataloader::With the toy scheduler from Ruby's tests#test_0003_works with GraphQL:
NotImplementedError: method `hash' called on hidden T_FILE object (0x0000000119cd5ca0 flags=0xb)
    src/graphql-ruby/spec/support/dummy_scheduler.rb:159:in `io_wait'
    src/graphql-ruby/spec/graphql/dataloader/async_dataloader_spec.rb:69:in ``'
    src/graphql-ruby/spec/graphql/dataloader/async_dataloader_spec.rb:69:in `sleep'
```

I tried extracting their scheduler to make a self contained repro, but without success:

```
class DummyScheduler
  def initialize
    @readable = {}
    @writable = {}
    @waiting = {}

    @closed = false

    @lock = Mutex.new
    @blocking = 0
    @ready = []

    @urgent = IO.pipe
  end

  attr :readable
  attr :writable
  attr :waiting

  def next_timeout
    _fiber, timeout = @waiting.min_by{|key, value| value}

    if timeout
      offset = timeout - current_time
```

```
      if offset < 0
        return 0
      else
        return offset
      end
    end
  end

  def run
    # $stderr.puts [__method__, Fiber.current].inspect

    while @readable.any? or @writable.any? or @waiting.any? or @blocking.positive?
      # Can only handle file descriptors up to 1024...
      readable, writable = IO.select(@readable.keys + [@urgent.first], @writable.keys, [], next_timeout)

      # puts "readable: #{readable}" if readable&.any?
      # puts "writable: #{writable}" if writable&.any?

      selected = {}

      readable && readable.each do |io|
        if fiber = @readable.delete(io)
          selected[fiber] = IO::READABLE
        elsif io == @urgent.first
          @urgent.first.read_nonblock(1024)
        end
      end

      writable && writable.each do |io|
        if fiber = @writable.delete(io)
          selected[fiber] |= IO::WRITABLE
        end
      end

      selected.each do |fiber, events|
        fiber.resume(events)
      end

      if @waiting.any?
        time = current_time
        waiting, @waiting = @waiting, {}

        waiting.each do |fiber, timeout|
          if fiber.alive?
            if timeout <= time
              fiber.resume
            else
              @waiting[fiber] = timeout
            end
          end
        end
      end

      if @ready.any?
        ready = nil

        @lock.synchronize do
          ready, @ready = @ready, []
        end

        ready.each do |fiber|
          fiber.resume
        end
      end
    end
  end

  def close
    # $stderr.puts [__method__, Fiber.current].inspect

    raise "Scheduler already closed!" if @closed

    self.run
  ensure
    @urgent.each(&:close)
```

```
    @urgent = nil

    @closed = true

    # We freeze to detect any unintended modifications after the scheduler is closed:
    self.freeze
  end

  def closed?
    @closed
  end

  def current_time
    Process.clock_gettime(Process::CLOCK_MONOTONIC)
  end

  def timeout_after(duration, klass, message, &block)
    fiber = Fiber.current

    self.fiber do
      sleep(duration)

      if fiber && fiber.alive?
        fiber.raise(klass, message)
      end
    end

    begin
      yield(duration)
    ensure
      fiber = nil
    end
  end

  def process_wait(pid, flags)
    # $stderr.puts [__method__, pid, flags, Fiber.current].inspect

    # This is a very simple way to implement a non-blocking wait:
    Thread.new do
      Process::Status.wait(pid, flags)
    end.value
  end

  def io_wait(io, events, duration)
    p io
    # $stderr.puts [__method__, io, events, duration, Fiber.current].inspect

    unless (events & IO::READABLE).zero?
      @readable[io] = Fiber.current
    end

    unless (events & IO::WRITABLE).zero?
      @writable[io] = Fiber.current
    end

    Fiber.yield
  end

  # Used for Kernel#sleep and Mutex#sleep
  def kernel_sleep(duration = nil)
    # $stderr.puts [__method__, duration, Fiber.current].inspect

    self.block(:sleep, duration)

    return true
  end

  # Used when blocking on synchronization (Mutex#lock, Queue#pop, SizedQueue#push, ...)
  def block(blocker, timeout = nil)
    # $stderr.puts [__method__, blocker, timeout].inspect

    if timeout
      @waiting[Fiber.current] = current_time + timeout
      begin
        Fiber.yield
```

```
      ensure
        # Remove from @waiting in the case #unblock was called before the timeout expired:
        @waiting.delete(Fiber.current)
      end
    else
      @blocking += 1
      begin
        Fiber.yield
      ensure
        @blocking -= 1
      end
    end
  end

  # Used when synchronization wakes up a previously-blocked fiber (Mutex#unlock, Queue#push, ...).
  # This might be called from another thread.
  def unblock(blocker, fiber)
    # $stderr.puts [__method__, blocker, fiber].inspect
    # $stderr.puts blocker.backtrace.inspect
    # $stderr.puts fiber.backtrace.inspect

    @lock.synchronize do
      @ready << fiber
    end

    io = @urgent.last
    io.write_nonblock('.')
  end

  def fiber(&block)
    fiber = Fiber.new(blocking: false, &block)

    fiber.resume

    return fiber
  end

  def address_resolve(hostname)
    Thread.new do
      Addrinfo.getaddrinfo(hostname, nil).map(&:ip_address).uniq
    end.value
  end
end

Fiber.set_scheduler(DummyScheduler.new)

p `sleep 2`
```

Perhaps @ioquatix (Samuel Williams) would know how to reproduce?

#13 - 09/20/2023 02:54 PM - nobu (Nobuyoshi Nakada)

OK, https://github.com/nobu/ruby/tree/backquote-io-cleanup

#14 - 09/20/2023 03:21 PM - byroot (Jean Boussier)

@nobu (Nobuyoshi Nakada) looks good!

#15 - 09/21/2023 01:23 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset git|ab637cad2b582e8247bafd87a3b0f6323d564f64.

[Bug #19624] Clean up backquote IO

It should not be hidden, since it can be grabbed by a fiber scheduler.

Files

minimal-repro-case.rb	109 Bytes	04/30/2023	pineman (João Pinheiro)