Ruby - Bug #13405

IO#close raises "stream closed"

04/05/2017 07:35 PM - matthewd (Matthew Draper)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

IO#close is supposed to ignore an IOError indicating the stream is already closed.

Since #10153, however, it can raise the ruby_error_closed_stream special exception, because that exception's message is "stream closed" instead of "closed stream".

The fix seems easy -- the mismatched string should be updated to use the more common spelling, with something like:

```
diff --git a/test/lib/test/unit.rb b/test/lib/test/unit.rb
index 6cb22e725a..d1efa5dcfd 100644
--- a/test/lib/test/unit.rb
+++ b/test/lib/test/unit.rb
@@ -228,7 +228,7 @@ def run(task,type)
           rescue Errno::EPIPE
             died
           rescue IOError
-            raise unless ["stream closed","closed stream"].include? $!.message
+            raise unless $!.message == "closed stream"
             died
           end
         end
diff --git a/test/lib/test/unit/parallel.rb b/test/lib/test/unit/parallel.rb
index 50d4427189..09a5530b04 100644
--- a/test/lib/test/unit/parallel.rb
+++ b/test/lib/test/unit/parallel.rb
@@ -61,7 +61,7 @@ def _run_suite(suite, type) # :nodoc:
         begin
           th.join
         rescue IOError
-          raise unless ["stream closed","closed stream"].include? $!.message
+          raise unless $!.message == "closed stream"
         end
         i.close

diff --git a/test/ruby/test_io.rb b/test/ruby/test_io.rb
index ca3f1e2d3b..5775e31dde 100644
--- a/test/ruby/test_io.rb
+++ b/test/ruby/test_io.rb
@@ -3411,7 +3411,7 @@ def test_race_closed_stream
       end
       sleep 0.01
       r.close
-      assert_raise_with_message(IOError, /stream closed/) do
+      assert_raise_with_message(IOError, /closed stream/) do
         thread.join
       end
       assert_equal(true, closed, "#{bug13158}: stream should be closed")
diff --git a/thread.c b/thread.c
index 5d27681b40..4e6faf6dc8 100644
--- a/thread.c
+++ b/thread.c
@@ -4883,7 +4883,7 @@ Init_Thread(void)
```

```
     rb_define_method(rb_cThread, "name=", rb_thread_setname, 1);
     rb_define_method(rb_cThread, "inspect", rb_thread_inspect, 0);

-    rb_vm_register_special_exception(ruby_error_closed_stream, rb_eIOError, "stream closed");
+    rb_vm_register_special_exception(ruby_error_closed_stream, rb_eIOError, "closed stream");

     cThGroup = rb_define_class("ThreadGroup", rb_cObject);
     rb_define_alloc_func(cThGroup, thgroup_s_alloc);
```

I can't work out how to prove this fixes the problem, however. :(

The existing test in test_io.rb shows how to cause the special exception to be raised from gets, but I haven't managed to synthetically force close to raise it.

I know it's possible, though: we're seeing this problem semi-frequently in the Rails test suite (e.g. https://travis-ci.org/rails/rails/jobs/218895049#L499) -- though it's partly hidden by #13239 when it occurs on 2.2 and 2.3.

Related issues:

Related to Ruby - Feature #10718: IO#close should not raise IOError on closed...	**Closed**	
Related to Ruby - Bug #13158: UNIXServer#closed? returns false after UNIXServ...	**Closed**	
Related to Ruby - Bug #10153: File.open block does not throw "No space left o...	**Closed**	**08/19/2014**

Associated revisions

Revision f9ca64368317a8cdfc59e6faf942030d245fb415 - 04/09/2017 05:09 AM - nobu (Nobuyoshi Nakada)

thread.c: refine stream closed message

- thread.c (Init_Thread): [EXPERIMENTAL] refine the "stream closed" special exception message, by explicating that it is caused by threading. [ruby-core:80583] [Bug #13405]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@58286 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision f9ca6436 - 04/09/2017 05:09 AM - nobu (Nobuyoshi Nakada)

thread.c: refine stream closed message

- thread.c (Init_Thread): [EXPERIMENTAL] refine the "stream closed" special exception message, by explicating that it is caused by threading. [ruby-core:80583] [Bug #13405]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@58286 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 04/06/2017 01:02 AM - shyouhei (Shyouhei Urabe)

matthewd (Matthew Draper) wrote:

> IO#close is supposed to ignore an IOError indicating the stream is already closed.

Is it? I can't find a reason behind this (OK, I know the source code is TRYING to behave that way, but not sure if that is ultimately intended or not).

The proposed fix to ruby_error_closed_stream seems a good catch though. I'd like to +1 regardless of the answer to above question.

#2 - 04/06/2017 02:21 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

"closed stream" and "stream closed" are different.
The former is raised when trying an operation on an IO which has been closed already.
The latter is raised when the IO is closed in an operation by another thread.

#3 - 04/06/2017 03:24 AM - shyouhei (Shyouhei Urabe)

nobu (Nobuyoshi Nakada) wrote:

> "closed stream" and "stream closed" are different.

I then strongly feel that the exception messages are too cryptic. At least "stream closed"'s message shall include that the exception is something thread-related.

#4 - 04/06/2017 07:30 AM - matthewd (Matthew Draper)

matthewd (Matthew Draper) wrote:

> IO#close is supposed to ignore an IOError indicating the stream is already closed.

Sorry, I misread here: io_close ignores the exception, but rb_io_close_m does not. It seems strange that IO.pipe.each(&:close) and IO.pipe {} treat it differently, but that's probably not important right now.

nobu (Nobuyoshi Nakada) wrote:

> "closed stream" and "stream closed" are different.
> The former is raised when trying an operation on an IO which has been closed already.
> The latter is raised when the IO is closed in an operation by another thread.

Setting aside the fact the two messages mean the same thing (ruby_error_closed_stream and test_race_closed_stream even both use the "wrong" word order)... is that distinction important? It tells the caller whether the racing thread ran before we started executing this instruction or after we dropped the GVL to do the requested IO, but I don't see how they can use that information: it appears to just expose them to an implementation detail.

Particularly for close: it doesn't attempt any IO operation if it's already closed. So the only way it can raise either exception is if it wasn't closed at the time we entered rb_io_close_m.

Given that, it seems irrelevant to ruby-land exactly which side of the kernel-level close our race landed on. We wanted it closed, and it's closed; raising an exception due to a concurrent close is inconsistent with #10718.

Maybe the top point is relevant after all, and I'm claiming rb_io_close_m *should* swallow both forms of the exception?

I'm not sure, but this sounds like it might actually be related to #13158, FYI -- the test failures we're seeing are new and relatively frequent: whether that change or another related one, I think they have appeared with the recent batch of releases.

#5 - 04/07/2017 04:29 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #10718: IO#close should not raise IOError on closed IO objects. added

#6 - 04/07/2017 04:29 AM - shyouhei (Shyouhei Urabe)

- Related to Bug #13158: UNIXServer#closed? returns false after UNIXServer#close called added

#7 - 04/07/2017 04:30 AM - shyouhei (Shyouhei Urabe)

- Related to Bug #10153: File.open block does not throw "No space left on device (Errno::ENOSPC)" if the data fits the buffer of IO.write added

#8 - 04/07/2017 04:31 AM - shyouhei (Shyouhei Urabe)

- Status changed from Rejected to Open

Let me reopen.

#9 - 04/09/2017 05:09 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset trunk|r58286.

---

thread.c: refine stream closed message

- thread.c (Init_Thread): [EXPERIMENTAL] refine the "stream closed" special exception message, by explicating that it is caused by threading. [ruby-core:80583] [Bug #13405]