Ruby master - Feature #7220

Separate IO#dup, StringIO#initialize_copy from dup(2)

10/27/2012 03:38 AM - brixen (Brian Shirai)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	3.0

Description

Calling StringIO#initialize_copy causes the two objects to share the position, and eof status.

Is this a bug?

```
sasha:rubinius brian$ irb
1.9.3p286 :001 > require 'stringio'
=> true
1.9.3p286 :002 > a = StringIO.new "abcdefuq"
=> #StringIO:0x00000101016a88
1.9.3p286 :003 > b = StringIO.new
=> #StringIO:0x00000101010728
1.9.3p286 :004 > b.send :initialize_copy, a
=> #StringIO:0x00000101010728
1.9.3p286 :005 > a.pos
=> 0
1.9.3p286 :006 > b.pos
=> 0
1.9.3p286 :007 > b.getc
=> "a"
1.9.3p286 :008 > a.pos
=> 1
1.9.3p286 :009 > a.getc
=> "b"
1.9.3p286 :010 > b.pos
=> 2
1.9.3p286 :011 > b.read
=> "cdefuq"
1.9.3p286 :012 > a.eof?
=> true
```

Thanks,
Brian

History

#1 - 10/27/2012 03:53 AM - brixen (Brian Shirai)

I have found the test for #dup that actually asserts this behavior is correct. Please tell me that is a mistake.

Thanks,
Brian

#2 - 10/27/2012 04:23 AM - headius (Charles Nutter)

I believe, as you supposed on IRC, that this is designed to mimic IO behavior. IO#dup will dup the underlying file descriptor, but they do not maintain their own notion of position into the file; advancing one advances the other. Surely StringIO, as a supposed drop-in for IO, needs to mimic this behavior?

#3 - 10/27/2012 04:42 AM - brixen (Brian Shirai)

If StringIO manifests this behavior as a copy of IO's behavior, both should be consider bugs.

RubySpec has a quarantined spec for this behavior in IO with a comment stating that there are platform incompatibilities with this IO "state aliasing": https://github.com/rubyspec/rubyspec/blob/master/core/io/dup_spec.rb#L34-54

IO#dup creates an instance with a different underlying fd (#fileno): https://github.com/rubyspec/rubyspec/blob/master/core/io/dup_spec.rb#L30-32

There is no other Ruby core class that causes aliasing when calling #dup. String#dup, for example, is called in code precisely to create a new String so the original would not be mutated.

That IO and StringIO do cause this aliasing is a deviation from the typical behavior of #dup, makes no sense, and is not at all required. It's perfectly possible to do the following:

```
sasha:rubinius brian$ cat foobar.txt
123456
sasha:rubinius brian$ irb
1.9.3p286 :001 > a = File.open("foobar.txt", "r")
=> #File:foobar.txt
1.9.3p286 :002 > b = File.open("foobar.txt", "r")
=> #File:foobar.txt
1.9.3p286 :003 > a.getc
=> "1"
1.9.3p286 :004 > a.pos
=> 1
1.9.3p286 :005 > b.pos
=> 0
1.9.3p286 :006 > b.getc
=> "1"
1.9.3p286 :007 > a.fileno
=> 5
1.9.3p286 :008 > b.fileno
=> 6
1.9.3p286 :009 > c = b.dup
=> #File:foobar.txt
1.9.3p286 :010 > c.fileno
=> 7
```

Thanks,
Brian

#4 - 10/27/2012 06:15 AM - drbrain (Eric Hodel)

IO#dup calls dup(2) and StringIO#dup matches this behavior.

Making IO#dup call open(2) would break backwards compatibility.

How do you propose we implement dup(2) for IO?

#5 - 10/27/2012 06:46 AM - brixen (Brian Shirai)

If dup(2) functionality is actually needed, create an IO.system_dup method that return a dup(2) fd and use that fd in IO.new/IO.for_fd.

The fact there there is a dup() system call should not make IO.dup inconsistent with the semantics of every other core #dup method.

#6 - 10/27/2012 06:55 AM - brixen (Brian Shirai)

There's also a dup2() system call. Why don't we provide that one as well?

#7 - 10/27/2012 10:18 AM - drbrain (Eric Hodel)

- Tracker changed from Bug to Feature

- Subject changed from StringIO#initialize_copy causes aliasing between the objects to Separate IO#dup, StringIO#initialize_copy from dup(2)

- Target version set to 3.0

This is intentional behavior which has existed since 1998. It is not a bug.

When I am working with IOs I expect the ruby methods to follow POSIX conventions more than ruby conventions. This method is not the only one in the standard library that doesn't follow ruby conventions.

If you wish to change this behavior you must demonstrate the change will be harmless and easy for existing libraries to adapt to. I don't believe this is the case (due to the drastic change in behavior this would introduce), or that such a change is worthwhile after nearly 15 years.

#8 - 10/27/2012 10:54 AM - normalperson (Eric Wong)

"brixen (Brian Ford)" brixen@gmail.com wrote:

> There's also a dup2() system call. Why don't we provide that one as well?

IO#reopen already provides dup2() (or dup3() if available)

#9 - 10/27/2012 11:23 AM - normalperson (Eric Wong)

"brixen (Brian Ford)" brixen@gmail.com wrote:

> There is no other Ruby core class that causes aliasing when calling #dup. String#dup, for example, is called in code precisely to create a new String so the original would not be mutated.

No other Ruby core class is a delegate for OS-level objects, either.

> That IO and StringIO do cause this aliasing is a deviation from the typical behavior of #dup, makes no sense, and is not at all required. It's perfectly possible to do the following:

It makes sense when I look at it this way:

```
class Foo # this could be a numeric FD or IO object
  attr_reader :array
  def initialize
    @array = [] # the underlying file handle in the kernel
  end
end

a = Foo.new
b = a.dup
p(a.array.object_id == b.array.object_id) # => true
```

> sasha:rubinius brian$ cat foobar.txt
> 123456
> sasha:rubinius brian$ irb
> 1.9.3p286 :001 > a = File.open("foobar.txt", "r")
> => #File:foobar.txt
> 1.9.3p286 :002 > b = File.open("foobar.txt", "r")
> => #File:foobar.txt

It's impossible to implement File#dup using open() reliably. foobar.txt can be unlinked or replaced by an entirely different file in between the File.open calls.

> 1.9.3p286 :003 > a.getc
> => "1"
> 1.9.3p286 :004 > a.pos
> => 1
> 1.9.3p286 :005 > b.pos
> => 0
> 1.9.3p286 :006 > b.getc
> => "1"

However, if you want separate offsets, you can still use dup() but always implement read/write using pread()/pwrite() (and manually maintain per-object offsets in userspace).

#10 - 10/27/2012 12:28 PM - brixen (Brian Shirai)

drbrain (Eric Hodel) wrote:

> This is intentional behavior which has existed since 1998. It is not a bug.
>
> When I am working with IOs I expect the ruby methods to follow POSIX conventions more than ruby conventions. This method is not the only one in the standard library that doesn't follow ruby conventions.
>
> If you wish to change this behavior you must demonstrate the change will be harmless and easy for existing libraries to adapt to. I don't believe this is the case (due to the drastic change in behavior this would introduce), or that such a change is worthwhile after nearly 15 years.

The time it has been implemented is a ridiculous standard. So is requiring it to be easy to adapt to. Nothing that has changed in 1.9 has respected such a standard.

There are serious problems with aliasing the same fd as IO#dup does. It is not consistent across platforms. At the very least, the implementation leaves huge holes, like allowing one fd to be closed, and that IO's #closed? will return true, but the aliasing IO's #closed? will return false, and closing it will raise Errno::EBADF. That behavior shouldn't be behind a common Ruby method or concept.

Further, if StringIO is supposed to mimic all this, there are numerous bugs because StringIO instances that are aliases via #dup do report the same value for eg closed?.

I'm not asking to remove the ability to use dup(2) but that it not be hidden behind Ruby's concept of #dup. I am certain that most Ruby developers understand nothing of the complexities of aliasing a system resource like an fd between different Ruby objects because I've fixed tons of EBADF bugs is RubySpec due to IO#dup. MRI very likely has no tests for such problems.

#11 - 10/27/2012 12:30 PM - brixen (Brian Shirai)

drbrain (Eric Hodel) wrote:

> This is intentional behavior which has existed since 1998. It is not a bug.
>
> When I am working with IOs I expect the ruby methods to follow POSIX conventions more than ruby conventions. This method is not the only one in the standard library that doesn't follow ruby conventions.

Ruby's accidental POSIX semantics are a problem that eg JRuby and any attempt to make Ruby function well on Windows must constantly struggle with. They are not a reason to defend a poor design decision.

#12 - 10/27/2012 02:23 PM - normalperson (Eric Wong)

"brixen (Brian Ford)" brixen@gmail.com wrote:

> There are serious problems with aliasing the same fd as IO#dup does.
> It is not consistent across platforms. At the very least, the
> implementation leaves huge holes, like allowing one fd to be closed,
> and that IO's #closed? will return true, but the aliasing IO's
> #closed? will return false, and closing it will raise Errno::EBADF.
> That behavior shouldn't be behind a common Ruby method or concept.

On *nix, IO#dup creates a new fd, but not a new file handle (inside the kernel). The close() syscall decrements a refcount and only releases the file handle when there are no other references to it.

I don't know non-*nix platforms, but assuming those platforms don't support fork(), either, a non-*nix IO#close/#dup which emulates POSIX semantics could probably work like this:

```
FD_REFCOUNT = [] # fileno => refcount
FD_REFCOUNT_LOCK = Mutex.new

def close
raise IOError if @closed
@closed = true
FD_REFCOUNT_LOCK.synchronize do
FD_REFCOUNT[fileno] -= 1
refcount = FD_REFCOUNT[fileno]

   # call real close() only when refcount is zero
   sysclose(fileno) if refcount == 0
 end

end

def dup
raise IOError if @closed
FD_REFCOUNT_LOCK.synchronize do
FD_REFCOUNT[fileno] += 1
end
super
end
```

This is basically what happens inside the kernel anyways (except the kernel is multi-process-aware). For platforms without fork(), you should be able to emulate POSIX dup()/close() semantics using the example above as a starting point.

> Further, if StringIO is supposed to mimic all this, there are numerous
> bugs because StringIO instances that are aliases via #dup do report
> the same value for eg closed?.

Yeah, StringIO looks buggy here (but I don't expect StringIO to ever

perfectly match IO).