Ruby - Bug #18465

Make `IO#write` atomic.

01/09/2022 06:33 AM - ioquatix (Samuel Williams)

Status:	Closed		
Priority:	Normal		
Assignee:	ioquatix (Samuel Williams)		
Target version:			
ruby -v:		**Backport:**	2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN

Description

Right now, IO#write including everything that calls it including IO#puts, has a poorly specified behaviour w.r.t. other fibers/threads that call IO#write at the same time.

Internally, we have a write lock, however it's only used to lock against individual writes rather than the whole operation. From a user point of view, there is some kind of atomicity, but it's not clearly defined and depends on many factors, e.g. whether write or writev is used internally.

We propose to make IO#write an atomic operation, that is, IO#write on a synchronous/buffered IO will always perform the write operation using a lock around the entire operation.

In theory, this should actually be more efficient than the current approach which may acquire and release the lock several times per operation, however in practice I'm sure it's almost unnoticeable.

Where it does matter, is when interleaved operations invoke the fiber scheduler. By using a single lock around the entire operation, rather than one or more locks around the system calls, the entire operation is more predictable and behaves more robustly.

History

#1 - 01/09/2022 06:34 AM - ioquatix (Samuel Williams)

- Assignee set to ioquatix (Samuel Williams)

As part of working through the implementation, I merged https://github.com/ruby/ruby/pull/5418.

The follow up PR, making io_binwrite and io_binwritev atomic, is here: https://github.com/ruby/ruby/pull/5419.

#2 - 01/09/2022 06:46 AM - ioquatix (Samuel Williams)

I tried to make a micro-benchmark measuring this.

```
> make benchmark ITEM=io_write
/Users/samuel/.rubies/ruby-3.0.3/bin/ruby --disable=gems -rrubygems -I../benchmark/lib ../benchmark/benchmark-driver/exe/benchmark-driver \
            --executables="compare-ruby::/Users/samuel/.rubies/ruby-3.0.3/bin/ruby --disable=gems -I.ext/common --disable-gem" \
            --executables="built-ruby::./miniruby -I../lib -I. -I.ext/common  ../tool/runruby.rb --extout=.ext  -- --disable-gems --disable-gem" \
            --output=markdown --output-compare -v $(find ../benchmark -maxdepth 1 -name 'io_write' -o -name '*io_write*.yml' -o -name '*io_write*.rb' | sort)
compare-ruby: ruby 3.0.3p157 (2021-11-24 revision 3fb7d2cadc) [arm64-darwin21]
built-ruby: ruby 3.2.0dev (2022-01-08T22:41:20Z io-puts-write_lock 079b0c0ee7) [arm64-darwin21]
# Iteration per second (i/s)

|          |compare-ruby|built-ruby|
|:---------|-----------:|---------:|
|io_write  |       5.631|     6.418|
|          |           -|     1.14x|
```

It looks like in this very specific case, it's a little bit faster.

#3 - 01/09/2022 08:01 AM - ioquatix (Samuel Williams)

Even though stderr should not be buffered, I feel like it would be advantageous to use a write lock too, to avoid interleaved log output if possible. Obviously still an issue for multi-process all logging to the same stderr.

#4 - 01/09/2022 01:39 PM - Eregon (Benoit Daloze)

I think it'd better to guarantee atomicity for puts and write, even if the same fd is used by multiple IO instances, and even if the same fd is used by multiple processes.
The write lock does not address both of these cases and it's then leaking implementation details (e.g., other Rubies might not have a write lock).
So that would mean either use writev(2) if available, or concatenate the strings first and then a single write(2).
This is what TruffleRuby does, and it's fully portable.

#5 - 01/09/2022 01:41 PM - Eregon (Benoit Daloze)

Ah and in either case the IO scheduler should either receive the already-concatenated string, or all strings to write together in a single hook call.
Calling the write hook for each argument is of course broken.

#6 - 01/09/2022 08:19 PM - ioquatix (Samuel Williams)

> I think it'd better to guarantee atomicity for puts and write, even if the same fd is used by multiple IO instances, and even if the same fd is used by multiple processes.

The current implementation before and after this PR makes no such guarantee unfortunately. The best you can do as a user is buffer your own string and call write with that as an argument to get any kind of atomic behaviour, but that only applies to non-sync IOs.

> The write lock does not address both of these cases and it's then leaking implementation details (e.g., other Rubies might not have a write lock).

The write lock internally protects sync=true IO which has an internal per-IO buffer. All I've done in my PR is perform the lock once per operation rather than once per system call, so I've moved the lock "up" a little bit to reduce the number of times it would be invoked and increase the amount of synchronisation so that IOs can't interrupt each other.

> This is what TruffleRuby does, and it's fully portable.

https://github.com/oracle/truffleruby/blob/bd36e75003a1f2d57dbc947350cb076e9a827cbd/src/main/ruby/truffleruby/core/io.rb#L2375-L2394

How is sync handled? I don't see the internal buffer is used in def write.

> Ah and in either case the IO scheduler should either receive the already-concatenated string, or all strings to write together in a single hook call.
> Calling the write hook for each argument is of course broken.

The interface for IO#write has to deal with both buffered and un-buffered operations and so we hooked into the internal read and write functions of IO. The best we can hope for is to tidy up how they are used. We don't provide a strong guarantee between one call to IO#write corresponding to one call to Scheduler#io_write... while it's true in most cases, it's not true in all cases. This is really just an artefact of the complexity of the current implementation in io.c.

#7 - 01/10/2022 11:59 AM - Eregon (Benoit Daloze)

ioquatix (Samuel Williams) wrote in #note-6:

> The current implementation before and after this PR makes no such guarantee unfortunately. The best you can do as a user is buffer your own string and call write with that as an argument to get any kind of atomic behaviour, but that only applies to non-sync IOs.

Right, but we could provide this, which is very clearly wanted (nobody wants interleaving between the line and the \n for puts) for puts, if we ensure writev() or all strings are joined before a single write call.
Re sync IOs it'd just work with my approach.

> The write lock internally protects sync=true IO which has an internal per-IO buffer.

Yes, the write lock feels like the wrong primitive to use here, it's the write buffer's lock, and there might not be one for non-buffered IO (or if there is then it's just extra cost).

> https://github.com/oracle/truffleruby/blob/bd36e75003a1f2d57dbc947350cb076e9a827cbd/src/main/ruby/truffleruby/core/io.rb#L2375-L2394

> How is sync handled? I don't see the internal buffer is used in def write.

There is no write buffering in TruffleRuby, we found that:

1. it's not necessary semantically (OTOH it is needed for reading due to ungetc, gets, etc)
2. it doesn't seem to improve performance in most cases, actually it makes worse by having extra copies.
3. better memory footprint due to not having a write buffer per IO

> The interface for IO#write has to deal with both buffered and un-buffered operations and so we hooked into the internal read and write functions of IO. The best we can hope for is to tidy up how they are used. We don't provide a strong guarantee between one call to IO#write corresponding to one call to Scheduler#io_write... while it's true in most cases, it's not true in all cases. This is really just an artefact of the complexity of the current implementation in io.c.

I think doing my suggestion would fix it, for the writev() case you'd either pass all parts to the hook or join before. For the non-writev case it would already be joined.

Any concern however doing the approach in https://bugs.ruby-lang.org/issues/18465#note-4?
It provides the stronger guarantee people actually want (e.g., no interleaving with a subprocess sharing stdout/stderr) and seems to only have benefits.

#8 - 01/10/2022 07:53 PM - ioquatix (Samuel Williams)

Thanks for all that information.

This is a bug fix, but what you are proposing sounds like a feature request.

I want to merge this bug fix, but I think we should consider the direction you propose. I'm happy to raise this point at the developer meeting.

#9 - 01/10/2022 07:54 PM - ioquatix (Samuel Williams)

By the way, even calling write directly is no guarantee of synchronous output between threads/processes - on Linux there is an informal guarantee of page-sized atomicity.

#10 - 01/11/2022 05:24 PM - normalperson (Eric Wong)

"ioquatix (Samuel Williams)" noreply@ruby-lang.org wrote:

> By the way, even calling write directly is no guarantee of
> synchronous output between threads/processes - on Linux there
> is an informal guarantee of page-sized atomicity.

write(2) w/ O_APPEND on local FSes is atomic. Preforked servers
(e.g. Apache) have been relying on that for decades to write log
files.

And PIPE_BUF for pipes/FIFOs is POSIX, (which equals page size on Linux).

> Bug #18465: Make IO#write atomic.
> https://bugs.ruby-lang.org/issues/18465#change-95859

#11 - 01/11/2022 06:23 PM - Eregon (Benoit Daloze)

If the scheduler hook is called under the write lock that sounds like it could cause additional problems including deadlocks, long waits, etc.
So my POV is the fix as originally proposed with the write lock is incomplete and likely to cause more issues as the atomicity is not properly preserved for the Fiber scheduler case.

#12 - 01/13/2022 05:48 AM - ioquatix (Samuel Williams)

I would personally like to simplify IO implementation but I'm not sure if major refactor is acceptable especially given the chance for performance regressions.

You are right, if someone holds a lock when writing to stdout or stderr it in theory could be a problem if the scheduler also tries to re-enter the locked segment. That being said, logging to IO from the scheduler requires an incredibly careful implementation as in theory you can already hold a lock while entering into the scheduler in the current implementation, IIRC. I'd need to check.

#13 - 05/28/2022 02:45 AM - ioquatix (Samuel Williams)

In some cases, Kernel#p can block when calling rb_io_flush if the output is a pipe. Because of that, rb_io_blocking_region can be interrupted while doing flush. This causes the spec test_async_interrupt_and_p to fail, because it expects Kernel#p to be completely uninterruptible, no matter what. I think that's wrong, but we can retain the current behaviour by adding rb_uninterruptible around the rb_io_flush operation too. This retains compatibility, but I'll follow up with a separate issue to make Kernel#p interruptible (like every other IO operation).

#14 - 05/28/2022 04:02 AM - ioquatix (Samuel Williams)

- Status changed from Open to Closed

#15 - 05/28/2022 04:03 AM - ioquatix (Samuel Williams)

Merged changes and follow up issue: https://bugs.ruby-lang.org/issues/18810

#16 - 05/30/2022 03:19 AM - mame (Yusuke Endoh)

I cannot understand this change well. Could you show me a code example to demonstrate this change?

I tried the following code in Ruby 3.1, but I couldn't observe the behavior that A and B are interleaved.

```
a = ["A" * 100_000] * 1000
b = ["B" * 100_000] * 1000

open("output.txt", "w") do |f|
  th = Thread.new { f.write(*a) }
  f.write(*b)
  th.join
end

p File.read("t").gsub(/(.)\1*/) { $1 } #=> "BA" or "AB"
```