Ruby master - Feature #14900

Extra allocation in String#byteslice

07/07/2018 09:47 AM - janko (Janko Marohnić)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

When executing String#byteslice with a range, I noticed that sometimes the original string is allocated again. When I run the following script:

```
require "objspace"

string = "a" * 100_000

GC.start
GC.disable
generation = GC.count

ObjectSpace.trace_object_allocations do
  string.byteslice(50_000..-1)

  ObjectSpace.each_object(String) do |string|
    p string.bytesize if ObjectSpace.allocation_generation(string) == generation
  end
end
```

it outputs

```
50000
100000
6
5
```

The one with 50000 bytes is the result of String#byteslice, but the one with 100000 bytes is the duplicated original string. I expected only the result of String#byteslice to be amongst new allocations.

If instead of the last 50000 bytes I slice the *first* 50000 bytes, the extra duplication doesn't occur.

```
# ...
  string.byteslice(0, 50_000)
# ...

50000
5
```

It's definitely ok if the implementation of String#bytesize allocates extra strings as part of the implementation, but it would be nice if they were deallocated before returning the result.

EDIT: It seems that String#slice has the same issue.

History

#1 - 07/07/2018 09:53 AM - janko (Janko Marohnić)

- Description updated

#2 - 07/07/2018 08:48 PM - ioquatix (Samuel Williams)

Nice catch I will try to verify on my end too

#3 - 07/08/2018 02:52 AM - ioquatix (Samuel Williams)

Okay, I reproduced the error. I made a test case here:

https://github.com/ioquatix/ruby/commit/9fb5cd644209efc79378841e1b6eb644876393b0

I test both prefix and postfix as you discuss in your initial report.

#4 - 07/08/2018 02:52 AM - ioquatix (Samuel Williams)

One thing I noticed if I freeze source string, the extra memory allocation goes away.

#5 - 07/08/2018 03:22 AM - ioquatix (Samuel Williams)

Okay I made an attempt to fix this: https://github.com/ruby/ruby/pull/1909

#6 - 07/08/2018 03:31 AM - ioquatix (Samuel Williams)

I think there are several things to consider here:

- Even though the string appears to be two allocations, it's only one allocation but the 2nd one is sharing the first's data.
- I guess that subsequent slice would share the underling frozen string?
- In some cases, byteslice might be less efficient, e.g. 100Mbyte buffer, slice the last 10bytes, it makes an entire copy of the source string, but all you were interested in was 10 bytes at the end.

#7 - 07/08/2018 09:46 AM - funny_falcon (Yura Sokolov)

ioquatix (Samuel Williams), your patch doesn't seems to be correct for me on first glance.

Imagine pipelined RPC server:

- we read data into buffer
- while buffer larger than request size
  - detect first request and split buffer into request and rest of buffer

Same for any other binary parser.

With current behavior, operation "get rest of buffer" will copy buffer into shared frozen string only once.
With your patch it will copy every time.
So instead on linear complexity we will have quadratic complexity.

Thinking second time, it is possible to use frozen string explicitely for buffer, just not so trivial (while there are not enough data for request, buffer should not frozen, and << should be used, otherwise it should be frozen, and + used).

Some programs will certainly become slower with this change, until they fixed.

I'm not against the patch, but new behavior should be carefully documented and mentioned in a Changelog as a change, that could negatively affect performance if not concerned.

#8 - 07/08/2018 09:47 AM - ioquatix (Samuel Williams)

Yeah, I agree, this patch probably isn't right, but I just try to figure it out what is going on and suggest a solution. The outcome may be that this is normal behaviour. Thanks for your feedback.

#9 - 07/08/2018 10:45 AM - ioquatix (Samuel Williams)

The way I've implemented it now (as in your first example) is something like this:

```
@buffer = read_data
if @buffer.bytesize > REQUEST_SIZE
    @buffer.freeze
    request_buffer = @buffer.byteslice(0, REQUEST_SIZE)
    @buffer = @buffer.byteslice(REQUEST_SIZE, @buffer.bytesize)
end
```

Because we will recreate @buffer from remainder, it makes sense to freeze the source to avoid generating a hidden copy. Does that make sense?

I believe if we were to propose an implementation of byteslice! it would look like the above.

#10 - 07/08/2018 11:35 AM - ioquatix (Samuel Williams)

I played around with my assumptions here. By far the worst from a memory POV was slice!, which given a string of 5MB, produces 7.5MB allocations. The equivalent sequence of byteslice as above only allocates 2.5MB.

Here were my comparisons:

```
measure_memory("Initial allocation") do
```

```
    string = "a" * 5*1024*1024
    string.freeze
end # => 5.0 MB

measure_memory("Byteslice from start to middle") do
    # Why does this need to allocate memory? Surely it can share the original allocation?
    x = string.byteslice(0, string.bytesize / 2)
end # => 2.5 MB

measure_memory("Byteslice from middle to end") do
    string.byteslice(string.bytesize / 2, string.bytesize)
end # => 0.0 MB

measure_memory("Slice! from start to middle") do
    string.dup.slice!(0, string.bytesize / 2) # dup doesn't make any difference to size of allocations
end # => 7.5 MB

measure_memory("Byte slice into two halves") do
    head = string.byteslice(0, string.bytesize / 2)
    remainder = string.byteslice(string.bytesize / 2, string.bytesize)
end # 2.5 MB
```

(examples are also here: https://github.com/socketry/async-io/blob/master/examples/allocations/byteslice.rb)

In the best case, the last example should be able to reuse the source string entirely, but Ruby doesn't seem capable of doing that yet. Perhaps a specific implementation of byteslice! could address this use case with zero allocations?

#11 - 08/31/2020 09:41 PM - jeremyevans0 (Jeremy Evans)

- Backport deleted (2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN)

- ruby -v deleted (ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-darwin17])

- Tracker changed from Bug to Feature