Ruby master - Bug #14275

GC not aggressive enough

01/03/2018 01:52 AM - normalperson (Eric Wong)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

Ruby memory usage has always been a thorn in my side, more so than (lack of)
speed. I sometimes take vacations into Perl5 or C to escape from it.

Sometimes a well-placed String#clear has a good effect, but I'm not sure
if it's acceptable for Rubyists to sprinkle throughout their code.

I've made some changes to net/http for 2.5 to improve the situation:

https://svn.ruby-lang.org/cgi-bin/viewvc.cgi?view=revision&revision=58840
https://svn.ruby-lang.org/cgi-bin/viewvc.cgi?view=revision&revision=58846

And proposed one more for 2.6: https://bugs.ruby-lang.org/issues/14268

But a String#clear is still necessary for the end user. Even
with my patch for [Feature #14268], that example takes around
10x more memory without String#clear.

Can Ruby be better out-of-the box?

The attached script takes about 120MB on my system (without String#clear)

Lazy sweep seems to hurt in this case, because large (16K)
buffers are floating around unused with delayed free(3).
Sweeping immediately for malloc increase (and not other GC
trigger points) reduces memory from around 120 MB to 75 MB.

```
--- a/gc.c
+++ b/gc.c
@@ -7825,7 +7825,7 @@ objspace_malloc_increase(rb_objspace_t *objspace, void *mem, size_t new_size
, si
         gc_rest(objspace); /* gc_rest can reduce malloc_increase */
         goto retry;
         }
-        garbage_collect_with_gvl(objspace, FALSE, FALSE, FALSE, GPR_FLAG_MALLOC);
+        garbage_collect_with_gvl(objspace, FALSE, FALSE, TRUE, GPR_FLAG_MALLOC);
     }
      }
```

Maybe more could be done and less intrusively. Providing a
custom custom malloc (not just wrapper API) would allow us to
finish sweeping before calls to mmap(2)/sbrk(2) are necessary to
request memory from the kernel. That might be a maintenance
nightmare...

History

#1 - 01/03/2018 09:18 AM - shevegen (Robert A. Heiler)

> Sometimes a well-placed String#clear has a good effect, but
> I'm not sure if it's acceptable for Rubyists to sprinkle

> throughout their code.

I myself have been slowly starting to make use of .dup more since I also tend to use frozen string set to true.

While my ruby code "appears" to be faster than before, probably also because ruby itself became faster as well - and as matz once said, nobody minds if ruby becomes faster - I still don't really like .dup much.

Code that would have been before like:

```
_ = ''
```

would then be:

```
_ = ''.dup
```

And even though it may be faster to have all Strings mutable by default, I simply think that the first variant is prettier to read.

If you'd then add:

```
_.clear
```

just for performance/speed/memory gain, and add these where it may matter, it sorta feels as if ruby transmutes into a language where the higher expression suddenly becomes intermingled with method calls or syntax that feels less like ruby. It's an impression I have even more with crystal; while the syntax is pretty much ruby, the type system just makes for a different language IMO.

By the way, the above applies to my own code; I have no problem at all if stdlib becomes faster. I don't have to read or modify its source, so it's fine by me. :)

But I personally, in my code, I'd rather prefer to keep the code succinct and pretty when possible.

> But a String#clear is still necessary for the end user.

There is an additional problem. Not only an extra line, but also that the user is then required to know about these things. And I am not sure that this should be the way to go. I think matz once said this about some ruby code in the rails ecosystem surprising him how people use ruby but I am not sure if I remember this correctly since it has been so many years ago; it may have just been about people making more use of symbols back then.

Files

net-http-readbody.rb	642 Bytes	01/03/2018	normalperson (Eric Wong)