Ruby master - Feature #5875

Couple of tiny changes to string

01/10/2012 10:00 PM - funny_falcon (Yura Sokolov)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:	2.0.0	

Description

- change capacity increment from (capa + 1) * 2 to capa * 2 + 1 previous increment formula leads to inconvenient allocation patterns: 25bytes, 51bytes, etc new formula leads to more comfortable allocation pattern: 24b, 48b, 96b
- change STR_BUF_MIN_SIZE from 128 to 79 128 leads to allocation of 129 bytes, which is very uncomfortable for allocators and unnecessary large. (during Redmine startup this method is called about 3000000 times with capa < 128)

https://github.com/ruby/ruby/pull/80
https://github.com/funny-falcon/ruby/commit/2240a04d49118d9fa6f038655dac27f0ad96ed6b.patch

Associated revisions

Revision ed5401a6 - 07/14/2016 11:30 PM - normal

string.c: reduce malloc overhead for default buffer size

- string.c (STR_BUF_MIN_SIZE): reduce from 128 to 127 [ruby-core:76371] [Feature #12025]
- string.c (rb_str_buf_new): adjust for above reduction

From Jeremy Evans code@jeremyevans.net:

This changes the minimum buffer size for string buffers from 128 to

1. The underlying C buffer is always 1 more than the ruby buffer, so this changes the actual amount of memory used for the minimum string buffer from 129 to 128. This makes it much easier on the malloc implementation, as evidenced by the following code (note that time -l is used here, but Linux systems may need time -v).

```
$ cat bench_mem.rb
i = ARGV.first.to_i
Array.new(1000000){" " * i}
$ /usr/bin/time -l ruby bench_mem.rb 128
3.10 real         2.19 user         0.46 sys
289080  maximum resident set size
72673  minor page faults
13  block output operations
29  voluntary context switches
$ /usr/bin/time -l ruby bench_mem.rb 127
2.64 real         2.09 user         0.27 sys
162720  maximum resident set size
40966  minor page faults
2  block output operations
4  voluntary context switches
```

To try to ensure a power-of-2 growth, when a ruby string capacity
needs to be increased, after doubling the capacity, add one. This
ensures the ruby capacity will be odd, which means actual amount
of memory used will be even, which is probably better than the
current case of the ruby capacity being even and the actual amount
of memory used being odd.

A very similar patch was proposed 4 years ago in feature #5875. It
ended up being rejected, because no performance increase was shown.
One reason for that is that ruby does not use STR_BUF_MIN_SIZE
unless rb_str_buf_new is called, and that previously did not have
a ruby API, only a C API, so unless you were using a C extension
that called it, there would be no performance increase.

With the recently proposed feature #12024, String.buffer is added,
which is a ruby API for creating string buffers. Using
String.buffer(100) wastes much less memory with this patch, as the

malloc implementation can more easily deal with the power-of-2
sized memory usage. As measured above, memory usage is 44% less,
and performance is 17% better.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55686 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 55686 - 07/14/2016 11:30 PM - normalperson (Eric Wong)

string.c: reduce malloc overhead for default buffer size

- string.c (STR_BUF_MIN_SIZE): reduce from 128 to 127 [ruby-core:76371] [Feature #12025]
- string.c (rb_str_buf_new): adjust for above reduction

From Jeremy Evans code@jeremyevans.net:

This changes the minimum buffer size for string buffers from 128 to

1. The underlying C buffer is always 1 more than the ruby buffer, so this changes the actual amount of memory used for the minimum string buffer from 129 to 128. This makes it much easier on the malloc implementation, as evidenced by the following code (note that time -l is used here, but Linux systems may need time -v).

```
$ cat bench_mem.rb
i = ARGV.first.to_i
Array.new(1000000){" " * i}
$ /usr/bin/time -l ruby bench_mem.rb 128
3.10 real         2.19 user         0.46 sys
289080  maximum resident set size
72673  minor page faults
13  block output operations
29  voluntary context switches
$ /usr/bin/time -l ruby bench_mem.rb 127
2.64 real         2.09 user         0.27 sys
162720  maximum resident set size
40966  minor page faults
2  block output operations
4  voluntary context switches
```

To try to ensure a power-of-2 growth, when a ruby string capacity
needs to be increased, after doubling the capacity, add one. This
ensures the ruby capacity will be odd, which means actual amount
of memory used will be even, which is probably better than the
current case of the ruby capacity being even and the actual amount
of memory used being odd.

A very similar patch was proposed 4 years ago in feature #5875. It
ended up being rejected, because no performance increase was shown.
One reason for that is that ruby does not use STR_BUF_MIN_SIZE
unless rb_str_buf_new is called, and that previously did not have
a ruby API, only a C API, so unless you were using a C extension
that called it, there would be no performance increase.

With the recently proposed feature #12024, String.buffer is added,
which is a ruby API for creating string buffers. Using
String.buffer(100) wastes much less memory with this patch, as the
malloc implementation can more easily deal with the power-of-2
sized memory usage. As measured above, memory usage is 44% less,
and performance is 17% better.

Revision 55686 - 07/14/2016 11:30 PM - normal

string.c: reduce malloc overhead for default buffer size

- string.c (STR_BUF_MIN_SIZE): reduce from 128 to 127 [ruby-core:76371] [Feature #12025]
- string.c (rb_str_buf_new): adjust for above reduction

From Jeremy Evans code@jeremyevans.net:

This changes the minimum buffer size for string buffers from 128 to

1. The underlying C buffer is always 1 more than the ruby buffer, so this changes the actual amount of memory used for the minimum string buffer from 129 to 128. This makes it much easier on the malloc implementation, as evidenced by the following code (note that time -l is used here, but Linux systems may need time -v).

```
$ cat bench_mem.rb
i = ARGV.first.to_i
```

```
Array.new(1000000){" " * i}
$ /usr/bin/time -l ruby bench_mem.rb 128
3.10 real         2.19 user         0.46 sys
289080  maximum resident set size
72673  minor page faults
13  block output operations
29  voluntary context switches
$ /usr/bin/time -l ruby bench_mem.rb 127
2.64 real         2.09 user         0.27 sys
162720  maximum resident set size
40966  minor page faults
2  block output operations
4  voluntary context switches
```

To try to ensure a power-of-2 growth, when a ruby string capacity
needs to be increased, after doubling the capacity, add one. This
ensures the ruby capacity will be odd, which means actual amount
of memory used will be even, which is probably better than the
current case of the ruby capacity being even and the actual amount
of memory used being odd.

A very similar patch was proposed 4 years ago in feature #5875. It
ended up being rejected, because no performance increase was shown.
One reason for that is that ruby does not use STR_BUF_MIN_SIZE
unless rb_str_buf_new is called, and that previously did not have
a ruby API, only a C API, so unless you were using a C extension
that called it, there would be no performance increase.

With the recently proposed feature #12024, String.buffer is added,
which is a ruby API for creating string buffers. Using
String.buffer(100) wastes much less memory with this patch, as the
malloc implementation can more easily deal with the power-of-2
sized memory usage. As measured above, memory usage is 44% less,
and performance is 17% better.

Revision 55686 - 07/14/2016 11:30 PM - normal

string.c: reduce malloc overhead for default buffer size

- string.c (STR_BUF_MIN_SIZE): reduce from 128 to 127 [ruby-core:76371] [Feature #12025]
- string.c (rb_str_buf_new): adjust for above reduction

From Jeremy Evans code@jeremyevans.net:

This changes the minimum buffer size for string buffers from 128 to

1. The underlying C buffer is always 1 more than the ruby buffer, so this changes the actual amount of memory used for the minimum string buffer from 129 to 128. This makes it much easier on the malloc implementation, as evidenced by the following code (note that time -l is used here, but Linux systems may need time -v).

```
$ cat bench_mem.rb
i = ARGV.first.to_i
Array.new(1000000){" " * i}
$ /usr/bin/time -l ruby bench_mem.rb 128
3.10 real         2.19 user         0.46 sys
289080  maximum resident set size
72673  minor page faults
13  block output operations
29  voluntary context switches
$ /usr/bin/time -l ruby bench_mem.rb 127
2.64 real         2.09 user         0.27 sys
162720  maximum resident set size
40966  minor page faults
2  block output operations
4  voluntary context switches
```

To try to ensure a power-of-2 growth, when a ruby string capacity
needs to be increased, after doubling the capacity, add one. This
ensures the ruby capacity will be odd, which means actual amount
of memory used will be even, which is probably better than the
current case of the ruby capacity being even and the actual amount
of memory used being odd.

A very similar patch was proposed 4 years ago in feature #5875. It
ended up being rejected, because no performance increase was shown.

One reason for that is that ruby does not use STR_BUF_MIN_SIZE
unless rb_str_buf_new is called, and that previously did not have
a ruby API, only a C API, so unless you were using a C extension
that called it, there would be no performance increase.

With the recently proposed feature #12024, String.buffer is added,
which is a ruby API for creating string buffers. Using
String.buffer(100) wastes much less memory with this patch, as the
malloc implementation can more easily deal with the power-of-2
sized memory usage. As measured above, memory usage is 44% less,
and performance is 17% better.

Revision 55686 - 07/14/2016 11:30 PM - normal

string.c: reduce malloc overhead for default buffer size

- string.c (STR_BUF_MIN_SIZE): reduce from 128 to 127 [ruby-core:76371] [Feature #12025]
- string.c (rb_str_buf_new): adjust for above reduction

From Jeremy Evans code@jeremyevans.net:

This changes the minimum buffer size for string buffers from 128 to

1. The underlying C buffer is always 1 more than the ruby buffer, so this changes the actual amount of memory used for the minimum string buffer from 129 to 128. This makes it much easier on the malloc implementation, as evidenced by the following code (note that time -l is used here, but Linux systems may need time -v).

```
$ cat bench_mem.rb
i = ARGV.first.to_i
Array.new(1000000){" " * i}
$ /usr/bin/time -l ruby bench_mem.rb 128
3.10 real         2.19 user         0.46 sys
289080  maximum resident set size
72673  minor page faults
13  block output operations
29  voluntary context switches
$ /usr/bin/time -l ruby bench_mem.rb 127
2.64 real         2.09 user         0.27 sys
162720  maximum resident set size
40966  minor page faults
2  block output operations
4  voluntary context switches
```

To try to ensure a power-of-2 growth, when a ruby string capacity
needs to be increased, after doubling the capacity, add one. This
ensures the ruby capacity will be odd, which means actual amount
of memory used will be even, which is probably better than the
current case of the ruby capacity being even and the actual amount
of memory used being odd.

A very similar patch was proposed 4 years ago in feature #5875. It
ended up being rejected, because no performance increase was shown.
One reason for that is that ruby does not use STR_BUF_MIN_SIZE
unless rb_str_buf_new is called, and that previously did not have
a ruby API, only a C API, so unless you were using a C extension
that called it, there would be no performance increase.

With the recently proposed feature #12024, String.buffer is added,
which is a ruby API for creating string buffers. Using
String.buffer(100) wastes much less memory with this patch, as the
malloc implementation can more easily deal with the power-of-2
sized memory usage. As measured above, memory usage is 44% less,
and performance is 17% better.

History

#1 - 01/12/2012 02:46 AM - kosaki (Motohiro KOSAKI)

- Status changed from Open to Closed

Closed by reporter's request.