Ruby trunk - Feature #15631

Let round_capa for ID table not allocate excess capacity for power of 2 ints >= 4

03/01/2019 01:48 PM - ahorek (Pavel Rosický)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

right now round_capa value is rounded up to the next power of 2

```
round_capa(4) -> returns 8
round_capa(8) -> returns 16
round_capa(16) -> returns 32

round_capa(5) -> returns 8
round_capa(9) -> returns 16
round_capa(17) -> returns 32
etc.
```

it seems wasteful to allocate the extra items capacity, so this PR changes that to

```
round_capa(4) -> returns 4
round_capa(8) -> returns 8
round_capa(16) -> returns 16

round_capa(5) -> returns 8
round_capa(9) -> returns 16
round_capa(17) -> returns 32
etc.
```

the main purpose is to reduce memory usage especially during boot

my patch also uses BUILTIN_CLZ macro instead of shifts that makes it slightly faster

here's a benchmark

```
require 'benchmark/ips'

Benchmark.ips do |x|
  x.config(time: 20, warmup: 3)

  x.report('struct', "Struct.new(*('a'..'z').map { |x| x.to_sym })")
end

trunk
Warming up --------------------------------------
              struct   527.000  i/100ms
Calculating -------------------------------------
              struct      5.461k (± 5.5%) i/s -    109.089k in  20.040253s

methodmising - POW2_P (github)
Warming up --------------------------------------
              struct   544.000  i/100ms
Calculating -------------------------------------
              struct      5.570k (± 4.1%) i/s -    111.520k in  20.057245s

ahorek - BUILTIN_CLZ (id_table.c.patch)
Warming up --------------------------------------
              struct   571.000  i/100ms
Calculating -------------------------------------
              struct      5.812k (± 3.6%) i/s -    116.484k in  20.070607s
```

discussion https://github.com/ruby/ruby/pull/2083

History

#1 - 03/05/2019 07:57 PM - ahorek (Pavel Rosický)

- Description updated

#2 - 03/11/2019 10:49 AM - methodmissing (Lourens Naudé)

Thanks for raising this Pavel.

st_init_table_with_size(0) effectively also allocates additional capacity, but if and how quickly the hash tables mutate I'll investigate later.

References https://github.com/ruby/ruby/blob/trunk/st.c#L573-L578 , https://github.com/ruby/ruby/blob/trunk/st.c#L595 and https://github.com/ruby/ruby/blob/trunk/st.c#L332-L359

A simple peek suggests a total table size of 152 bytes on init, but will investigate time to mutation of these 0 sized tables this evening:

```
diff --git a/st.c b/st.c
index ed235c674e..f2b99d7771 100644
--- a/st.c
+++ b/st.c
@@ -615,6 +615,8 @@ st_init_table_with_size(const struct st_hash_type *type, st_index_t size)
 #ifdef ST_DEBUG
     st_check(tab);
 #endif
+    printf("# st_init_table_with_size(%d) -> %d (%d)\n", size, n, st_memsize(tab));
+
     return tab;
 }
```

```
linking miniruby
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(102) -> 7 (3384)
# st_init_table_with_size(255) -> 8 (7224)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(1000) -> 10 (28728)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(1000) -> 10 (28728)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(7) -> 3 (248)
# st_init_table_with_size(15) -> 4 (440)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(16) -> 5 (888)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
```

```
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
# st_init_table_with_size(0) -> 2 (152)
```

ahorek (Pavel Rosický) wrote:

> right now round_capa value is rounded up to the next power of 2
>
> ```
> round_capa(4) -> returns 8
> round_capa(8) -> returns 16
> round_capa(16) -> returns 32
>
> round_capa(5) -> returns 8
> round_capa(9) -> returns 16
> round_capa(17) -> returns 32
> etc.
> ```
>
> it seems wasteful to allocate the extra items capacity, so this PR changes that to
>
> ```
> round_capa(4) -> returns 4
> round_capa(8) -> returns 8
> round_capa(16) -> returns 16
>
> round_capa(5) -> returns 8
> round_capa(9) -> returns 16
> round_capa(17) -> returns 32
> etc.
> ```
>
> the main purpose is to reduce memory usage especially during boot
>
> my patch also uses BUILTIN_CLZ macro instead of shifts that makes it slightly faster
>
> here's a benchmark
>
> ```
> require 'benchmark/ips'
>
> Benchmark.ips do |x|
>   x.config(time: 20, warmup: 3)
>
>   x.report('struct', "Struct.new(*('a'..'z').map { |x| x.to_sym })")
> end
>
> trunk
> Warming up --------------------------------------
>                struct   527.000  i/100ms
> Calculating -------------------------------------
>                struct      5.461k (± 5.5%) i/s -    109.089k in  20.040253s
>
> methodmising - POW2_P (github)
> Warming up --------------------------------------
>                struct   544.000  i/100ms
> Calculating -------------------------------------
>                struct      5.570k (± 4.1%) i/s -    111.520k in  20.057245s
>
> ahorek - BUILTIN_CLZ (id_table.c.patch)
> Warming up --------------------------------------
>                struct   571.000  i/100ms
> Calculating -------------------------------------
>                struct      5.812k (± 3.6%) i/s -    116.484k in  20.070607s
> ```
>
> discussion https://github.com/ruby/ruby/pull/2083

Files

id_table.c.patch	534 Bytes	03/01/2019	ahorek (Pavel Rosický)