Ruby master - Bug #17583
Segfault on large stack(RUBY_THREAD_VM_STACK_SIZE)
01/26/2021 08:55 AM - yoshiokatsuneo (Tsuneo Yoshioka)

Status: Closed
Priority: Normal
Assignee: ko1 (Koichi Sasada)
Target version: ruby -v:
ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-linux]

| Backport | 2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN |

Description
When I set large stack size like 100MB using RUBY_THREAD_VM_STACK_SIZE,
I get the segmentation fault.
The error looks happens when RUBY_THREAD_VM_STACK_SIZE is larger than around 17MB.

How to reproduce:
```
$ RUBY_THREAD_VM_STACK_SIZE=17000000 ruby -e ''
```

Output:
```
# RUBY_THREAD_VM_STACK_SIZE=17000000 ruby -e ''
/usr/local/rbenv/versions/3.0.0/lib/ruby/3.0.0/x86_64-linux/enc/encdb.so: [BUG] Segmentation fault
at 0x0000000000000048
ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-linux]
```

-- Control frame information ---------------------------------------------
c:0002 p:-972860 s:0006 e:000005 TOP [FINISH]
c:0001 p:0000 s:0003 E:0008e0 (none) [FINISH]

-- Machine register context ---------------------------------------------
RIP: 0x00007f9a84541143 RBP: 0x0000000000000051 RSP: 0x00007ffca756efe8
RAX: 0x0000000000000000 RBX: 0x000000000070bf98 RCX: 0x000000000000080c
RDX: 0x0000000000000000 RDI: 0x0000000000000000 RSI: 0x0000000000000000
R8: 0x0000000000000000 R9: 0x0000000000000000 R10: 0x0000000000000000
R11: 0x3f3c0feefeb77f71 R12: 0x0000000000000000 R13: 0x0000000000000000
R14: 0x0000000000000000 R15: 0x00000000000000f3 EFL: 0x0000000000010202

-- C level backtrace information ----------------------------------------
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(rb_print_backtrace+0x14) [0x7f9a8478c33f] vm_dump.c:758
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(rb_vm_bugreport) vm_dump.c:998
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(bug_report_end+0x0) [0x7f9a845b0d73] error.c:786
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(rb_bug_for_fatal_signal) error.c:786
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(sigsev+0x51) [0x7f9a846e5e71] signal.c:960
/lib/x86_64-linux-gnu/libpthread.so.0(_restore_rt+0x0) [0x7f9a844933c0] ../sysdeps/threads/funlocfile.c:28
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(rb_freeze_singleton_class+0x13) [0x7f9a84541143 ] class.c:1865
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(rb_obj_freeze_inline+0x23) [0x7f9a84651934] ./include/ruby/internal/fl_type.h:466
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(rb_obj_freeze) object.c:1281
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(rbimpl_fl_set_raw_raw+0x0) [0x7f9a845999ae] encoding.c:125
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(RB_FL_SET_RAW) ./include/ruby/internal/fl_type.h:298
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(enc_new) encoding.c:126
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(enc_list_update) encoding.c:137
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(enc_register_at) encoding.c:392
/usr/local/rbenv/versions/3.0.0/lib/libruby.so.3.0(rb_encdb_replicate+0xc9bd8) [0x7f9a8459a05]
I confirmed that the problem happens on both Ubuntu 20.04 and macOS(Big Sur/11.1).

Related issues:
Has duplicate Ruby master - Bug #17668: Large RUBY_THREAD_VM_STACK_SIZE cause...
Closed

Revision 8ebb5e23 - 07/02/2021 01:49 AM - nobu (Nobuyoshi Nakada)
Keep GC disabled until VM bootstrap has done [Bug #17583]

Associated revisions
Revision 8ebb5e23 - 07/02/2021 01:49 AM - nobu (Nobuyoshi Nakada)
Keep GC disabled until VM bootstrap has done [Bug #17583]

Associated revisions
Revision 8ebb5e23 - 07/02/2021 01:49 AM - nobu (Nobuyoshi Nakada)
Keep GC disabled until VM bootstrap has done [Bug #17583]

Associated revisions
Revision 8ebb5e23 - 07/02/2021 01:49 AM - nobu (Nobuyoshi Nakada)
Keep GC disabled until VM bootstrap has done [Bug #17583]

History
#1 - 01/26/2021 08:56 AM - yoshiokatsuneo (Tsuneo Yoshioka)
- ruby -v changed from 3.0.0 to ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-linux]

#2 - 01/26/2021 09:04 AM - yoshiokatsuneo (Tsuneo Yoshioka)
- Description updated

#3 - 01/26/2021 09:04 AM - yoshiokatsuneo (Tsuneo Yoshioka)
- Description updated

#4 - 01/26/2021 03:35 PM - nobu (Nobuyoshi Nakada)
Seems mark_object_ary broken?
This should be the array of arrays.

```ruby
* thread #1, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1, address=0x10)
  frame #0: 0x00000001007698a9  miniruby`lookup_method_table(klass=108645493675760, id=3281) at vm_method.c:6
9933
  696  lookup_method_table(VALUE klass, ID id)
  697  {
  698     st_data_t body;
--> 699     struct rb_id_table *m_tbl = RCLASS_M_TBL(klass);
  700    
  701    if (rb_id_table_lookup(m_tbl, id, &body)) {
  702      return (rb_method_entry_t *) body;
  703    }
Target 0: (miniruby) stopped.
(lldb) p ruby_current_vm_ptr->mark_object_ary
(VALUE) $0 = 108645493686200
(lldb) rp ruby_current_vm_ptr->mark_object_ary
bits [ ]
T_ARRAY: len=1 (ownership) capa=128
(const VALUE *) $2 = 0x00006190000ebf68 {
  (const VALUE) [0] = 0x000062d0000ebf68 }
(lldb) rp $2[0]
bits [ ]
T_STRING: (const char [73]) $4 = " def self.verify_compaction_references(toward: nil, double_heap: false)"
```

#5 - 01/28/2021 03:43 PM - xtkoba (Tee KOBAYASHI)
The crash threshold of RUBY_THREAD_VM_STACK_SIZE seems to coincide with the default value of GC_MALLOC_LIMIT_MIN (which is 16MB).
I think something is messed up by GC in an early stage of initialization. In fact, the crash seems to disappear when the function garbage_collect is made to do nothing other than returning TRUE.

#6 - 01/29/2021 06:31 PM - ko1 (Koichi Sasada)
- Assignee set to ko1 (Koichi Sasada)
- Status changed from Open to Assigned

#7 - 03/02/2021 09:18 AM - xtkoba (Tee KOBAYASHI)
Bug #17668 seems to be a duplicate of this bug.
As I wrote in #note-5, this issue seems to be caused by running GC during initialization. If so, a straightforward workaround would be to suppress GC during init:

```
--- a/eval.c
+++ b/eval.c
@@ -99,6 +99,8 @@
  return state;
```
extern int suppress_garbage_collection;
+
*/!
* Calls ruby_setup() and check error.*
*
@@ -107,6 +109,7 @@
  void
ruby_init(void)
{
+    suppress_garbage_collection = 1;
    int state = ruby_setup();
    if (state) {
        if (RTEST(ruby_debug))
@@ -365,6 +368,7 @@
    int
ruby_run_node(void *n)
{
+    suppress_garbage_collection = 0;
    rb_execution_context_t *ec = GET_EC();
    int status;
    if (!ruby_executable_node(n, &status)) { --- a/gc.c
+++ b/gc.c
@@ -8214,11 +8214,15 @@
#define GC_ENABLE_LAZY_SWEEP to 0, but this is not good because it disables lazy sweeping throughout the entire lifecycle. It would be better if we could suppress lazy sweeping during initialization only.

#8 - 03/03/2021 10:45 PM - jeremyevans0 (Jeremy Evans)
- Has duplicate Bug #17668: Large RUBY_THREAD_VM_STACK_SIZE causes segmentation fault (again) added

#9 - 03/09/2021 09:10 AM - nobu (Nobuyoshi Nakada)
Does this work?
https://github.com/nobu/ruby/tree/disable-gc-while-VM-bootstrap

#10 - 04/28/2021 11:52 PM - jeremyevans0 (Jeremy Evans)
nobu (Nobuyoshi Nakada) wrote in #note-9:

Does this work?
https://github.com/nobu/ruby/tree/disable-gc-while-VM-bootstrap

nobu (Nobuyoshi Nakada) I tested it and it fixes the problem on OpenBSD/amd64.

#11 - 07/01/2021 07:34 PM - jeremyevans0 (Jeremy Evans)
nobu (Nobuyoshi Nakada) wrote in #note-9:

Does this work?
https://github.com/nobu/ruby/tree/disable-gc-while-VM-bootstrap

Since it worked for me, I rebased this tree on master and submitted a pull request for it: https://github.com/ruby/ruby/pull/4617

#12 - 07/02/2021 01:50 AM - nobu (Nobuyoshi Nakada)
- Status changed from Assigned to Closed
Keep GC disabled until VM bootstrap has done [Bug #17583]

#13 - 07/31/2021 01:02 AM - ciel (T Yamada)

Hi, could you backport this to Ruby 3.0?