Ruby master - Bug #3940

Lazy sweep and ObjectSpace.each_object

10/13/2010 02:14 PM - ko1 (Koichi Sasada)

Status:	Closed		
Priority:	Normal		
Assignee:	authorNari (Narihiro Nakamura)		
Target version:	1.9.3		
ruby -v:	ruby 1.9.3dev (2010-10-13 trunk 29477)	**Backport:**	

Description

=begin
　ささだです。

　Lazy sweep 中に ObjectSpace.each_object (相当) を行うと、SEGV するので、下記のようなパッチを作成しました。

32bit 環境の test-all で、最近 sdbm で SEGV してたのは多分これが問題。
ObjectSpace.each_object を使っているので。

　推測ですが、下記のような原因ではないかと思います。

(1) GC mark 終了時、オブジェクト o1 と、そこから参照される o2 が
mark されず、sweep 対象になる
(2) lazy sweep により、o2 が回収される
(3) ObjectSpace.each_object により、o1 へアクセスする
(4) o1 から o2 を参照しようとして SEGV

　解決策として、中田さんに ObjectSpace.each_object に相当する関数
rb_objspace_each_objects を実行する前に sweep を完全に終了させる、という
方法を教えてもらいました。

　バグの確認コードとパッチをお送りします。もし、適当でしたら取り込んで頂
けると幸いです。

確認コード:

```
loop{
cls = (0..10_000).map{Class.new}
cls.each{|c| c.new}
ObjectSpace.each_object{|e| e.object_id; ''}
}
```

パッチ:

```
Index: gc.c
===================================================================
--- gc.c   (revision 29471)
+++ gc.c   (working copy)
@@ -2040,6 +2040,17 @@
return FALSE;
}

+static void
+rest_sweep(rb_objspace_t *objspace)
+{
```

- if (objspace->heap.sweep_slots) {
- while (objspace->heap.sweep_slots) {
- lazy_sweep(objspace);
- }
- after_gc_sweep(objspace);
- } +} + static void gc_marks(rb_objspace_t *objspace);

static int
@@ -2536,6 +2547,8 @@
rb_objspace_t *objspace = &rb_objspace;
volatile VALUE v;

- rest_sweep(objspace); +   i = 0;   while (i < heaps_used) { while (0 < i && (uintptr_t)membase < (uintptr_t)objspace->heap.sorted[i-1].slot->membase)

--
// SASADA Koichi at atdot dot net
=end

Associated revisions

Revision a29dc5676415163b0e2416a07de4a9d8055a4670 - 10/21/2010 04:18 AM - nari

- gc.c (rb_objspace_each_objects): don't lazy sweep in rb_objspace_each_objects. [Bug #3940] [ruby-dev:42369]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@29543 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision a29dc567 - 10/21/2010 04:18 AM - nari

- gc.c (rb_objspace_each_objects): don't lazy sweep in rb_objspace_each_objects. [Bug #3940] [ruby-dev:42369]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@29543 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 29543 - 10/21/2010 04:18 AM - nari

- gc.c (rb_objspace_each_objects): don't lazy sweep in rb_objspace_each_objects. [Bug #3940] [ruby-dev:42369]

Revision 29543 - 10/21/2010 04:18 AM - nari

- gc.c (rb_objspace_each_objects): don't lazy sweep in rb_objspace_each_objects. [Bug #3940] [ruby-dev:42369]

Revision 29543 - 10/21/2010 04:18 AM - nari

- gc.c (rb_objspace_each_objects): don't lazy sweep in rb_objspace_each_objects. [Bug #3940] [ruby-dev:42369]

Revision 29543 - 10/21/2010 04:18 AM - nari

- gc.c (rb_objspace_each_objects): don't lazy sweep in rb_objspace_each_objects. [Bug #3940] [ruby-dev:42369]

Revision 29543 - 10/21/2010 04:18 AM - nari

- gc.c (rb_objspace_each_objects): don't lazy sweep in rb_objspace_each_objects. [Bug #3940] [ruby-dev:42369]

Revision 29543 - 10/21/2010 04:18 AM - nari

- gc.c (rb_objspace_each_objects): don't lazy sweep in rb_objspace_each_objects. [Bug #3940] [ruby-dev:42369]

History

#1 - 10/13/2010 02:27 PM - usa (Usaku NAKAMURA)

- Category set to core

- Status changed from Open to Assigned

- Assignee set to authorNari (Narihiro Nakamura)

- Priority changed from 3 to 5

- Target version set to 1.9.3

- ruby -v set to ruby 1.9.3dev (2010-10-13 trunk 29477)

=begin

=end

#2 - 10/14/2010 10:34 AM - matz (Yukihiro Matsumoto)

=begin
まつもと ゆきひろです

In message "Re: [ruby-dev:42369] [BUG: trunk] Lazy sweep and ObjectSpace.each_object"
on Wed, 13 Oct 2010 14:13:46 +0900, SASADA Koichi ko1@atdot.net writes:

|　解決策として、中田さんに ObjectSpace.each_object に相当する関数
|rb_objspace_each_objects を実行する前に sweep を完全に終了させる、という
|方法を教えてもらいました。

オブジェクトがsweep対象かどうかはわかるはずなんだから、sweep
されるはずのものはeach_objectの対象にしない手もあるんじゃな
いの? なんか勘違いしてる?

=end

#3 - 10/14/2010 07:13 PM - ko1 (Koichi Sasada)

=begin
　ささだです。

(2010/10/14 2:34), Yukihiro Matsumoto wrote:

> |　解決策として、中田さんに ObjectSpace.each_object に相当する関数
> |rb_objspace_each_objects を実行する前に sweep を完全に終了させる、という
> |方法を教えてもらいました。
>
> オブジェクトがsweep対象かどうかはわかるはずなんだから、sweep
> されるはずのものはeach_objectの対象にしない手もあるんじゃな
> いの? なんか勘違いしてる?

　手もあると思います。

　ただ、 "rb_objspace_each_objects()" の仕様上、slot を callback にその
まま渡すようにしているため、free / sweep 対象かは、callback 先で確認しな
ければなりません。free オブジェクトかどうかは、

```
flag が 0 かどうか
```

で見ることが出来ますが、sweep 対象（実質 free）であるかどうかは、

```
slot が sweep_slots かつ、
mark されていないか
```

を確認しなければなりません。callback に sweep_slots かどうかを入れるのも
大変だし、そもそも rb_objspace_each_objects() なんて使われるのはレアケー
スだし、まー全部やっちゃえばいいか、という感じです。

　と、今気づきましたが、rb_objspace_each_objects() の callback で GC が
起きても同じ問題がありますね。callback する前で sweep を完了するように変
更しても、結局問題は残るし。どうしようかな。

　上記 check を行う rb_objspace_live_p(VALUE) みたいなのを用意して、これ
を使うようにしてもらう、とか?

--
// SASADA Koichi at atdot dot net

=end

#4 - 10/15/2010 05:17 PM - keiju (Keiju Ishitsuka)

=begin
けいじゅ@いしつかです.

In [ruby-dev:42369] the message: "[ruby-dev:42369] [BUG: trunk] Lazy
sweep and ObjectSpace.each_object", on Oct/13 14:13(JST) SASADA Koichi
writes:

> 　ささだです。

> 　Lazy sweep 中に ObjectSpace.each_object (相当) を行うと、SEGV するので、下記のようなパッチを作成しました。

ちょっとお聞きしたいんですけど, この問題って, どの辺りのバージョンから
入っています?

それと, 1.9.2での話なんですが, finalizerを指定している場合プロセス終了
時にSEGVすることがあるんですが, これと関係ありますでしょうか?

__
-->> 石塚 圭樹 <<---
--------------------------------->> e-mail: keiju@ishitsuka.com <<---

=end

#5 - 10/15/2010 05:27 PM - ko1 (Koichi Sasada)

=begin
　ささだです。

(2010/10/15 9:16), 石塚圭樹 wrote:

> > 　Lazy sweep 中に ObjectSpace.each_object (相当) を行うと、SEGV するので、下記のようなパッチを作成しました。
> > ちょっとお聞きしたいんですけど, この問題って, どの辺りのバージョンから
> > 入っています?
>
> それと, 1.9.2での話なんですが, finalizerを指定している場合プロセス終了
> 時にSEGVすることがあるんですが, これと関係ありますでしょうか?

　1.9.3 以降です。

　1.9.2 のほうは、別の問題かと思います。

--
// SASADA Koichi at atdot dot net

=end

#6 - 10/15/2010 08:40 PM - keiju (Keiju Ishitsuka)

=begin
けいじゅ@いしつかです.

In [ruby-dev:42396] the message: "[ruby-dev:42396] Re: [BUG: trunk]
Lazy sweep and ObjectSpace.each_object", on Oct/15 17:26(JST) SASADA
Koichi writes:

> 　ささだです。
>
> > ちょっとお聞きしたいんですけど, この問題って, どの辺りのバージョンから
> > 入っています?
>
> 　1.9.3 以降です。

そうですか, 残念.

ところで, 添付のようなエラーが出てSEGVしちゃうんですけど, 原因が何か分
かります?

何をやっているかというと, 親プロセスから子プロセスをforkしていて, 子側
がSEGVしています.

親側では, threadとかmutexを使っていて色々と非同期処理をしています.
fork後, 子側でそのthreadがGCされるときに発生しているみたいなんですが...

非同期処理をしながらforkするところに無理があるって話もありそうなんです
が...

それと, このバックトレースではeach_object中に発生していますが, その前
に GC.start を入れると, そこで発生します.

-- ここから
/var/projects/fairy/fairy/lib/fairy/share/base-app.rb:94: [BUG] thread_free: locking_mutex must be NULL (0x87fd5f0:138337388)
ruby 1.9.1p430 (2010-08-16 revision 28998) [i686-linux]

-- control frame ----------
c:0015 p:0015 s:0049 b:0048 l:000031 d:000047 BLOCK /var/projects/fairy/fairy/lib/fairy/share/base-app.rb:94
c:0014 p:---- s:0045 b:0045 l:000044 d:000044 FINISH
c:0013 p:---- s:0043 b:0043 l:000042 d:000042 CFUNC :each_object
c:0012 p:0022 s:0039 b:0039 l:000031 d:000038 BLOCK /var/projects/fairy/fairy/lib/fairy/share/base-app.rb:92
c:0011 p:---- s:0037 b:0037 l:000036 d:000036 FINISH
c:0010 p:---- s:0035 b:0035 l:000034 d:000034 CFUNC :fork
c:0009 p:0035 s:0032 b:0032 l:000031 d:000031 METHOD /var/projects/fairy/fairy/lib/fairy/share/base-app.rb:91
c:0008 p:0016 s:0027 b:0027 l:000026 d:000026 METHOD /var/projects/fairy/fairy/lib/fairy/share/base-app.rb:18
c:0007 p:0080 s:0022 b:0022 l:000015 d:000021 BLOCK /var/projects/fairy/fairy/lib/fairy/master.rb:112
c:0006 p:0019 s:0019 b:0019 l:000018 d:000018 METHOD internal:prelude:8
c:0005 p:0012 s:0016 b:0016 l:000015 d:000015 METHOD /var/projects/fairy/fairy/lib/fairy/master.rb:110
c:0004 p:0145 s:0013 b:0013 l:000012 d:000012 METHOD /var/projects/fairy/lib/deep-connect/evaluator.rb:32
c:0003 p:0064 s:0006 b:0006 l:0023fc d:000005 BLOCK /var/projects/fairy/lib/deep-connect/session.rb:155
c:0002 p:---- s:0004 b:0004 l:000003 d:000003 FINISH
c:0001 p:---- s:0002 b:0002 l:000001 d:000001 CONT#0 controller.rb[186] Controller[0]#block (2 levels) in terminate: TERMINATE: #2.5.3
CONT#0 controller.rb[191] Controller[0]#block (2 levels) in terminate: TERMINATE: #2.6
CONT#0 controller.rb[166] Controller[0]#block (2 levels) in terminate: TERMINATE: #2.3
CONT#0 controller.rb[168] Controller[0]#block (2 levels) in terminate: TERMINATE: #2.4
TOP

-- Ruby level backtrace information--
/var/projects/fairy/fairy/lib/fairy/share/base-app.rb:94:in block (2 levels) in start_subcommand'
/var/projects/fairy/fairy/lib/fairy/share/base-app.rb:92:ineach_object'
/var/projects/fairy/fairy/lib/fairy/share/base-app.rb:92:in block in start_subcommand'
/var/projects/fairy/fairy/lib/fairy/share/base-app.rb:91:infork'
/var/projects/fairy/fairy/lib/fairy/share/base-app.rb:91:in start_subcommand'
/var/projects/fairy/fairy/lib/fairy/share/base-app.rb:18:instart_subcommand'
/var/projects/fairy/fairy/lib/fairy/master.rb:112:in block in assgin_controller'
<internal:prelude>:8:insynchronize'
[M] master.rb[80] Master#when_disconnected: MASTER: disconnected: Start terminationcts/fairy/lib/deep-connect/evaluator.rb:32:in
evaluate_request'
/var/projects/fairy/lib/deep-connect/session.rb:155:inblock in receive'

-- C level backtrace information ---
0x8157b71 fairy master (rb_vm_bugreport+0xa1) [0x8157b71]
0x8193156 fairy master [0x8193156]
0x8193208 fairy master (rb_bug+0x28) [0x8193208]
0x814311d fairy master [0x814311d]
0x8064e9e fairy master [0x8064e9e]
0x8065339 fairy master (rb_gc_finalize_deferred+0x59) [0x8065339]
0x815c2ad fairy master [0x815c2ad]
0x8155fe9 fairy master [0x8155fe9]
0x8148ddb fairy master [0x8148ddb]
0x814f644 fairy master [0x814f644]
0x8150a7f fairy master (rb_yield+0x4f) [0x8150a7f]
0x80682e6 fairy master [0x80682e6]
0x8143f7d fairy master [0x8143f7d]
0x81440a4 fairy master [0x81440a4]
0x8155f54 fairy master [0x8155f54]
0x8148ddb fairy master [0x8148ddb]

__
--->> 石塚 圭樹 <<---
---------------------------------->> e-mail: keiju@ishitsuka.com <<---

=end

#7 - 10/15/2010 09:08 PM - ko1 (Koichi Sasada)

=begin
□ささだです。

(2010/10/15 12:40), 石塚圭樹 wrote:

> ところで, 添付のようなエラーが出てSEGVしちゃうんですけど, 原因が何か分
> かります?

> 何をやっているかというと，親プロセスから子プロセスをforkしていて，子側
> がSEGVしています．
>
> 親側では，threadとかmutexを使っていて色々と非同期処理をしています．
> fork後，子側でそのthreadがGCされるときに発生しているみたいなんですが...
>
> 非同期処理をしながらforkするところに無理があるって話もありそうなんです
> が...
>
> それと，このバックトレースではeach_object中に発生していますが，その前
> に GC.start を入れると，そこで発生します．
>
> -- ここから
> /var/projects/fairy/fairy/lib/fairy/share/base-app.rb:94: [BUG] thread_free: locking_mutex must be NULL (0x87fd5f0:138337388)
> ruby 1.9.1p430 (2010-08-16 revision 28998) [i686-linux]

□さすがにこれだけでは...、と思ったけど、fork した直後に locking_mutex
を空にする処理を走らせているのに、それが間に合ってない感じですかね。

□ファイナライザで何をやっているか、などを併せて最小ケースを貰えると良い
かもしれません。

--
// SASADA Koichi at atdot dot net

=end

#8 - 10/18/2010 01:02 PM - authorNari (Narihiro Nakamura)

=begin
nariです。

2010年10月14日19:13 SASADA Koichi ko1@atdot.net:

> □ささだです。
>
> (2010/10/14 2:34), Yukihiro Matsumoto wrote:
>
>> |□解決策として、中田さんに ObjectSpace.each_object に相当する関数
>> |rb_objspace_each_objects を実行する前に sweep を完全に終了させる、という
>> |方法を教えてもらいました。
>>
>> オブジェクトがsweep対象かどうかはわかるはずなんだから、sweep
>> されるはずのものはeach_objectの対象にしない手もあるんじゃな
>> いの？ なんか勘違いしてる？
>
> □手もあると思います。
>
> ただ、"rb_objspace_each_objects()" の仕様上、slot を callback にその
> まま渡すようにしているため、free / sweep 対象かは、callback 先で確認しな
> ければなりません。free オブジェクトかどうかは、
>
> ```
>   flag が 0 かどうか
> ```
>
> で見ることが出来ますが、sweep 対象（実質 free）であるかどうかは、
>
> ```
>   slot が sweep_slots かつ、
>   mark されていないか
> ```
>
> を確認しなければなりません。callback に sweep_slots かどうかを入れるのも
> 大変だし、そもそも rb_objspace_each_objects() なんて使われるのはレアケー
> スだし、まー全部やっちゃえばいいか、という感じです。
>
> と、今気づきましたが、rb_objspace_each_objects() の callback で GC が
> 起きても同じ問題がありますね。callback する前で sweep を完了するように変
> 更しても、結局問題は残るし。どうしようかな。
>
> 上記 check を行う rb_objspace_live_p(VALUE) みたいなのを用意して、これ
> を使うようにしてもらう、とか？
>
> --
> // SASADA Koichi at atdot dot net

rb_objspace_each_objects() の実行前には sweep を完全に終了させることに
して、実行中は rb_gc_disable() して、実行後に元に戻すようにどうかなと思った
のですが、いかがでしょうか?

--
Narihiro Nakamura (nari)

=end

#9 - 10/18/2010 01:16 PM - ko1 (Koichi Sasada)

=begin
　ささだです。

(2010/10/18 5:01), Narihiro Nakamura wrote:

> rb_objspace_each_objects() の実行前には sweep を完全に終了させることに
> して、実行中は rb_gc_disable() して、実行後に元に戻すようにどうかなと思った
> のですが、いかがでしょうか?

　GC.enable で回避できてしまう（SEGV するパスが出来てしまう）のでまずい
のではないでしょうか。

　少し考えてみたのですが、例えば、Lazy sweep 禁止フラグ（eager に sweep
する）を設けるのはどうでしょうか。Ruby 側からは触らせないフラグにすれ
ば、SEGV する危険は防ぐことが出来るのではないかと。

--
// SASADA Koichi at atdot dot net

=end

#10 - 10/18/2010 01:35 PM - authorNari (Narihiro Nakamura)

=begin
nariです。

2010年10月18日13:15 SASADA Koichi ko1@atdot.net:

> 　ささだです。
>
> (2010/10/18 5:01), Narihiro Nakamura wrote:
>
>> rb_objspace_each_objects() の実行前には sweep を完全に終了させることに
>> して、実行中は rb_gc_disable() して、実行後に元に戻すようにどうかなと思った
>> のですが、いかがでしょうか?
>
> 　GC.enable で回避できてしまう（SEGV するパスが出来てしまう）のでまずい
> のではないでしょうか。

そうですね。

> 少し考えてみたのですが、例えば、Lazy sweep 禁止フラグ（eager に sweep
> する）を設けるのはどうでしょうか。Ruby 側からは触らせないフラグにすれ
> ば、SEGV する危険は防ぐことが出来るのではないかと。

なるほど。そのように作ってみます。

--
Narihiro Nakamura (nari)

=end

#11 - 10/18/2010 03:14 PM - authorNari (Narihiro Nakamura)

=begin
nariです。

Lazy sweep 禁止フラグを作りました。
反対がなければコミットしたいと思います。

callbackの途中でGCが呼ばれてSEGVを起こすようなコードを作成し、通ることを確認しました。
make checkも通ることを確認しています。

確認コード:

```
loop {
GC.disable
10.times do
a = []
1000.times{ a << "" }
a.dup
end
GC.enable
ObjectSpace.each_object(Array) do |a|
a.map(&:object_id)
10000.times{''.dup}
end
}
```

パッチ:

```
diff --git a/gc.c b/gc.c
index b011f4a..a9a4560 100644
--- a/gc.c
+++ b/gc.c
@@ -332,6 +332,7 @@ typedef struct rb_objspace {
} heap;
struct {
int dont_gc;
```

- int dont_lazy_sweep; int during_gc;   } flags;   struct { @@ -2040,6 +2041,17 @@ lazy_sweep(rb_objspace_t *objspace)   return FALSE; }

```
+static void
+rest_sweep(rb_objspace_t *objspace)
+{
```

- if (objspace->heap.sweep_slots) {
- while (objspace->heap.sweep_slots) {
- lazy_sweep(objspace);
- }
- after_gc_sweep(objspace);
- } +} + static void gc_marks(rb_objspace_t *objspace);

```
static int
@@ -2047,6 +2059,9 @@ gc_lazy_sweep(rb_objspace_t *objspace)
{
int res;
```

- if (objspace->flags.dont_lazy_sweep)

- [redacted]

  ```
  +
  INIT_GC_PROF_PARAMS;

  if (!ready_to_gc(objspace)) return TRUE;
  @@ -2536,6 +2551,9 @@ rb_objspace_each_objects(int (*callback)(void
  *vstart, void *vend,
  rb_objspace_t *objspace = &rb_objspace;
  volatile VALUE v;
  ```

- rest_sweep(objspace);

- objspace->flags.dont_lazy_sweep = TRUE;

  ```
  +
  i = 0;
  while (i < heaps_used) {
  while (0 < i && (uintptr_t)membase <
  (uintptr_t)objspace->heap.sorted[i-1].slot->membase)
  @@ -2562,6 +2580,7 @@ rb_objspace_each_objects(int (*callback)(void
  *vstart, void *vend,
  }
  }
  ```

- objspace->flags.dont_lazy_sweep = FALSE;
  return;
  }

--
Narihiro Nakamura (nari)

=end

#12 - 10/18/2010 11:54 PM - ko1 (Koichi Sasada)

=begin
　ささだです。

(2010/10/18 7:14), Narihiro Nakamura wrote:

> Lazy sweep 禁止フラグを作りました。
> 反対がなければコミットしたいと思います。
>
> callbackの途中でGCが呼ばれてSEGVを起こすようなコードを作成し、通ることを確認しました。
> make checkも通ることを確認しています。

　これだと、例えば例外が出た場合、dont_lazy_sweep が TRUE のままになって
しまうような気がします。ensure で囲まないと。

余談

　ちょっと関係ないですが、gc_lazy_sweep() という関数は、下記の挙動だと思
うのですが;

・もし、遅延した slot があれば、sweep をする
　これで、freelist が NULL でなければ、ここで終わり
・GC を開始（mark）
・遅延 sweep をして、結果を返す

sweep という関数で mark までしちゃうのはいいでしょうか。

　あまり、対案はないのですが、なんとなく、garbage_collect() を今の
gc_lazy_sweep() として、今の garbage_collect() を、
garbage_collect_eager() みたいにしてしまうのはどうだろう、と思いました。

--
// SASADA Koichi at atdot dot net

=end

#13 - 10/19/2010 02:55 PM - authorNari (Narihiro Nakamura)

=begin
nariです。

> 　これだと、例えば例外が出た場合、dont_lazy_sweep が TRUE のままになって
> しまうような気がします。ensure で囲まないと。

ぉぉ、そうですね...。ご指摘ありがとうございます。
そのようなパッチを作ってみました。本メールの一番最後に貼り付けておきます。

> 余談
>
> ちょっと関係ないですが、gc_lazy_sweep() という関数は、下記の挙動だと思
> うのですが;
>
> ・もし、遅延した slot があれば、sweep をする
> これで、freelist が NULL でなければ、ここで終わり
> ・GC を開始（mark）
> ・遅延 sweep をして、結果を返す
>
> sweep という関数で mark までしちゃうのはいいでしょうか。

なるほど。

関数名にgc_というプレフィクスが付いているので、マークしてもいいのかなとか、そういう言い訳を思いついたのですがやっぱりダメですね。

> あまり、対案はないのですが、なんとなく、garbage_collect() を今の gc_lazy_sweep() として、今の garbage_collect() を、garbage_collect_eager() みたいにしてしまうのはどうだろう、と思いました。

garbage_collect_eager() よりも garbage_collect_full() の方が馴染みがあるのかなと思いました。こちらの方も暇を見つけて直しておきます。

パッチ:
diff --git a/gc.c b/gc.c
index b011f4a..ad49fd5 100644
--- a/gc.c
+++ b/gc.c
@@ -332,6 +332,7 @@ typedef struct rb_objspace {
} heap;
struct {
int dont_gc;

- int dont_lazy_sweep; int during_gc;   } flags;   struct { @@ -2040,6 +2041,17 @@ lazy_sweep(rb_objspace_t *objspace)   return FALSE; }

+static void
+rest_sweep(rb_objspace_t *objspace)
+{

- if (objspace->heap.sweep_slots) {
- while (objspace->heap.sweep_slots) {
- lazy_sweep(objspace);
- }
- after_gc_sweep(objspace);
- } +} + static void gc_marks(rb_objspace_t *objspace);

static int
@@ -2047,6 +2059,9 @@ gc_lazy_sweep(rb_objspace_t *objspace)
{
int res;

- if (objspace->flags.dont_lazy_sweep)

- [redacted]

  +
  INIT_GC_PROF_PARAMS;

  if (!ready_to_gc(objspace)) return TRUE;
  @@ -2489,6 +2504,55 @@ Init_heap(void)
  init_heap(&rb_objspace);
  }

+
+static VALUE
+lazy_sweep_enable(void)
+{

- rb_objspace_t *objspace = &rb_objspace; +
- objspace->flags.dont_lazy_sweep = FALSE;
- return Qnil; +} + +static VALUE +objspace_each_objects(VALUE arg) +{
- size_t i;
- RVALUE *membase = 0;
- RVALUE *pstart, *pend;
- rb_objspace_t *objspace = &rb_objspace;
- VALUE *args = (VALUE *)arg;
- volatile VALUE v; +
- i = 0;
- while (i < heaps_used) {
- while (0 < i && (uintptr_t)membase < (uintptr_t)objspace->heap.sorted[i-1].slot->membase)
- i--;
- while (i < heaps_used && (uintptr_t)objspace->heap.sorted[i].slot->membase <= (uintptr_t)membase)
- i++;
- if (heaps_used <= i)
- break;

- membase = objspace->heap.sorted[i].slot->membase; +
- pstart = objspace->heap.sorted[i].slot->slot;
- pend = pstart + objspace->heap.sorted[i].slot->limit; +
- for (; pstart != pend; pstart++) {
- if (pstart->as.basic.flags) {
- v = (VALUE)pstart; /* acquire to save this object */
- break;
- }
- }
- if (pstart != pend) {
- if ((*(int ()*)(void *, void *, size_t, void *))args[0])(pstart, pend, sizeof(RVALUE), (void *)args[1])) {
- return;
- }
- }
- } +
- return Qnil; +} + /*
  - rb_objspace_each_objects() is special C API to walk through
  - Ruby object space. This C API is too difficult to use it. @@ -2530,39 +2594,15 @@ rb_objspace_each_objects(int (*callback)(void *vstart, void *vend, size_t stride, void *d), void *data) {
- size_t i;
- RVALUE *membase = 0;
- RVALUE *pstart, *pend;
- VALUE args[2]; rb_objspace_t *objspace = &rb_objspace;
- volatile VALUE v; -
- i = 0;
- while (i < heaps_used) {
- while (0 < i && (uintptr_t)membase < (uintptr_t)objspace->heap.sorted[i-1].slot->membase)
- i--;
- while (i < heaps_used && (uintptr_t)objspace->heap.sorted[i].slot->membase <= (uintptr_t)membase)
- i++;
- if (heaps_used <= i)
- break;

- membase = objspace->heap.sorted[i].slot->membase;

- pstart = objspace->heap.sorted[i].slot->slot;

- **pend = pstart + objspace->heap.sorted[i].slot->limit;**

- for (; pstart != pend; pstart++) {

- [redacted]

- [redacted]

- [redacted]

- [redacted]

- }

- if (pstart != pend) {

- [redacted]

- [redacted]

- [redacted]

- }

- }

- rest_sweep(objspace);

- objspace->flags.dont_lazy_sweep = TRUE;

- return;

- args[0] = (VALUE)callback;

- args[1] = (VALUE)data;

- rb_ensure(objspace_each_objects, (VALUE)args, lazy_sweep_enable, Qnil);

```
    }

struct os_each_struct {

--
Narihiro Nakamura (nari)

=end
```

#14 - 10/21/2010 01:27 PM - Anonymous

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

```
=begin
This issue was solved with changeset r29543.
Koichi, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

=end
```