Finding Memory Leaks in the Ruby Ecosystem

Peter Zhu

Ruby Core Committer
Senior Developer, Shopify

Adam Hess

Staff Developer, GitHub

https://blog.peterzhu.ca/assets/rubykaigi_2024_slides.pdf

Peter Zhu

- Based in Toronto, Canada
- Ruby Core Committer
- Senior Developer on the Ruby Infrastructure team at Shopify
- Co-author of Variable Width Allocation in Ruby
- Author of ruby_memcheck and autotuner
- Photography geek, follow me @peterzhu.photos on Instagram!

Adam Hess

- Based on Seattle, WA
- Staff Software Engineer on the Ruby Architecture team at GitHub
- Early Prism contributor
- General Parser and compiler nerd

Memory Leaks

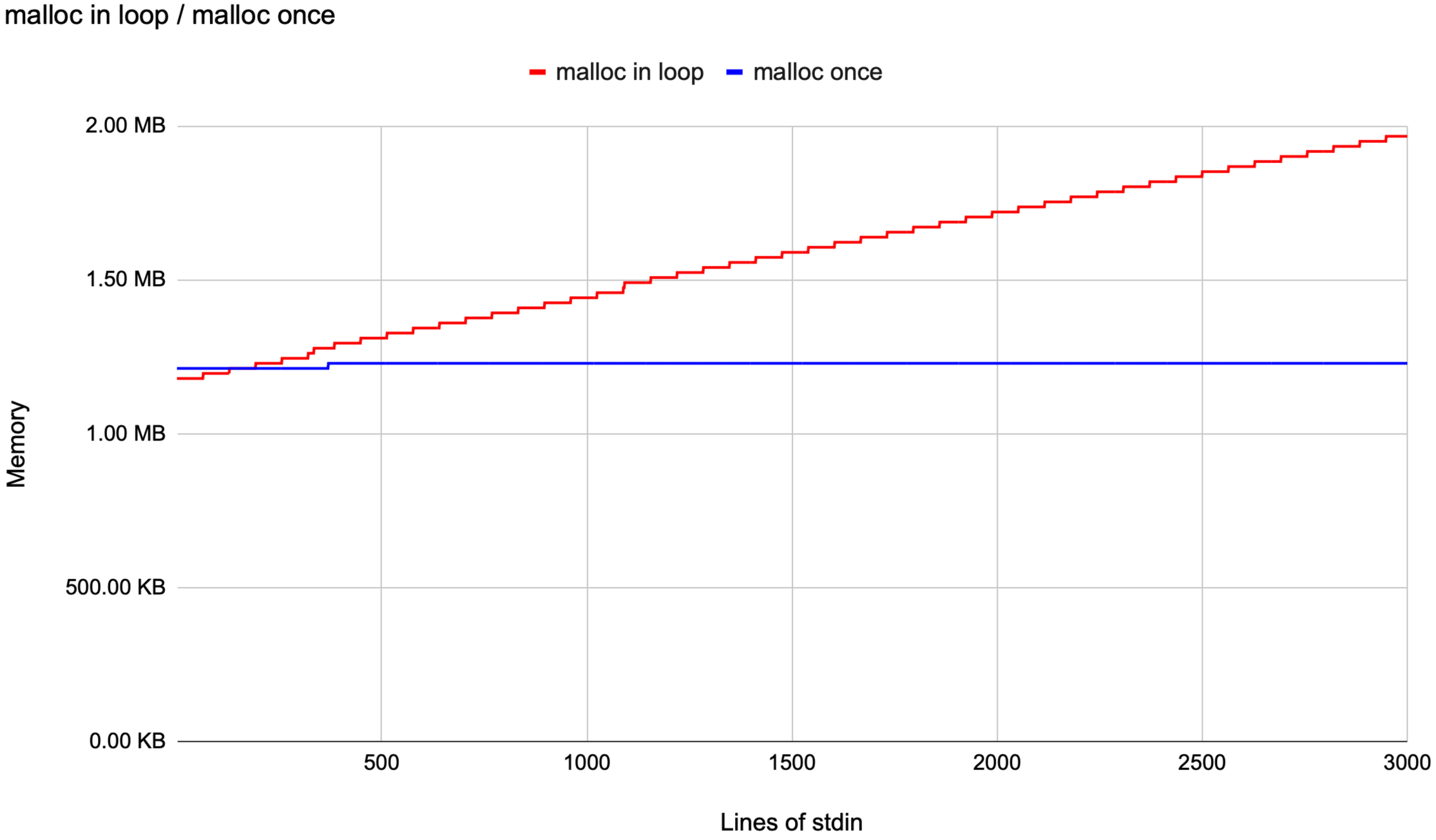
Allocation in Loop

```
int main() {
  while(1) {
    char * name = malloc(256);
    if (fgets(name, 256, stdin) == NULL) {
      break;
    }
    printf("%s", name);
  }
}
```

Allocate once

```
int main() {
  char * name = malloc(256);
  while(1) {
    if (fgets(name, 256, stdin) == NULL) {
      break;
    }
    printf("%s", name);
  }
}
```

What would happen if you ran these two programs?


malloc in loop / malloc once
malloc in loop
malloc once
2.00 MB
1.50 MB
1.00 MB
500.00 KB
0.00 KB
Memory
500
1000
1500
2000
2500
3000
Lines of stdin

Memory leaks are bad

What if our program is complex?

What can Valgrind tell us?

Allocation in Loop

```c
int main() {
  while(1) {
    char * name = malloc(256);
    if (fgets(name, 256, stdin) == NULL) {
      break;
    }
    printf("%s", name);
  }
}
```

Allocate Once

```c
int main() {
  char * name = malloc(256);
  while(1) {
    if (fgets(name, 256, stdin) == NULL) {
      break;
    }
    printf("%s", name);
  }
}
```

The leaks look the same


```
==29710== 256 bytes in 1 blocks are definitely lost in loss record 1 of 1
==29710==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==29710==    by 0x109167: main (in /workspaces/github/a.out)
```

```
==29710== 512 bytes in 2 blocks are definitely lost in loss record 1 of 1
==29710==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==29710==    by 0x109167: main (in /workspaces/github/a.out)
```

Can we Help Valgrind identify only meaningful memory leaks?

Prior Art

[EN]Automatically Find Memory Leaks in Native Gems / Peter Zhu @peterzhu2118

https://youtu.be/SchKPrZefXY

ruby_memcheck uses Valgrind Memcheck

```
$ valgrind --num-callers=50 --error-limit=no --undef-value-errors=no --leak-check=full --show-leak-kinds=definite ruby -e ""

==2504== Memcheck, a memory error detector
==2504== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2504== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==2504== Command: ruby -e
==2504==
==2504== Warning: client switching stacks?  SP change: 0x1ffe8020e0 ⟶ 0x1fff000010
==2504==          to suppress, use: --max-stackframe=8380208 or greater
==2504==
==2504== HEAP SUMMARY:
==2504==     in use at exit: 3,285,702 bytes in 20,867 blocks
==2504==   total heap usage: 67,572 allocs, 46,705 frees, 17,742,444 bytes allocated
==2504==
==2504== 8 bytes in 1 blocks are definitely lost in loss record 390 of 11,569
==2504==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==2504==    by 0x1609B5: objspace_xmalloc0 (gc.c:11445)
==2504==    by 0x3C781E: ibf_load_alloc (compile.c:10827)
==2504==    by 0x3C781E: ibf_load_param_keyword (compile.c:11280)
==2504==    by 0x3C781E: ibf_load_iseq_each (compile.c:11856)
==2504==    by 0x3C781E: rb_ibf_load_iseq_complete (compile.c:12748)
==2504==    by 0x3CC001: ibf_load_iseq.isra.0 (compile.c:12803)
==2504==    by 0x3CC4B7: ibf_load_code.isra.0 (compile.c:11166)
==2504==    by 0x3C8A1A: ibf_load_iseq_each (compile.c:11866)
==2504==    by 0x3C8A1A: rb_ibf_load_iseq_complete (compile.c:12748)
```

Valgrind is unusable on Ruby

ruby_memcheck uses heuristics to filter false-positives

Did ruby_memcheck work?


[Ruby] Fix memory leak in grpc_rb_call_run_batch #33368
Merged
apolcyn merged 1 commit into grpc:master from Shopify:pz-call-run-batch-mem-leak on Jun 9, 2023
Conversation 1
Commits 1
Checks 3
Files changed 1
peterzhu2118 commented on Jun 7, 2023
The function grpc_rb_call_run_batch has many places that could raise errors, including in child functions. Since a raised error will longjump out of the function, it will cause memory leaks since the function cannot perform any clean up. This commit fixes the issue by wrapping the whole function in an rb_ensure, which will ensure that a cleanup function is ran before the error is propagated upwards.
Contributor
Reviewers
veblush
apolcyn
Assignees
apolcyn
Notifications
Fork 155
Star 939
Actions
Projects
Security
Insights
Issues 599
Pull requests 288
New issue
Fork 10.3k
Star 40.4k
Star 5.9k
Wiki
+227 −105
+60 −36
+60 −39
FROM ruby:3.0
ARG work_dir=/ bundle_dir=/usr/local/bundle # if
WORKDIR ${work_dir}
RUN bundle config set --local path ${bundle_dir}
RUN apt update && apt install -y libgeos-dev valgrind
COPY Gemfile rgeo.gemspec ./
COPY lib/rgeo/version.rb ./lib/rgeo/
RUN bundle install
COPY . .
RUN rake compile

ruby_memcheck is just a hack

https://blog.peterzhu.ca/ruby-memcheck/

https://youtu.be/SchKPrZefXY

home about twitter rss

How ruby_memcheck Finds Memory Leaks in Native Gems

Nov 05, 2021 (updated at Nov 14, 2022)

This is not an article on how to use ruby_memcheck. Refer to the repository for documentation.

This article was adapted as a talk at RubyKaigi 2022.

If you just want to know how ruby_memcheck works without the backstory, you can jump to the `How ruby_memcheck filters for memory leaks` section.

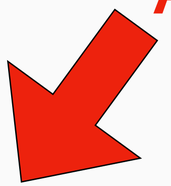
On the afternoon of Friday, October 8, 2021, I found a memory leak in a native gem called liquid-c. I wasn't searching for this memory leak, I just happened to stumble across it. Fixing it was easy enough, but I was sure there were more memory leaks. So I had a choice, I could spend a day or two to debug and find the memory leaks, or I could sink an unknown amount of time to try to make a tool (that may or may or not work) to find it for me. Of course, I chose the latter approach.

So the following week, I spent a week experimenting and prototyping ways to automatically

[EN]Automatically Find Memory Leaks in Native Gems / Peter Zhu @peterzhu2118

Can we do more?

Allocated once



```c
int main() {
  char * name = malloc(256);
  while(1) {
    if (fgets(name, 256, stdin) == NULL) {
      break;
    }
    printf("%s", name);
  }
}
```

Allocated once

```c
int main() {
  char * name = malloc(256);
  while(1) {
    if (fgets(name, 256, stdin) == NULL) {
      break;
    }
    printf("%s", name);
  }
  free(name);
}
```

Free before Exit

Valgrind now knows this doesn't leak

```
==4036== HEAP SUMMARY:
==4036==     in use at exit: 0 bytes in 0 blocks
==4036==   total heap usage: 3 allocs, 3 frees, 5,376 bytes allocated

==4036==
==4036== All heap blocks were freed -- no leaks are possible
```

Can we do this for Ruby?

Yes, we can!

Feature #19993 OPEN

Edit Watch Like 1 ...

Optionally Free all memory at exit

« Previous | Next »

Added by HParker (Adam Hess) 5 months ago. Updated 4 months ago.

Status:	Open
Assignee:	-
Target version:	-

[ruby-core:115304]

Description

Quote

Add a runtime option allowing Ruby to optionally free all memory at shutdown.

why

Today, memory sanitizers are difficult to use with Ruby, since not all memory is freed at shutdown. it is difficult to detect memory leaks or errors in Ruby or in Ruby C extensions when these tools are not available.

While implementing this feature, we were able to identify and fix a number of memory leaks and errors.

https://github.com/ruby/ruby/pull/8556

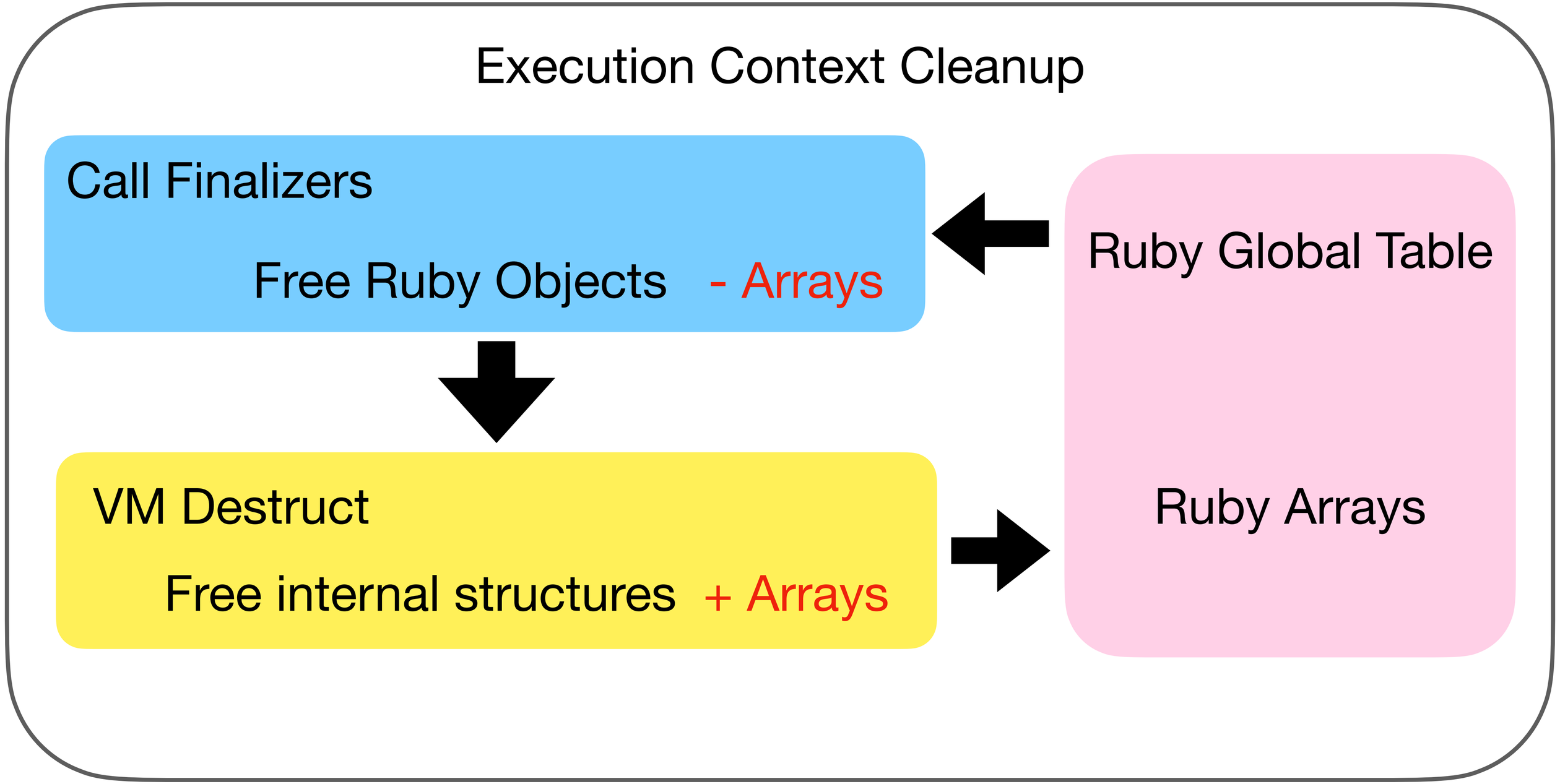
bugs.ruby-lang.org #19993

RUBY_FREE_AT_EXIT=1

Implementing in Ruby

```
if (rb_free_at_exit) {
    rb_free_default_rand_key();
    rb_free_encoded_insn_data();
    rb_free_global_enc_table();
    rb_free_loaded_builtin_table();
    // ... Free everything else
    rb_free_warning();
}
```

There are cyclic problems


Execution Context Cleanup
Call Finalizers
Free Ruby Objects - Arrays
Ruby Global Table
VM Destruct
Free internal structures + Arrays
Ruby Arrays

Using RUBY_FREE_AT_EXIT

RUBY_FREE_AT_EXIT=0

Before

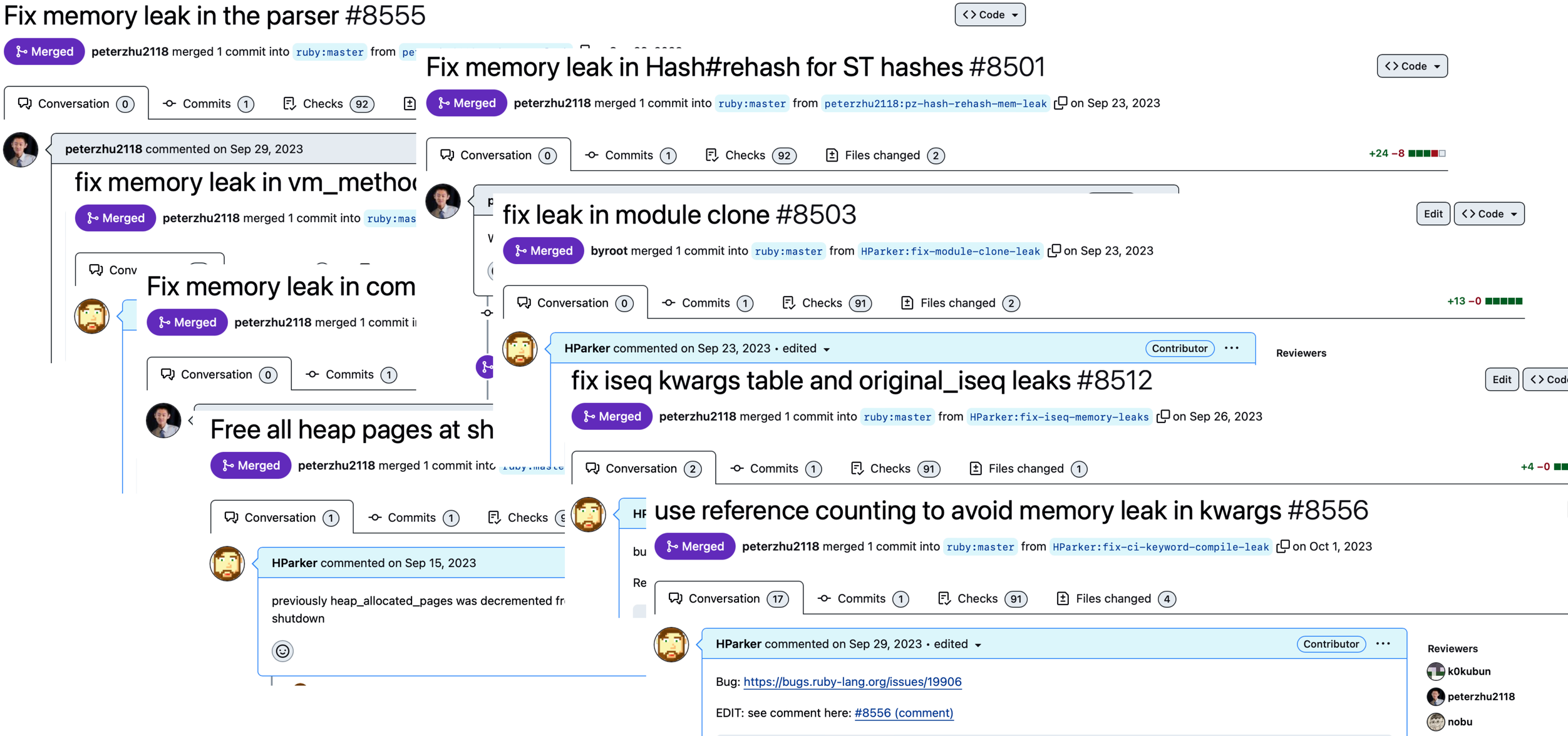
```
valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes -- ./miniruby -e ""

[snip 104,272 lines ...]

==10572== LEAK SUMMARY:
==10572==    definitely lost: 412,680 bytes in 3,983 blocks
==10572==    indirectly lost: 424,137 bytes in 5,191 blocks
==10572==      possibly lost: 1,049,728 bytes in 4 blocks
==10572==    still reachable: 176,420 bytes in 447 blocks
==10572==         suppressed: 0 bytes in 0 blocks
==10572==
==10572== For lists of detected and suppressed errors, rerun with: -s
==10572== ERROR SUMMARY: 3249 errors from 3249 contexts (suppressed: 0 from 0)
```

RUBY_FREE_AT_EXIT=1

After

```
RUBY_FREE_AT_EXIT=1 valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes -- ./miniruby -e ""

[snip 32 lines ...]

==10663== LEAK SUMMARY:
==10663==    definitely lost: 0 bytes in 0 blocks
==10663==    indirectly lost: 0 bytes in 0 blocks
==10663==      possibly lost: 1,104 bytes in 1 blocks
==10663==    still reachable: 0 bytes in 0 blocks
==10663==         suppressed: 0 bytes in 0 blocks
==10663==
==10663== For lists of detected and suppressed errors, rerun with: -s
==10663== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

There were some existing memory leaks



Impacts

Ran Ruby test suite using memory leaks tool and RUBY_FREE_AT_EXIT

ruby/ruby **Fix memory leak when evacuating generic ivars** ✕ 1

#8980 by peterzhu2118 was merged on Nov 21, 2023

ruby/ruby **Fix memory leak in the parser** ✕

#8555 by peterzhu2118 was merged on Sep 29, 2023

ruby/ruby **Fix memory leak in Hash#rehash for ST hashes** ✕

#8501 by peterzhu2118 was merged on Sep 23, 2023

[Bug #20228] Fix memory leak in Regexp timeout #9765

Merged **peterzhu2118** merged 3 commits into `ruby:master` from `peterzhu2118:regexp-timeout-mem-leak` on Feb 2

Conversation 0 · Commits 3 · Checks 97 · Files changed 4

peterzhu2118 commented on Jan 30 — Member

If a Regexp::TimeoutError is raised, the `stk_base` and `OnigRegion` will leak.

For example:

```
Regexp.timeout = 0.001
regex = /^(a*)*$/
str = "a" * 1000000 + "x"

10.times do
  100.times do
    begin
      regex =~ str
    rescue
    end
  end

  puts `ps -o rss= -p #{$$}`
end
```

Before:

```
328800
632416
934368
1230448
1531088
1831248
2125072
2414384
2703440
2995664
```

After:

```
39280
47888
49024
56240
56496
56512
56592
56592
56720
56720
```

https://github.com/ruby/ruby/pull/9765

For example:

```ruby
Regexp.timeout = 0.001
regex = /^(a*)*$/
str = "a" * 1000000 + "x"

10.times do
  100.times do
    begin
      regex =~ str
    rescue
    end
  end

  puts `ps -o rss= -p #{$$}`
end
```

For example:

```
Regexp.timeout = 0.001
regex = /^(a*)*$/
str = "a" * 1000000 + "x"

10.times do
  100.times do
    begin
      regex =~ str
    rescue
    end
  end

  puts `ps -o rss= -p #{$$}`
end
```

For example:

```
Regexp.timeout = 0.001
regex = /^(a*)*$/
str = "a" * 1000000 + "x"

10.times do
  100.times do
    begin
      regex =~ str
    rescue
    end
  end

  puts `ps -o rss= -p #{$$}`
end
```

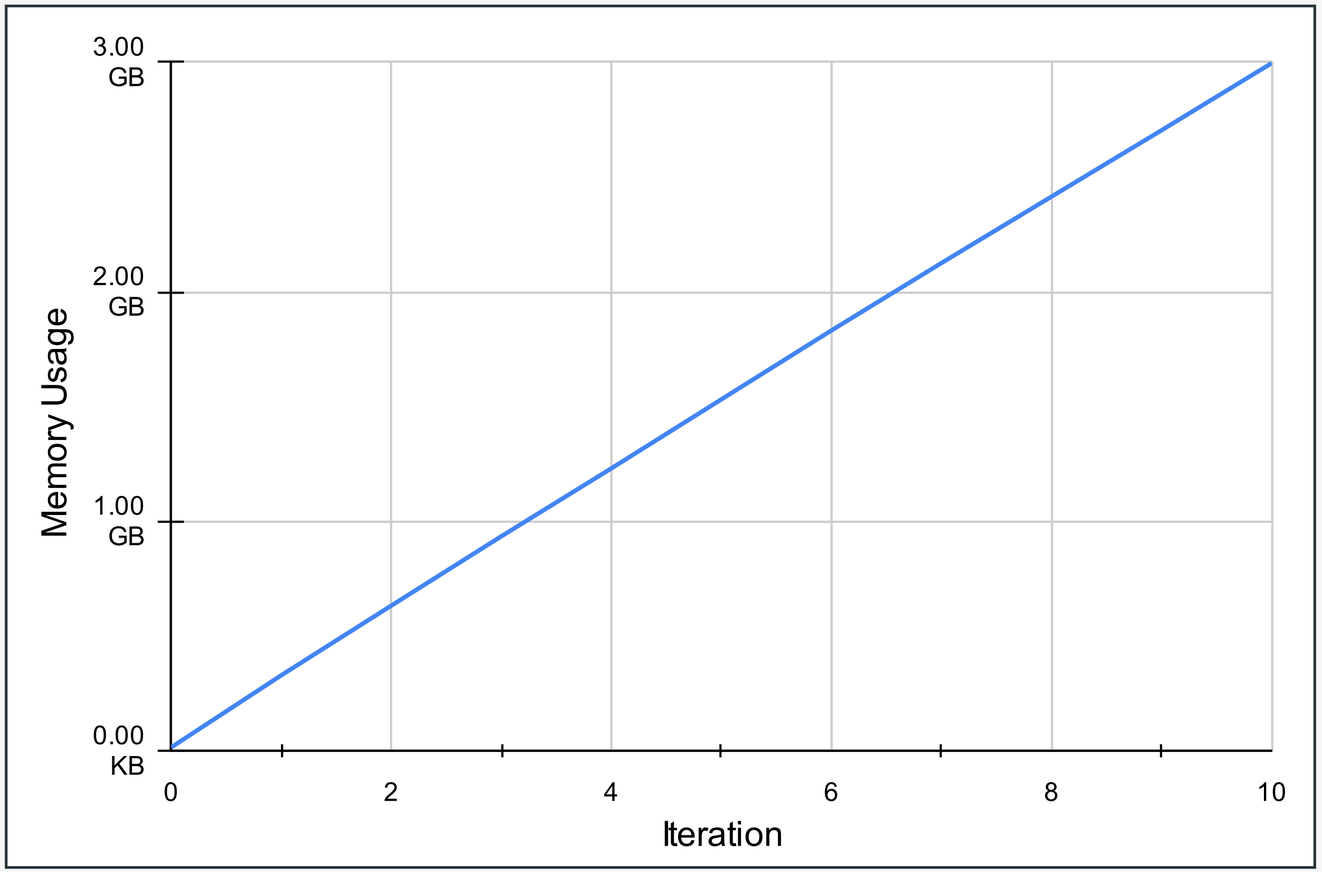
For example:

```ruby
Regexp.timeout = 0.001
regex = /^(a*)*$/
str = "a" * 1000000 + "x"

10.times do
  100.times do
    begin
      regex =~ str
    rescue
    end
  end

  puts `ps -o rss= -p #{$$}`
end
```

```
end
```

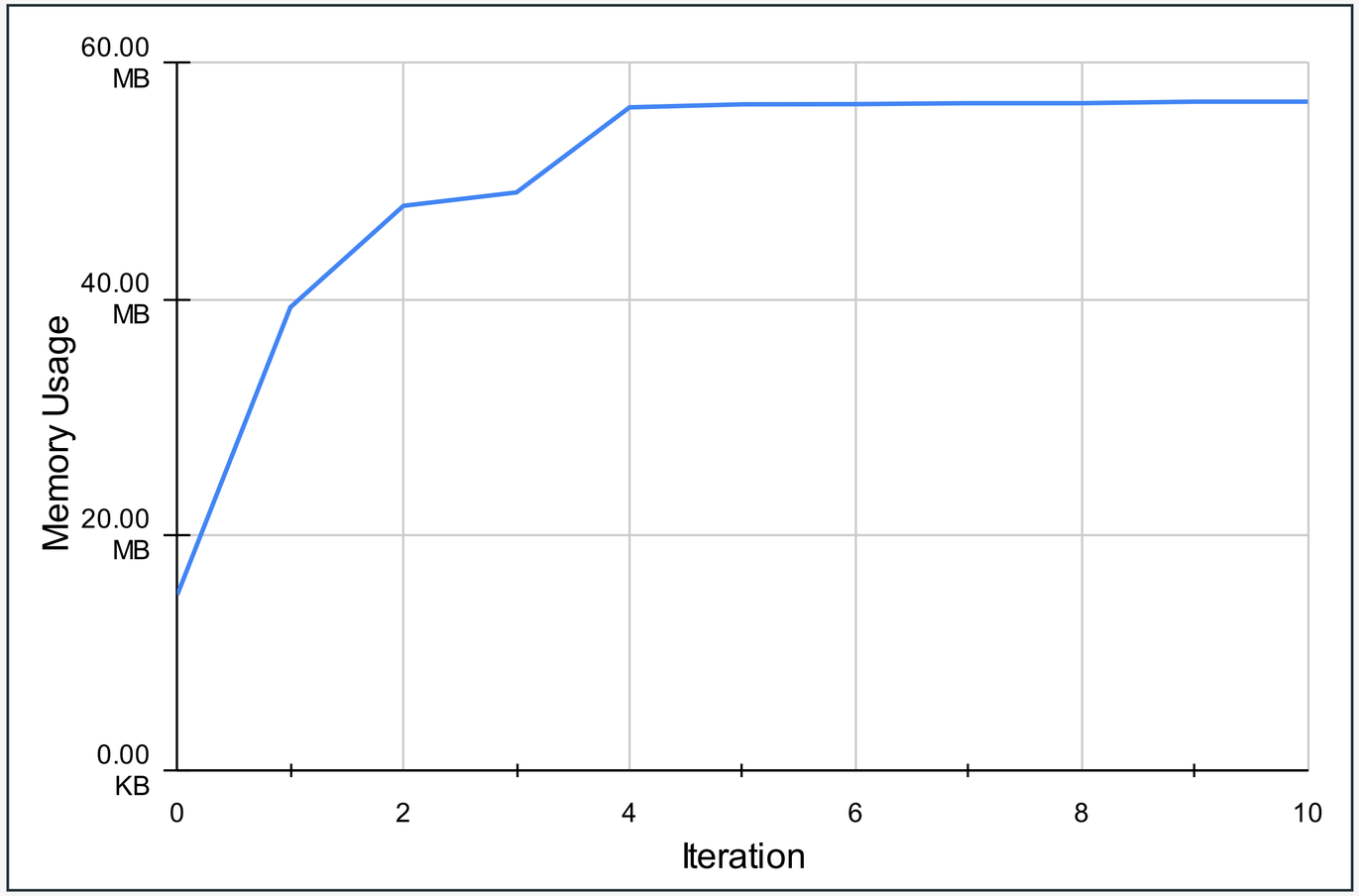
Before:

```
328800
632416
934368
1230448
1531088
1831248
2125072
2414384
2703440
2995664
```

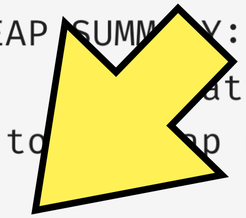


After:

```
39280
47888
49024
56240
56496
56512
56592
56592
56720
56720
```

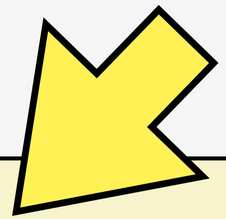


```
$ RUBY_FREE_AT_EXIT=1 valgrind --leak-check=full --undef-value-errors=no --show-possibly-lost=no ruby test.rb
==30828== Memcheck, a memory error detector
==30828== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==30828== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==30828== Command: ruby test.rb
==30828==
==30828== Warning: client switching stacks?  SP change: 0x1ffe8020e0 --> 0x1ffefff930
==30828==          to suppress, use: --max-stackframe=8378448 or greater
==30828== Warning: set address range perms: large range [0x6950000, 0x1e950000) (defined)
./ruby: warning: Free at exit is experimental and may be unstable
==30828==
==30828== HEAP SUMMARY:
==30828==     in use at exit: 40,734 bytes in 75 blocks
==30828==   total heap usage: 80,640 allocs, 80,565 frees, 40,383,673 bytes allocated
==30828==
==30828== 30,720 bytes in 1 blocks are definitely lost in loss record 1 of 11
==30828==    at 0x4850164: realloc (vg_replace_malloc.c:1690)
```



```
==30828==
==30828== HEAP SUMMARY:
==30828==     at exit: 40,734 bytes in 75 blocks
==30828==   total heap usage: 80,640 allocs, 80,565 frees, 40,383,673 bytes allocated
==30828==
==30828== 30,720 bytes in 1 blocks are definitely lost in loss record 1 of 11
==30828==    at 0×4850164: realloc (vg_replace_malloc.c:1690)
==30828==    by 0×281A63: stack_double (regexec.c:1239)
==30828==    by 0×289394: match_at (regexec.c:3749)
==30828==    by 0×28CD4C: onig_search_gpos (regexec.c:5423)
==30828==    by 0×28D289: onig_search (regexec.c:5152)
==30828==    by 0×26DF46: reg_onig_search (re.c:1725)
==30828==    by 0×26DF46: rb_reg_onig_match (re.c:1661)
==30828==    by 0×26DF46: rb_reg_search_set_match (re.c:1752)
==30828==    by 0×26E45E: reg_match_pos (re.c:3578)
==30828==    by 0×26E45E: rb_reg_match (re.c:3640)
==30828==    by 0×34F3A3: vm_opt_regexpmatch2 (vm_insnhelper.c:6518)
==30828==    by 0×34F3A3: vm_exec_core (insns.def:1495)
==30828==    by 0×33C8CC: vm_exec_loop (vm.c:2516)
==30828==    by 0×33C8CC: rb_vm_exec (vm.c:2492)
==30828==    by 0×146E44: rb_ec_exec_node (eval.c:283)
==30828==    by 0×148E41: ruby_run_node (eval.c:323)
==30828==    by 0×1438F6: rb_main (main.c:39)
==30828==    by 0×1438F6: main (main.c:58)
==30828==
==30828== 32 bytes in 1 blocks are definitely lost in loss record 2 of 11
==30828==    at 0×48487EE: malloc (vg_replace_malloc.c:442)
```

```
==30828==    by 0x34F3A3: vm_exec_core (insns.def:1495)
==30828==    by 0x33C8CC: vm_exec_loop (vm.c:2516)
==30828==    by 0x33C8CC: rb_vm_exec (vm.c:2492)
==30828==    by 0x146E44: rb_ec_exec_node (eval.c:283)
==30828==    by 0x148E41: ruby_run_node (eval.c:323)
==30828==    by 0x1438F6: rb_main (main.c:39)
==30828==    by 0x1438F6: main (main.c:58)
==30828==
==30828== LEAK SUMMARY:
==30828==    definitely lost: 30,784 bytes in 3 blocks
==30828==    indirectly lost: 0 bytes in 0 blocks
==30828==      possibly lost: 608 bytes in 1 blocks
==30828==    still reachable: 9,342 bytes in 71 blocks
==30828==         suppressed: 0 bytes in 0 blocks
==30828== Reachable blocks (those to which a pointer was found) are not shown.
==30828== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==30828==
==30828== For lists of detected and suppressed errors, rerun with: -s
==30828== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 0 from 0)
```



```
$ RUBY_FREE_AT_EXIT=1 leaks -q --atExit -- ruby test.rb

leaks Report Version: 4.0, multi-line stacks
Process 6479: 563 nodes malloced for 7887 KB
Process 6479: 3 leaks for 7897184 total leaked bytes.

STACK OF 1 INSTANCE OF 'ROOT LEAK: <realloc in match_at>':
11  dyld                                    0×190e760e0 start + 2360
10  ruby                                    0×1045702c8 main + 104  main.c:58
9   ruby                                    0×10461a5cc ruby_run_node + 68  eval.c:323
8   ruby                                    0×10461a6b4 rb_ec_exec_node + 160  eval.c:287
7   ruby                                    0×1047a3a2c rb_vm_exec + 492  vm.c:0
6   ruby                                    0×1047aa03c vm_exec_core + 21660  insns.def:1497
5   ruby                                    0×104700860 rb_reg_match + 152  re.c:3641
4   ruby                                    0×1046ff130 rb_reg_search_set_match + 444  re.c:1752
```

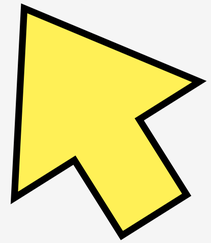
```
$ RUBY_FREE_AT_EXIT=1 leaks -q --atExit -- ruby test.rb

leaks Report Ve[illegible] 4.0, multi-line stacks
Process 6479: 5[illegible]odes malloced for 7887 KB
Process 6479: 3 leaks for 7897184 total leaked bytes.

STACK OF 1 INSTANCE OF 'ROOT LEAK: <realloc in match_at>':
11  dyld                                    0×190e760e0 start + 2360
10  ruby                                    0×1045702c8 main + 104  main.c:58
9   ruby                                    0×10461a5cc ruby_run_node + 68  eval.c:323
8   ruby                                    0×10461a6b4 rb_ec_exec_node + 160  eval.c:287
7   ruby                                    0×1047a3a2c rb_vm_exec + 492  vm.c:0
6   ruby                                    0×1047aa03c vm_exec_core + 21660  insns.def:1497
5   ruby                                    0×104700860 rb_reg_match + 152  re.c:3641
4   ruby                                    0×1046ff130 rb_reg_search_set_match + 444  re.c:1752
3   ruby                                    0×10471c84c onig_search_gpos + 1684  regexec.c:5423
2   ruby                                    0×104718030 match_at + 21968  regexec.c:3749
1   libsystem_malloc.dylib                  0×1910397e8 _realloc + 468
```

```
$ RUBY_FREE_AT_EXIT=1 leaks -q --atExit -- ruby test.rb

leaks Report Version: 4.0, multi-line stacks
Process 6479: 563 nodes malloced for 7887 KB
Process 6479: 3 leaks for 7897184 total leaked bytes.

STACK OF 1 INSTANCE OF 'ROOT LEAK: <realloc in match_at>':
11  dyld                                0x190e760e0 start + 2360
10  ruby                                0x1045702c8 main + 104  main.c:58
9   ruby                                0x10461a5cc ruby_run_node + 68  eval.c:323
8   ruby                                0x10461a6b4 rb_ec_exec_node + 160  eval.c:287
7   ruby                                0x1047a3a2c rb_vm_exec + 492  vm.c:0
6   ruby                                0x1047aa03c vm_exec_core + 21660  insns.def:1497
5   ruby                                0x104700860 rb_reg_match + 152  re.c:3641
4   ruby                                0x1046ff130 rb_reg_search_set_match + 444  re.c:1752
3   ruby                                0x10471c84c onig_search_gpos + 1684  regexec.c:5423
2   ruby                                0x104718030 match_at + 21968  regexec.c:3749
1   libsystem_malloc.dylib              0x1910397e8 _realloc + 468
0   libsystem_malloc.dylib              0x191038fb0 _malloc_zone_realloc + 144
====
    1 (7.53M) ROOT LEAK: <realloc in match_at 0x130218000> [7897088]

STACK OF 1 INSTANCE OF 'ROOT LEAK: <malloc in onig_region_resize>':
10  dyld                                0x190e760e0 start + 2360
9   ruby                                0x10048c2c8 main + 104  main.c:58
```

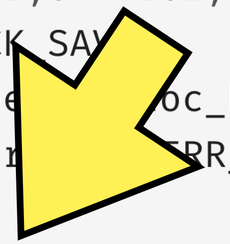
```
7   ruby                                0×1005366b4 rb_ec_exec_node + 160  eval.c:287
6   ruby                                0×1006bfa2c rb_vm_exec + 492  vm.c:0
5   ruby                                0×1006c603c vm_exec_core + 21660  insns.def:1497
4   ruby                                0×10061c860 rb_reg_match + 152  re.c:3641
3   ruby                                0×10061b130 rb_reg_search_set_match + 444  re.c:1752
2   ruby                                0×10063821c onig_search_gpos + 100  regexec.c:5175
1   ruby                                0×10062e608 onig_region_resize + 140  regexec.c:894
0   libsystem_malloc.dylib              0×191038a68 _malloc_zone_malloc_instrumented_or_legacy + 148
═══
    1 (48 bytes) ROOT LEAK: <malloc in onig_region_resize 0×11fe04930> [48]
```

```diff
diff --git a/re.c b/re.c
index 2f515779c7c6d4..9c474492711519 100644
--- a/re.c
+++ b/re.c
@@ -88,6 +88,9 @@ static const char casetable[] = {
 # error >>> "You lose. You will need a translation table for your character set." <<<
 #endif

+// The process-global timeout for regexp matching
+rb_hrtime_t rb_reg_match_time_limit = 0;
+
 int
 rb_memcicmp(const void *x, const void *y, long len)
 {
@@ -1732,6 +1735,23 @@ reg_onig_search(regex_t *reg, VALUE str, struct re_registers *regs, void *args_p
         ONIG_OPTION_NONE);
 }

+struct rb_reg_onig_match_args {
+    VALUE re;
+    VALUE str;
+    struct reg_onig_search_args args;
+    struct re_registers regs;
+
+    OnigPosition result;
+};
+
+static VALUE
+rb_reg_onig_match_try(VALUE value_args)
+{
+    struct rb_reg_onig_match_args *args = (struct rb_reg_onig_match_args *)value_args;
+    args->result = rb_reg_onig_match(args->re, args->str, reg_onig_search, &args->args, &args->regs);
+    return Qnil;
+}
+
 /* returns byte offset */
 static long
 rb_reg_search_set_match(VALUE re, VALUE str, long pos, int reverse, int set_backref_str, VALUE
*set_match)
@@ -1742,22 +1762,38 @@ rb_reg_search_set_match(VALUE re, VALUE str, long pos, int reverse, int
set_back
         return -1;
     }

-    struct reg_onig_search_args args = {
-        .pos = pos,
-        .range = reverse ? 0 : len,
+    struct rb_reg_onig_match_args args = {
+        .re = re,
+        .str = str,
+        .args = {
+            .pos = pos,
+            .range = reverse ? 0 : len,
+        },
+        .regs = {0}
     };

-    struct re_registers regs = {0};
+    /* If there is a timeout set, then rb_reg_onig_match could raise a
+     * Regexp::TimeoutError so we want to protect it from leaking memory. */
+    if (rb_reg_match_time_limit) {
+        int state;
+        rb_protect(rb_reg_onig_match_try, (VALUE)&args, &state);
+        if (state) {
+            onig_region_free(&args.regs, false);
+            rb_jump_tag(state);
+        }
+    }
+    else {
+        rb_reg_onig_match_try((VALUE)&args);
+    }

-    OnigPosition result = rb_reg_onig_match(re, str, reg_onig_search, &args, &regs);
-    if (result == ONIG_MISMATCH) {
+    if (args.result == ONIG_MISMATCH) {
         rb_backref_set(Qnil);
         return ONIG_MISMATCH;
     }

     VALUE match = match_alloc(rb_cMatch);
     rb_matchext_t *rm = RMATCH_EXT(match);
-    rm->regs = regs;
+    rm->regs = args.regs;

     if (set_backref_str) {
         RB_OBJ_WRITE(match, &RMATCH(match)->str, rb_str_new4(str));
@@ -1774,7 +1810,7 @@ rb_reg_search_set_match(VALUE re, VALUE str, long pos, int reverse, int set_back
     rb_backref_set(match);
     if (set_match) *set_match = match;

-    return result;
+    return args.result;
 }

 long
@@ -4601,12 +4637,9 @@ re_warn(const char *s)
     rb_warn("%s", s);
 }

-// The process-global timeout for regexp matching
-rb_hrtime_t rb_reg_match_time_limit = 0;
-
 // This function is periodically called during regexp matching
-void
-rb_reg_check_timeout(regex_t *reg, void *end_time_)
+bool
+rb_reg_timeout_p(regex_t *reg, void *end_time_)
 {
     rb_hrtime_t *end_time = (rb_hrtime_t *)end_time_;

@@ -4631,10 +4664,18 @@ rb_reg_check_timeout(regex_t *reg, void *end_time_)
     }
     else {
         if (*end_time < rb_hrtime_now()) {
-            // timeout is exceeded
-            rb_raise(rb_eRegexpTimeoutError, "regexp match timeout");
+            // Timeout has exceeded
+            return true;
         }
     }
+
+    return false;
+}
+
+void
+rb_reg_raise_timeout(void)
+{
+    rb_raise(rb_eRegexpTimeoutError, "regexp match timeout");
 }

 /*
diff --git a/regexec.c b/regexec.c
index 150d36ccc89745..81d0ea6a6b0109 100644
--- a/regexec.c
+++ b/regexec.c
@@ -2293,7 +2293,7 @@ match_at(regex_t* reg, const UChar* str, const UChar* end,
   UChar *pkeep;
   char *alloca_base;
   char *xmalloc_base = NULL;
-  OnigStackType *stk_alloc, *stk_base, *stk, *stk_end;
+  OnigStackType *stk_alloc, *stk_base = NULL, *stk, *stk_end;
   OnigStackType *stkp; /* used as any purpose. */
   OnigStackIndex si;
   OnigStackIndex *repeat_stk;
@@ -4202,6 +4202,11 @@ match_at(regex_t* reg, const UChar* str, const UChar* end,
   STACK_SAVE;
   xfree(xmalloc_base);
   return ONIGERR_UNEXPECTED_BYTECODE;
+
+ timeout:
+  xfree(xmalloc_base);
+  xfree(stk_base);
+  HANDLE_REG_TIMEOUT_IN_MATCH_AT;
 }


diff --git a/regint.h b/regint.h
index 034a31426c819f..57cbb81654faf3 100644
--- a/regint.h
+++ b/regint.h
@@ -154,13 +154,18 @@
 #ifdef RUBY

 # define CHECK_INTERRUPT_IN_MATCH_AT do { \
-  msa->counter++;                         \
-  if (msa->counter >= 128) {              \
-    msa->counter = 0;                     \
-    rb_reg_check_timeout(reg, &msa->end_time);  \
-    rb_thread_check_ints();               \
-  }                                       \
+  msa->counter++; \
+  if (msa->counter >= 128) { \
+    msa->counter = 0; \
+    if (rb_reg_timeout_p(reg, &msa->end_time)) { \
+      goto timeout; \
+    } \
+    rb_thread_check_ints(); \
+  } \
 } while(0)
+# define HANDLE_REG_TIMEOUT_IN_MATCH_AT do { \
+  rb_reg_raise_timeout(); \
+} while (0)
 # define onig_st_init_table                  st_init_table
 # define onig_st_init_table_with_size        st_init_table_with_size
 # define onig_st_init_numtable               st_init_numtable
@@ -996,7 +1001,8 @@ extern int onig_st_insert_strend(hash_table_type* table, const UChar* str_key, c
 #ifdef RUBY
 extern size_t onig_memsize(const regex_t *reg);
 extern size_t onig_region_memsize(const struct re_registers *regs);
-void rb_reg_check_timeout(regex_t *reg, void *end_time);
+bool rb_reg_timeout_p(regex_t *reg, void *end_time);
+NORETURN(void rb_reg_raise_timeout(void));
 #endif

 RUBY_SYMBOL_EXPORT_END
diff --git a/test/ruby/test_regexp.rb b/test/ruby/test_regexp.rb
index 2de622f9a5e7d0..e6eee24b2e6b9d 100644
--- a/test/ruby/test_regexp.rb
+++ b/test/ruby/test_regexp.rb
@@ -1807,6 +1807,23 @@ def test_s_timeout_corner_cases
     end;
   end

+  def test_s_timeout_memory_leak
+    assert_no_memory_leak([], "#{<<~"begin;"}", "#{<<~"end;"}", "[Bug #20228]", rss: true)
+      Regexp.timeout = 0.001
+      regex = /^(a*)*$/
+      str = "a" * 1000000 + "x"
+
+      code = proc do
+        regex =~ str
+      rescue
+      end
+
+      10.times(&code)
+    begin;
+      1_000.times(&code)
+    end;
+  end
+
   def per_instance_redos_test(global_timeout, per_instance_timeout, expected_timeout)
     assert_separately([], "#{<<-"begin;"}\n#{<<-'end;'}")
       global_timeout = #{ EnvUtil.apply_timeout_scale(global_timeout).inspect }
```

```
-
 // This function is periodically called during regexp matching
-void
-rb_reg_check_timeout(regex_t *reg, void *end_time_)
+bool
+rb_reg_timeout_p(regex_t *reg, void *end_time_)
 {
     rb_hrtime_t *end_time = (rb_hrtime_t *)end_time_;

@@ -4631,10 +4664,18 @@ rb_reg_check_timeout(regex_t *reg, void *end_time_)
     }
     else {
         if (*end_time < rb_hrtime_now()) {
-            // timeout is exceeded
-            rb_raise(rb_eRegexpTimeoutError, "regexp match timeout");
+            // Timeout has exceeded
+            return true;
         }
     }
+
+    return false;
+}
+
+void
+rb_reg_raise_timeout(void)
+{
```



```
-  OnigStackType *stk_alloc, *stk_base, *stk, *stk_end;
+  OnigStackType *stk_alloc, *stk_base = NULL, *stk, *stk_end;
   OnigStackType *stkp; /* used as any purpose. */
   OnigStackIndex si;
   OnigStackIndex *repeat_stk;
@@ -4202,6 +4202,11 @@ match_at(regex_t* reg, const UChar* str, const UChar* end,
   STACK_SA
   xfree       oc_base);
   retur       RR_UNEXPECTED_BYTECODE;
+
+ timeout:
+  xfree(xmalloc_base);
+  xfree(stk_base);
+  HANDLE_REG_TIMEOUT_IN_MATCH_AT;
  }


diff --git a/regint.h b/regint.h
index 034a31426c819f..57cbb81654faf3 100644
--- a/regint.h
+++ b/regint.h
@@ -154,13 +154,18 @@
 #ifdef RUBY

 # define CHECK_INTERRUPT_IN_MATCH_AT do { \
-  msa->counter++;                         \
+                                          \
```



How You Can Use This Feature

Find memory leaks in native gems

https://github.com/Shopify/ruby_memcheck

ruby_memcheck Public

Edit Pins | Unwatch 266 | Fork 11 | Starred 188

main | Branches | Tags | Go to file t | Add file | Code

peterzhu2118 Bump version to 2.3.0 — 274b95f · 4 months ago — 123 Commits

.github/workflows	Run apt-get update	7 months ago
exe	Add running on a Ruby script	8 months ago
lib	Bump version to 2.3.0	4 months ago
suppressions	Suppress Regexps created in date__parse	8 months ago
test	Set RUBY_FREE_AT_EXIT for Ruby >= 3.3.0	4 months ago
.gitignore	[ci skip] Ignore .DS_Store	7 months ago
.rubocop.yml	Skip function stack_chunk_alloc	2 years ago
Gemfile	Add rubocop	3 years ago
LICENSE.txt	Update LICENSE.txt	10 months ago
README.md	Add binary_name back	7 months ago
Rakefile	Automatically detect Ruby native extensions	10 months ago
ruby_memcheck.gemspec	Fix the homepage in `gemspec`	10 months ago

README | Code of conduct | MIT license | Security

ruby_memcheck

This gem provides a sane way to use Valgrind's memcheck on your native extension gem.

Table of contents

1. What is this gem?
   i. Who should use this gem?
   ii. How does it work?

About

Use Valgrind memcheck on your native gem without going crazy

ruby | valgrind

Readme
MIT license
Code of conduct
Security policy
Activity
Custom properties
188 stars
266 watching
11 forks
Report repository

Releases 24

2.3.0 Latest
on Dec 21, 2023

+ 23 releases

Used by 184

Contributors 5

Languages

Ruby 90.1% C 9.9%

ruby_memcheck found memory leaks in nokogiri, liquid-c, protobuf, gRPC

Find native level memory leaks in your app

Thank you!

blog.peterzhu.ca

peter@peterzhu.ca

@peterzhu2118

@peterzhu2118@ruby.social

hparker.xyz

adamhess1991@gmail.com

@theHessParker

@hparker@ruby.social