Ruby master - Bug #14357
thread_safe tests suite segfaults
01/15/2018 10:00 AM - vo.x (Vít Ondruch)

Status: Closed
Priority: Normal
Assignee:
Target version:
Backport: 2.3: REQUIRED, 2.4: DONE, 2.5: DONE

Description
The thread_safe gem is not maintained anymore, but I don't see any reason why its test suite should segfault with Ruby 2.5.

+ rspec -Ilib spec
Randomized with seed 62008
............./builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/support/threadsafe_test.rb:44: [BUG] Segmentation fault at 0x000000000000000c
-- Control frame information -----------------------------------------------
c:0011 p:0009 s:0073 e:000070 METHOD /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/support/threadsafe_test.rb:44 [FINISH]
c:0010 p:0008 s:0066 e:000065 METHOD /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/lib/thread_safe/non_concurrent_cache_backend.rb:1

c:0009 p:0006 s:0061 e:000060 METHOD /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/lib/thread_safe/mri_cache_backend.rb:17

c:0008 p:0011 s:0055 e:000054 METHOD /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/lib/thread_safe/cache.rb:72

c:0007 p:0033 s:0047 e:000046 METHOD /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/thread_safe/cache_loops_spec.rb:435

c:0006 p:0020 s:0027 e:000026 METHOD /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/thread_safe/cache_loops_spec.rb:435

c:0005 p:---- s:0031 e:000030 CFUNC :times

c:0004 p:0020 s:0027 e:000026 METHOD /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/thread_safe/cache_loops_spec.rb:435

c:0003 p:0082 s:0017 e:000016 METHOD /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/thread_safe/cache_loops_spec.rb:397

c:0002 p:0025 s:0006 e:000005 BLOCK /build/dir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/thread_safe/cache_loops_spec.rb:378 [FINISH]
c:0001 p:---- s:0003 e:000002 (none) [FINISH]
-- Ruby level backtrace information ----------------------------------------
/builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/thread_safe/cache_loops_spec.rb:378:in 'block (2 levels) in run_thread_loop'
/builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/thread_safe/cache_loops_spec.rb:397:in 'setup_sync_and_start_loop'
/builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/threadsafe_test.rb:44:in `put_ifAbsent_loop_outer_multiple_keys'
/builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/threadsafe_test.rb:44:in `times'
/builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/threadsafe_test.rb:44:in `block in put_ifAbsent_loop_outer_multiple_keys'
/builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/spec/threadsafe_test.rb:44:in `put_ifAbsent_loop_outer_multiple_keys'
/builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/lib/thread_safe/cache_backend.rb:17:in `compute_if_absent'
/builddir/build/BUILD/thread_safe-0.3.6/usr/share/gems/gems/thread_safe-0.3.6/lib/thread_safe/non_concurrent_cache_backend.rb:12:in `[]'
/builddir/build/BUILD/thread_safe-0.3.6/threadsafe_test.rb:44:in `eql?'
-- Machine register context -----------------------------------------------
RIP: 0x00000ff7c631f030f RBP: 0x00000ff7c5ace000
RAX: 0x0000000000000000  RBX: 0x00007f7c634eaee0  RCX: 0x0000000000000000
RDX: 0x0000000000000000  RSI: 0x00005577aecc5bd8  RDI: 0x00005577af09a890
R8: 0x00005577aecc5bb0  R9: 0x0000000000000000  R10: 0x0000000000000000
R11: 0x0000000000000000  R12: 0x00005577aecc5bd8  R13: 0x00005577af09a890
R14: 0x00005577af14e128  R15: 0x00007f7c5adece00  EFL: 0x0000000000010293
-- C level backtrace information -------------------------------------------
/lib64/libruby.so.2.5(0x7f7c6320c5f5) [0x7f7c6320c5f5]
/lib64/libruby.so.2.5(0x7f7c6320c82c) [0x7f7c6320c82c]
/lib64/libruby.so.2.5(0x7f7c630d7b34) [0x7f7c630d7b34]
/lib64/libruby.so.2.5(0x7f7c6319cfe2) [0x7f7c6319cfe2]
/lib64/libpthread.so.0(0x7f7c62e3a160) [0x7f7c62e3a160]
/lib64/libruby.so.2.5(0x7f7c631f030f) [0x7f7c631f030f]
/lib64/libruby.so.2.5(0x7f7c631fa208) [0x7f7c631fa208]
/lib64/libruby.so.2.5(0x7f7c631ff674) [0x7f7c631ff674]
/lib64/libruby.so.2.5(0x7f7c63203cad) [0x7f7c63203cad]
/lib64/libruby.so.2.5(0x7f7c63204d9a) [0x7f7c63204d9a]
/lib64/libruby.so.2.5(0x7f7c6320557e) [0x7f7c6320557e]
/lib64/libruby.so.2.5(rb_eql+0x6c) [0x7f7c63134adc]
/lib64/libruby.so.2.5(0x7f7c630f71a1) [0x7f7c630f71a1]
/lib64/libruby.so.2.5(0x7f7c631a574a) [0x7f7c631a574a]
/lib64/libruby.so.2.5(st_lookup+0x4b) [0x7f7c631a64cb]
/lib64/libruby.so.2.5(krb_hash_aref+0x1f) [0x7f7c630f9dde]
/lib64/libruby.so.2.5(0x7f7c631f89d2) [0x7f7c631f89d2]
/lib64/libruby.so.2.5(0x7f7c631ff674) [0x7f7c631ff674]
/lib64/libruby.so.2.5(0x7f7c63201234) [0x7f7c63201234]
/lib64/libruby.so.2.5(0x7f7c631283b0) [0x7f7c631283b0]
/lib64/libruby.so.2.5(0x7f7c631f4e99) [0x7f7c631f4e99]
/lib64/libruby.so.2.5(0x7f7c631f9815) [0x7f7c631f9815]
/lib64/libruby.so.2.5(0x7f7c631ff674) [0x7f7c631ff674]
/lib64/libruby.so.2.5(0x7f7c63202860) [0x7f7c63202860]
/lib64/libruby.so.2.5(0x7f7c6320299c) [0x7f7c6320299c]
/lib64/libruby.so.2.5(0x7f7c631cda51) [0x7f7c631cda51]
/lib64/libruby.so.2.5(0x7f7c631cdebb3) [0x7f7c631cdebb3]
/lib64/libpthread.so.0(0x7f7c62e3a160) [0x7f7c62e3a160]
/lib64/libc.so.6(clone+0x3f) [0x7f7c623950ef]
-- Other runtime information ------------------------------------------------
* Loaded script: /usr/bin/rspec
* Loaded features:
  0   enumerator.so
  1   thread.rb
  2   rational.so
  3   complex.so
  4   /usr/lib64/ruby/enc/encdb.so
  5   /usr/lib64/ruby/enc/trans/tdnbsd.so
  6   /usr/lib64/ruby/rbconfig.rb
  7   /usr/share/rubygems/rubygems/compatibility.rb
  8   /usr/share/rubygems/rubygems.defaults.rb
  9   /usr/share/rubygems/rubygems/deprecate.rb
 10   /usr/share/rubygems/rubygems/errors.rb
 11   /usr/share/rubygems/rubygems/version.rb
 12   /usr/share/rubygems/rubygems/requirement.rb
 13   /usr/share/rubygems/rubygems/platform.rb
 14   /usr/share/rubygems/rubygems/basic_specification.rb
 15   /usr/share/rubygems/rubygems/stub_specification.rb
 16   /usr/share/rubygems/rubygems/util/list.rb
 17   /usr/lib64/ruby/stringio.so
 18   /usr/share/rubygems/rubygems/specification.rb
 19   /usr/share/rubygems/rubygems/exceptions.rb
 20   /usr/share/rubygems/rubygems/defaults/operating_system.rb
 21   /usr/share/rubygems/rubygems/dependency.rb
 22   /usr/share/rubygems/rubygems/core_ext/kernel_gem.rb
 23   /usr/share/ruby/ruby.rubygems
 24   /usr/share/rubygems/rubygems/core_ext/kernel_require.rb
 25   /usr/share/rubygems/rubygems.rb
 26   /usr/share/rubygems/rubygems/path_support.rb
 27   /usr/share/rubygems/rubygems/bundler_version_finder.rb
 28   /usr/share/ruby/tsort.rb
You may have encountered a bug in the Ruby interpreter or extension libraries. Bug reports are welcome.

For details: http://www.ruby-lang.org/bugreport.html

Associated revisions

Revision 62396 - 02/13/2018 10:02 AM - normalperson (Eric Wong)

Calling the eql? and .hash methods during a Hash operation can result in a thread switch or a signal handler to run: allowing one execution context to rebuild the hash table while another is still reading or writing the table. This results in a use-after-free bug affecting the thread_safe-0.3.6 test suite and likely other bugs.

This bug did not affect users of commonly keys (String, Symbol, Fixnum) as those are optimized to avoid method dispatch for eql? and .hash methods.

A separate version of this change needs to be ported to Ruby 2.3.x which had a different implementation of st.c but was affected by the same bug.


Thanks to Vit Ondruch for reporting the bug.

From: Vladimir Makarov vmakarov@redhat.com

Revision ba81ac79 - 03/20/2018 09:37 AM - naruse (Yui NARUSE)

merge revision(s) 62396: [Backport #14357]

st.c: retry operations if rebuilt

Calling the eql? and .hash methods during a Hash operation can result in a thread switch or a signal handler to run: allowing one execution context to rebuild the hash table while another is still reading or writing the table. This results in a use-after-free bug affecting the thread_safe-0.3.6 test suite and likely other bugs.

This bug did not affect users of commonly keys (String, Symbol, Fixnum) as those are optimized to avoid method dispatch for eql? and .hash methods.

A separate version of this change needs to be ported to Ruby 2.3.x which had a different implementation of st.c but was affected by the same bug.

* st.c: Add comment about table rebuilding during comparison.
Calling the .eql? and .hash methods during a Hash operation can result in a thread switch or a signal handler to run: allowing one execution context to rebuild the hash table while another is still reading or writing the table. This results in a use-after-free bug affecting the thread_safe-0.3.6 test suite and likely other bugs.

This bug did not affect users of commonly keys (String, Symbol, Fixnum) as those are optimized to avoid method dispatch for .eql? and .hash methods.

A separate version of this change needs to be ported to Ruby 2.3.x which had a different implementation of st.c but was affected by the same bug.

* st.c: Add comment about table rebuilding during comparison.

Thanks to Vit Ondruch for reporting the bug.

From: Vladimir Makarov <vmakarov@redhat.com>
for .eql? and .hash methods.

A separate version of this change needs to be ported to Ruby 2.3.x which had a different implementation of st.c but was affected by the same bug.

* st.c: Add comment about table rebuilding during comparison.
  (DO_PTR_EQUAL_CHECK): New macro.
  (REBUILT_TABLE_ENTRY_IND, REBUILT_TABLE_BIN_IND): New macros.
  (find_entry, find_table_entry_ind, find_table_bin_ind): Use new macros. Return the rebuild flag.
  (find_table_bin_ptr_and_reserve): Ditto.
  (st_lookup, st_get_key, st_insert, st_insert2): Retry the operation if the table was rebuilt.
  (st_rehash_linear, st_rehash_indexed): Use DO_PTR_EQUAL_CHECK. Return the rebuild flag.
  (st_rehash): Retry the operation if the table was rebuilt.

[ruby-core:85510] [Ruby trunk Bug#14357]

Thanks to Vit Ondruch for reporting the bug.

From: Vladimir Makarov <vmakarov@redhat.com>

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_4@63805 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63805 - 06/30/2018 12:56 PM - usa (Usaku NAKAMURA)
merge revision(s) 62396: [Backport #14357]

st.c: retry operations if rebuilt

Calling the .eql? and .hash methods during a Hash operation can result in a thread switch or a signal handler to run: allowing one execution context to rebuild the hash table while another is still reading or writing the table. This results in a use-after-free bug affecting the thread_safe-0.3.6 test suite and likely other bugs.

This bug did not affect users of commonly keys (String, Symbol, Fixnum) as those are optimized to avoid method dispatch for .eql? and .hash methods.

A separate version of this change needs to be ported to Ruby 2.3.x which had a different implementation of st.c but was affected by the same bug.

* st.c: Add comment about table rebuilding during comparison.
  (DO_PTR_EQUAL_CHECK): New macro.
  (REBUILT_TABLE_ENTRY_IND, REBUILT_TABLE_BIN_IND): New macros.
  (find_entry, find_table_entry_ind, find_table_bin_ind): Use new macros. Return the rebuild flag.
  (find_table_bin_ptr_and_reserve): Ditto.
  (st_lookup, st_get_key, st_insert, st_insert2): Retry the operation if the table was rebuilt.
  (st_rehash_linear, st_rehash_indexed): Use DO_PTR_EQUAL_CHECK. Return the rebuild flag.
  (st_rehash): Retry the operation if the table was rebuilt.

[ruby-core:85510] [Ruby trunk Bug#14357]

Thanks to Vit Ondruch for reporting the bug.

From: Vladimir Makarov <vmakarov@redhat.com>

History
#1 - 02/03/2018 08:51 PM - normalperson (Eric Wong)

v.ondruch@tiscali.cz wrote:
https://bugs.ruby-lang.org/issues/14357

The thread_safe gem is not maintained anymore, but I don’t see
Right, no 3rd-party C exts loaded and I hit this in trunk, too.

Using -fsanitize=address reveals use-after-free in st.c

Investigating, but maybe Vladimir can find it sooner.

Randomized with seed 18515

---

=18224=ERROR: AddressSanitizer: heap-use-after-free on address 0x6230002112c0 at pc 0x557ae852ae33 bp 0x7fb3c088f5c0 sp 0x7b3c088f5b8

READ of size 8 at 0x6230002112c0 thread T332 (cache_loops_sp*)

#0 0x557ae852ae33 in find_table_entry_ind ../st.c:873
#1 0x557ae852ae34 in st_lookup ../st.c:1049
#2 0x557ae831139e in rb_hash_aref ../hash.c:853
#3 0x557ae8648e27 in vm_opt_aref ../vm_insnhelper.c:3650
#4 0x557ae8648e27 in vm_exec_core $SRC/ruby/insns.def:1175
#5 0x557ae8651696 in vm_exec ../vm.c:1791
#6 0x557ae8564272 in invoke_block ../vm.c:994
#7 0x557ae865272 in invoke_iseq_block_from_c ../vm.c:1046
#8 0x557ae8669c22 in invoke_block_from_c_bh ../vm.c:1064
#9 0x557ae8669c22 in vm_yield ../vm.c:1109
#10 0x557ae8669c22 in rb_yield ../vm.c:970
#11 0x557ae8669c22 in rb_yield_1 ../vm.c:976
#12 0x557ae8639a95 in int_dotimes ../numeric.c:4984
#13 0x557ae862da57 in vm_call_cfunc_with_frame ../vm_insnhelper.c:1921
#14 0x557ae862da57 in vm_call_cfunc ../vm_insnhelper.c:1937
#15 0x557ae866213 in vm_exec_core $SRC/ruby/insns.def:719
#16 0x557ae8651696 in vm_exec ../vm.c:1791
#17 0x557ae8654272 in invoke_block ../vm.c:994
#18 0x557ae8654272 in invoke_iseq_block_from_c ../vm.c:1046
#19 0x557ae8658126 in invoke_block_from_c_proc ../vm.c:1139
#20 0x557ae8658126 in vm_invoke_proc ../vm.c:1157
#21 0x557ae8658126 in rb_vm_invoke_proc ../vm.c:1178
#22 0x557ae85a95e3 in thread_do_start ../thread.c:603
#23 0x557ae85a95e3 in thread_start_func_2 ../thread.c:647
#24 0x557ae85a8680 in thread_start_func_1 ../thread_pthread.c:872
#25 0x7fb3d2b60663 in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x8063)
#26 0x7fb3d231662c in clone (/lib/x86_64-linux-gnu/libc.so.6+0xede2c)

0x6230002112c0 is located 2496 bytes inside of 6144-byte region [0x623000210900,0x623000212100) freed by thread T343 (cache_loops_sp*) here:

#0 0x7fb3d2222527 in __interceptor_free (/usr/lib/x86_64-linux-gnu/libasan.so.1+0x54527)
#1 0x557ae8303060 in objspace_xfree ../gc.c:7987
#2 0x557ae8303060 in ruby_sized_xfree ../gc.c:8082
#3 0x557ae8303060 in ruby_xfree ../gc.c:8089

previously allocated by thread T331 (cache_loops_sp*) here:

#0 0x7fb3d322273f in malloc (/usr/lib/x86_64-linux-gnu/libasan.so.1+0x5473f)
#1 0x557ae82fd573 in objspace_xmalloc0../gc.c:7297

Thread T332 (cache_loops_sp*) created by T0 here:

#0 0x7fb3d3111bb in pthread_create (/usr/lib/x86_64-linux-gnu/libasan.so.1+0x54949d)
#1 0x557ae859489d in native_thread_create ../thread_pthread.c:1008
#2 0x557ae859489d in thread_create_core ../thread.c:757
#3 0x557ae884894c ($SRC/ruby/a/i/bin/ruby+0x63f94c)

Thread T343 (cache_loops_sp*) created by T0 here:

#0 0x7fb3d3111bb in pthread_create (/usr/lib/x86_64-linux-gnu/libasan.so.1+0x54949d)
#1 0x557ae859489d in native_thread_create ../thread_pthread.c:1008
#2 0x557ae859489d in thread_create_core ../thread.c:757
#3 0x557ae884894c ($SRC/ruby/a/i/bin/ruby+0x63f94c)

Thread T331 (cache_loops_sp*) created by T0 here:

#0 0x7fb3d3111bb in pthread_create (/usr/lib/x86_64-linux-gnu/libasan.so.1+0x54949d)
#1 0x557ae859489d in native_thread_create ../thread_pthread.c:1008
#2 0x557ae859489d in thread_create_core ../thread.c:757
#3 0x557ae884894c ($SRC/ruby/a/i/bin/ruby+0x63f94c)

SUMMARY: AddressSanitizer: heap-use-after-free ../st.c:873 find_table_entry_ind

Shadow bytes around the buggy address:
0x0c468003a200: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
0x0c468003a210: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd

03/15/2022
Using -fsanitize=address reveals use-after-free in st.c. Investigating, but maybe Vladimir can find it sooner.

ST_DEBUG doesn't reveal anything nor does switching to QUADRATIC_PROBE. Waiting for valgrind which is very slow and I need to go afk to do other things...

Could be a GC problem related to coverage (also [Bug #14334]) since there's minimal changes in st.c from v2_4_0..v2_5_0 and I'm not hitting it with the 2.4 branch.

Nothing jumped out at me from reading diffs, I don't think the guard I proposed in [ruby-core:85352] is effective, even

Bisecting now, and going mostly away from computers for a bit...

git bisect start v2_5_0 v2_4_0
git bisect run /tmp/bisect.sh

---

#2 - 02/03/2018 11:21 PM - normalperson (Eric Wong)
Eric Wong wrote:

Using -fsanitize=address reveals use-after-free in st.c. Investigating, but maybe Vladimir can find it sooner.

ST_DEBUG doesn't reveal anything nor does switching to QUADRATIC_PROBE. Waiting for valgrind which is very slow and I need to go afk to do other things...

#3 - 02/04/2018 05:21 AM - normalperson (Eric Wong)
Eric Wong wrote:

ST_DEBUG doesn't reveal anything nor does switching to QUADRATIC_PROBE. Waiting for valgrind which is very slow and I need to go afk to do other things...

Could be a GC problem related to coverage (also [Bug #14334]) since there's minimal changes in st.c from v2_4_0..v2_5_0 and I'm not hitting it with the 2.4 branch.

Nothing jumped out at me from reading diffs, I don't think the guard I proposed in [ruby-core:8532] is effective, even

Bisecting now, and going mostly away from computers for a bit...
Could be a GC problem related to coverage (also [Bug #14334](https://bugs.ruby-lang.org/issues/14334)) since there's minimal changes in st.c from v2_4_0..v2_5_0 and I'm not hitting it with the 2.4 branch.

Or not... Hard to tell if I'm chasing multiple bugs, here.

Could be a GC problem related to coverage (also [Bug #14334](https://bugs.ruby-lang.org/issues/14334))

Nope, not a fault of coverage. Commented simplecov and coveralls from spec/spec_helper.rb in thread_safe-0.3.6 and asan still chokes...

The thread_safe gem is not maintained anymore, but I don't see any reason why its test suite should segfault with Ruby 2.5.

Right, no 3rd-party C exts loaded and I hit this in trunk, too. Using -fsanitize=address reveals use-after-free in st.c Investigating, but maybe Vladimir can find it sooner.

Maybe my initial investigation was correct, after all.

valgrind takes forever, but indicates the free is caused by rebuild_table; so it doesn't look like we missed GC marking during rebuild. Disabling the free(tab->entries) at line st.c:792 (patch below) seems to indicate success with the thread_safe test suite (letting it loop overnight).

Looks like the new_tab != tab case of rebuild is leaving a hanging reference somewhere.
Following patch hides the problem by introducing a leak:

```c
--- a/st.c
+++ b/st.c
@@ -789,7 +789,7 @@ rebuild_table(st_table *tab)
     if (tab->bins != NULL)
         free(tab->bins);
     tab->bins = new_tab->bins;
-    free(tab->entries);
+    /* free(tab->entries); */ /* NOT FOR PRODUCTION USE */
     tab->entries = new_tab->entries;
     free(new_tab);
 }
```

Looks like it's freshly rebuilt table. Pretty easy to reproduce the problem on 2.5, I remember it took more tries on 2.4 (didn't valgrind). An extra pair of eyes more experienced with this code than I am would be appreciated. Thanks.

---

#7 - 02/06/2018 03:12 PM - vmakarov (Vladimir Makarov)

On 02/06/2018 05:00 AM, Eric Wong wrote:

```
Eric Wong normalperson@yhbt.net wrote:

v.ondruch@tiscali.cz wrote:
```

On 02/06/2018 05:00 AM, Eric Wong wrote:

```
y.ondruch@tiscali.cz wrote:
```
The thread_safe gem is not maintained anymore, but I don't see any reason why its test suite should segfault with Ruby 2.5. Right, no 3rd-party C exts loaded and I hit this in trunk, too. Using -fsanitize=address reveals use-after-free in st.c. Investigating, but maybe Vladimir can find it sooner. Maybe my initial investigation was correct, after all.

valgrind takes forever, but indicates the free is caused by rebuild_table; so it doesn't look like we missed GC marking during rebuild. Disabling the free(tab->entries) at line st.c:792 (patch below) seems to indicate success with the thread_safe test suite (letting it loop overnight).

Looks like the new_tab != tab case of rebuild is leaving a hanging reference somewhere.

Looks like it's freshly rebuilt table. Pretty easy to reproduce the problem on 2.5, I remember it took more tries on 2.4 (didn't valgrind). An extra pair of eyes more experienced with this code than I am would be appreciated. Thanks.

Eric, thank you for working on the problem and analyzing it. I'll look at this and try to fix it as soon as possible.

Vladimir Makarov
vmakarov@redhat.com
wrote:

On 02/06/2018 05:00 AM, Eric Wong wrote:

during rebuild. Disabling the free(tab->entries) at line st.c:792 (patch below) seems to indicate success with the thread_safe test suite (letting it loop overnight).

It still crashed after four runs :< It might run longer with the simplecov/coveralls stuff commented out in spec_helper.rb since coverage creates a giant hash and might increase the chance of failure.

Eric, thank you for working on the problem and analyzing it. I'll look at this and try to fix it as soon as possible.

Good to know :)

Vladimir Makarov
vmakarov@redhat.com
wrote:

On 02/06/2018 05:00 AM, Eric Wong wrote:

during rebuild. Disabling the free(tab->entries) at line st.c:792 (patch below) seems to indicate success with the thread_safe test suite (letting it loop overnight).

It still crashed after four runs :< It might run longer with the simplecov/coveralls stuff commented out in spec_helper.rb since coverage creates a giant hash and might increase the chance of failure.

Eric, thank you for working on the problem and analyzing it. I'll look at this and try to fix it as soon as possible.

I reproduced this crash although the reproducing is not stable with or without valgrind.
It is a typical data race. The same problem existed in the old hash tables. It also rebuilt tables and freed old data structure.

File st.c was never thread-safe. The data races are/were possible in many places.

We could make st.c thread-safe. But I don't think it is a right way. It is not a trivial task and it also will hurt performance considerably. We still needs thread-unaware level to work with hash tables (st.c) for cases when tables are used internally in one thread.

So I think the crash should be fixed in other places where calls of st.c happen.

I don't know how it should be fixed because I don't know Ruby thread semantics. Does Ruby guarantee that there are no data races or should a ruby programmer still provides thread synchronization despite GIL? If it is later, thread_safe gem is probably buggy because one thread reading a table and another thread inserting elements while process table in a Ruby block. If there is no sync it is a typical data race and the result is unpredictable. In this case it is a segfault crash. We could just give a better message about the data races if segfault happens in st.c.

Also I don't know how GIL works. Where the thread switching can happen. Is the switch possible in find_table_ind or we just read unsync cached value in the thread because st.c never used atomics.

Unfortunately I am not well familiar with Ruby threads so it is hard for me to say how to fix it. I only think that we should keep st.c thread-unaware as it always was.

#10 - 02/07/2018 05:22 PM - normalperson (Eric Wong)

Vladimir Makarov vmakarov@redhat.com wrote:

On 02/06/2018 02:38 PM, Eric Wong wrote:

Vladimir Makarov vmakarov@redhat.com wrote:

On 02/06/2018 05:00 AM, Eric Wong wrote:

During rebuild. Disabling the free(tab->entries) at line st.c:792 (patch below) seems to indicate success with the thread_safe test suite (letting it loop overnight).
It still crashed after four runs :(. It might run longer with the simplecov/coveralls stuff commented out in spec_helper.rb since coverage creates a giant hash and might increase the chance of failure.

Eric, thank you for working on the problem and analyzing it. I'll look at this and try to fix it as soon as possible.

I reproduced this crash although the reproducing is not stable with or without valgrind.

It is a typical data race. The same problem existed in the old hash tables. It also rebuilt tables and freed old data structure.

Ah, thank you for your investigation. I missed this earlier; but I see a problem with the uncommon calls to .eq? and .hash via rb_any_cmp and obj_any_hash functions which may invoke scheduling in the middle of a lookup.

File st.c was never thread-safe. The data races are/were possible in many places.

We could make st.c thread-safe. But I don't think it is a right way. It is not a trivial task and it also will hurt performance considerably. We still needs thread-unaware level to work with hash tables (st.c) for cases when
tables are used internally in one thread.

100% agreed. st.c performance cannot be compromised in the common case for thread-safety.

So I think the crash should be fixed in other places where calls of st.c happen.

I don't know how it should be fixed because I don't know Ruby thread semantics. Does Ruby guarantee that there are no data races or should a ruby programmer still provides thread synchronization despite GIL? If it is later, thread_safe gem is probably buggy because one thread reading a table and another thread inserting elements while process table in a Ruby block. If there is no sync it is a typical data race and the result is unpredictable. In this case it is a segfault crash. We could just give a better message about the data races if segfault happens in st.c.

Yes, thread_safe does not live up to its name, apparently.

Regardless of its bugginess, I don't believe segfaults should be triggerable from pure Ruby code.

(I do not speak for the rest of the team)

Yes, probably doable. I thought about this while eating; but we can probably use rebuilds_num as a seqlock.
unsigned int seq;

retry:
seq = tab->rebuilds_num; /* needs barriers /
...
...->compare(a, b); /* may rebuild */
if (tab->rebuilds_num != seq)
goto retry;

Not free in terms of overhead, but should still be cheap and
shouldn't need atomics on the reader side; only memory barriers
to ensure correct ordering with some CPUs.

When growing/shrinking a Hash table, the GIL should be held, so that should be observed atomically by other threads.
Or is it the problem that #rehash needs to call back to Ruby code?

Right, the rebuild is done entirely with the GVL held, so that's
not a problem. It needs to update rebuild_num atomically
for the above reader pseudocode to work, but that's a cold path.

Vladimir: thoughts?

#13 - 02/08/2018 03:21 PM - vmakarov (Vladimir Makarov)

On 02/07/2018 06:19 PM, Eric Wong wrote:

eregontp@gmail.com wrote:

The GIL guarantees all C code is executed with the GIL held, so C code always perfectly sees effects performed by C code of other threads
(except rb_thread_call_without_gvl but it is not used here, is it?).
Correct, GVL protects common cases (string/fixnum/etc)

In this case: is the code during a Hash lookup, after calling Ruby's #hash/#eql? expecting some state to not have changed since before the
call?
If so, it should be enough to re-check that state after the call.
Yes, probably doable. I thought about this while eating;
but we can probably use rebuilds_num as a seqlock:

unsigned int seq;

retry:
seq = tab->rebuilds_num; /* needs barriers /
...
...->compare(a, b); /* may rebuild */
if (tab->rebuilds_num != seq)
goto retry;

Not free in terms of overhead, but should still be cheap and
shouldn't need atomics on the reader side; only memory barriers
to ensure correct ordering with some CPUs.

When growing/shrinking a Hash table, the GIL should be held, so that should be observed atomically by other threads.
Or is it the problem that #rehash needs to call back to Ruby code?
Right, the rebuild is done entirely with the GVL held, so that's
not a problem. It needs to update rebuild_num atomically
for the above reader pseudocode to work, but that's a cold path.

Vladimir: thoughts?

Yes, I think it could work. It is probably a solution with minimal
possible overhead. Besides compare, there are other external functions
called by st.c (e.g. func in st_insert2 or st_general_FOREACH) but
probably they don't go from C land to Ruby land (where the thread
switch can happen and the table modified or even eql method would be
written to modify the table which would be a very weird code). As I saw
these functions are used to inform GC only. But it worth to add
additional asserts to check it.

Strange that the old hash tables had no such code already. Probably we
were lucky that the bug did not triggered before.
I'll work on it. I think the patch will be ready on the next week. I need to evaluate a performance impact too.

#14 - 02/12/2018 04:00 PM - vmakarov (Vladimir Makarov)
- File thread-table-rebuild.patch added

vmakarov (Vladimir Makarov) wrote:

I'll work on it. I think the patch will be ready on the next week. I need to evaluate a performance impact too.

The following patch solves the problem of rebuilding table in one thread during a search of the same table in another thread. The thread switch can happen in Ruby method implementing equal or hash and the other thread rebuilds the table (by adding elements).

Analogous problem can happen even in one thread program if a developer implements an equal method which can change a table (although it would be a very weird program).

This problem results in ICE for new hash tables. For old hash tables, the problem would have corrupted hash table structure which probably results in later incorrect table behaviour.

The patch takes into account that the table can be rebuilt during hashing or comparison functions and takes a special precaution for this situation by restarting a hash table operation (or its part) again.

Before the patch, thread-safe gem crashed 2-3 times out of each 10 runs. After applying the patch, no crashes happen (I've tried 50 runs).

As for hash table performance, fortunately, I did not found performance degradations. Geometric mean for hash table benchmarks run on x86-64 (i7-4790K) by

ruby ..trunk/benchmark/driver.rb -p hash -r 8 -e trunk:: -e current::
is practically the same (about 0.3% difference).

I have no write access to Ruby repository. So please, consider the patch for inclusion into the trunk.

I cannot find where changelog entry should go for the current trunk, but here it is.

Mon Feb 12 10:41:18 2018 Vladimir Makarov vmakarov@redhat.com

* st.c: Add comment about table rebuilding during comparison.
 (DO_PTR_EQUAL_CHECK): New macro.
(REBUILT_TABLE_ENTRY_IND, REBUILT_TABLE_BIN_IND): New macros.
(peek, find_entry, find_table_entry_ind, find_table_bin_ind): Use new macros. Return the rebuild flag.
 (find_table_bin_ptr_and_reserve): Ditto.
 (st_lookup, st_get_key, st_insert, st_insert2): Retry the operation if the table was rebuilt.
 (st_rehash_linear, st_rehash_indexed): Use DO_PTR_EQUAL_CHECK.
 (st_rehash): Retry the operation if the table was rebuilt.

#15 - 02/13/2018 09:45 AM - normalperson (Eric Wong)
- Backport changed from 2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN to 2.3: REQUIRED, 2.4: REQUIRED, 2.5: REQUIRED

#16 - 02/13/2018 10:22 AM - normalperson (Eric Wong)

vmakarov@redhat.com wrote:

File thread-table-rebuild.patch added

03/15/2022
Thank you for working on this!

This problem results in ICE for new hash tables. For old hash tables, the problem would have corrupted hash table structure which probably results in later incorrect table behaviour.

ICE - Internal Compiler Error? Or did you mean "use-after-free"?

Yes, 2.3 and earlier st.c is affected by this, too, just seems to hit less frequently... 2.3 maintainer(s) will need to backport from scratch, maybe

I have no write access to Ruby repository. So please, consider the patch for inclusion into the trunk.

Looks good to me, committed as r62396.
You can probably ask hsbt and matz for commit access.

I cannot find where changelog entry should go for the current trunk, but here it is.

We write changelog entries in the commit message nowadays. In the future, you can send the output of "git format-patch" since it appears you're using git anyways and we can "git am" it.

I also wrote the following text to summarize in r62396

st.c: retry operations if rebuilt

Calling the .eql? and .hash methods during a Hash operation can result in a thread switch or a signal handler to run: allowing one execution context to rebuild the hash table while another is still reading or writing the table. This results in a use-after-free bug affecting the thread_safe-0.3.6 test suite and likely other bugs.

This bug did not affect users of commonly keys (String, Symbol, Fixnum) as those are optimized to avoid method dispatch for .eql? and .hash methods.

A separate version of this change needs to be ported to Ruby 2.3.x which had a different implementation of st.c but was affected by the same bug.

(and copied your changelog entry below)

#17 - 02/13/2018 09:31 PM - vmakarov (Vladimir Makarov)
On 02/13/2018 05:13 AM, Eric Wong wrote:

vmakarov@redhat.com wrote:

File thread-table-rebuild.patch added
Thank you for working on this!

This problem results in ICE for new hash tables. For old hash tables, the problem would have corrupted hash table structure which probably results in later incorrect table behaviour.
ICE - Internal Compiler Error? Or did you mean "use-after-free"?
Fortunately, not internal compiler error (I wrongly used the term from GCC development :). I meant use-after-free.
Yes, 2.3 and earlier st.c is affected by this, too, just seems to hit less frequently... 2.3 maintainer(s) will need to backport from scratch, maybe

I have no write access to Ruby repository. So please, consider the

I cannot find where changelog entry should go for the current trunk, but here it is.
We write changelog entries in the commit message nowadays. In the future, you can send the output of "git format-patch" since it appears you're using git anyways and we can "git am" it.

I also wrote the following text to summarize in r62396

```
st.c: retry operations if rebuilt
```

Calling the eql? and .hash methods during a Hash operation can result in a thread switch or a signal handler to run: allowing one execution context to rebuild the hash table while another is still reading or writing the table. This results in a use-after-free bug affecting the thread_safe-0.3.6 test suite and likely other bugs.

This bug did not affect users of commonly keys (String, Symbol, Fixnum) as those are optimized to avoid method dispatch for eql? and .hash methods.

A separate version of this change needs to be ported to Ruby 2.3.x which had a different implementation of st.c but was affected by the same bug.

(and copied your changelog entry below)

Thank you, Eric!

#18 - 02/23/2018 01:16 PM - vo.x (Vit Ondruch)
Thank you for the patch. I applied it to Ruby in Fedora and so far like 10 builds of thread_safe passed without issues. I hope I'll be able to update the Ruby package soon.

#19 - 03/05/2018 03:36 PM - vo.x (Vit Ondruch)
- Status changed from Open to Closed

thread_safe build just fine with the patch applied. Closing this so the patch can be backported soon.

#20 - 03/20/2018 09:37 AM - naruse (Yui NARUSE)
- Backport changed from 2.3: REQUIRED, 2.4: REQUIRED, 2.5: REQUIRED to 2.3: REQUIRED, 2.4: REQUIRED, 2.5: DONE

ruby_2_5 r62858 merged revision(s) 62396.

#21 - 06/30/2018 12:56 PM - usa (Usaku NAKAMURA)
- Backport changed from 2.3: REQUIRED, 2.4: REQUIRED, 2.5: DONE to 2.3: REQUIRED, 2.4: DONE, 2.5: DONE

ruby_2_4 r63805 merged revision(s) 62396.

**Files**

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread-table-rebuild.patch</td>
<td>18.6 KB</td>
<td>02/12/2018</td>
<td>vmakarov (Vladimir Makarov)</td>
</tr>
</tbody>
</table>