Ruby master - Bug #18501

[BUG] try to mark T_NONE object in RubyVM::InstructionSequence. load_from_binary

01/20/2022 03:02 PM - by root (Jean Boussier)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td>ruby -v:</td>
</tr>
<tr>
<td></td>
<td>ruby 3.1.0p0 (2021-12-25 revision fb4df44f16) [x86_64-linux]</td>
</tr>
</tbody>
</table>

Description

<OBJ_INFO:gc_mark_ptr@gc.c:6709> 0x00007fbf1fba1270 [2 M ] T_NONE
/tmp/bundle/ruby/3.1.0/gems/bootsnap-1.10.1/lib/bootsnap/compile_cache/iseq.rb:49: [BUG] try to mark T_NONE object
ruby 3.1.0p0 (2021-12-25 revision fb4df44f16) [x86_64-linux]

```
-- Control frame information -----------------------------------------------
c:0024 p:----- s:0126 e:000125 CFUNC :load_from_binary
c:0022 p:----- s:0114 e:000113 CFUNC :fetch
   c:0021 p:0061 s:0106 e:000105 METHOD /tmp/bundle/ruby/3.1.0/gems/bootsnap-1.10.1/lib/bootsnap/compile_cache/iseq.rb:60
   c:0020 p:0053 s:0099 e:000098 METHOD /tmp/bundle/ruby/3.1.0/gems/bootsnap-1.10.1/lib/bootsnap/compile_cache/iseq.rb:85 [FINISH]
c:0019 p:----- s:0093 e:000092 CFUNC :require
   c:0016 p:0005 s:0069 e:000068 BLOCK /app/test/load_selected_tests.rb:11 [FINISH]
c:0015 p:----- s:0065 e:000064 CFUNC :each
   c:0014 p:0006 s:0061 e:000060 METHOD /app/test/load_selected_tests.rb:10
   c:0013 p:0025 s:0057 e:000056 TOP /app/test/load_selected_tests.rb:28 [FINISH]
c:0012 p:----- s:0054 e:000053 CFUNC :require
   c:0011 p:0016 s:0049 e:000048 BLOCK /tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:287 [FINISH]
c:0010 p:----- s:0045 e:000044 CFUNC :each
   c:0009 p:0008 s:0041 e:000040 METHOD /tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:286
   c:0008 p:0021 s:0037 E:001fd0 METHOD /tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:78 [FINISH]
c:0007 p:----- s:0031 e:000030 CFUNC :public_send
   c:0006 p:0073 s:0026 e:000025 METHOD /tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:35
   c:0005 p:0007 s:0021 e:000020 METHOD /tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:19
   c:0004 p:0034 s:0016 E:000015 TOP /tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/exe/minitest-queue:5:in `main'
c:0003 p:----- s:0013 e:000012 CFUNC :load
   c:0002 p:0124 s:0008 E:0023a0 EVAL /tmp/bundle/ruby/3.1.0/bin/minitest-queue:25 [FINISH]
c:0001 p:0000 s:0003 E:002230 (none) [FINISH]
```

-- Ruby level backtrace information ----------------------------------------
/tmp/bundle/ruby/3.1.0/bin/minitest-queue:25:in `<main>'
/tmp/bundle/ruby/3.1.0/bin/minitest-queue:25:in `load'
/tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/exe/minitest-queue:5:in `<top (required)>'
/tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:19:in `invoke'
/tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:35:in `run'
/tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:35:in `public_send'
/tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:78:in `run_command'
/tmp/bundle/ruby/3.1.0/gems/ci-queue-0.22.0/lib/minitest/queue/runner.rb:286:in `load_tests'

03/20/2022 1/6
From my limited understanding it seems to happen if GC triggers at a very specific point.

Associated revisions

Revision 2a76440f - 02/04/2022 10:36 PM - tenderlovemaking (Aaron Patterson)

[Bug #18501] Fire write barrier after hash has been written

Before this change the write barrier was executed before the key and value were actually reachable via the Hash. This could cause inconsistencies in object coloration which would lead to accidental collection of dup'd keys.

Example:

1. Object O is grey, Object P is white.
2. Write barrier fires O -> P
3. Write barrier does nothing
4. Malloc happens, which starts GC
5. GC colors O black
6. P is written in to O (now we have O -> P reference)
7. P is now accidentally treated as garbage

Revision 86c8e151 - 02/07/2022 10:58 AM - naruse (Yui NARUSE)

merge revision(s) 2a76440fac62: [Backport #18501]

[Bug #18501] Fire write barrier after hash has been written

Before this change the write barrier was executed before the key and value were actually reachable via the Hash. This could cause inconsistencies in object coloration which would lead to accidental collection of dup'd keys.

Example:

1. Object O is grey, Object P is white.
2. Write barrier fires O -> P
3. Write barrier does nothing
4. Malloc happens, which starts GC
5. GC colors O black
6. P is written in to O (now we have O -> P reference)
7. P is now accidentally treated as garbage

---
Hash is writing T_NONE references

I think it's possible that T_NONE objects are being written in to a hash. Basically we dup the string key, and if the hash needs to expand, it does so before actually inserting the key. The expansion of the hash causes a malloc which kicks the GC, and the compiler has optimized the code such that the GC doesn't see a reference to the object in the stack or a register.

The GC ends up collecting the string, then the Hash writes a T_NONE to the underlying table.

Let's follow the order of events for code like this:

```
hash["foo"] = "bar"
```

1. **Ruby -> rb_hash_aset** is called
   
   This function checks if the key is a string, and if so it does something special. Namely, it calls RHASH_UPDATE_ITER with a special callback `hash_aset_str`. RHASH_UPDATE_ITER is just a wrapper for tbl_update.

2. **rb_hash_aset -> tbl_update**.
   
   This function just calls in to rb_hash_stlike_update along with a struct. The `func` member of the struct is our callback `hash_aset_str`.

3. **tbl_update -> rb_hash_stlike_update**
   
   This function just checks if we have an AR table or not. In this case, we have an AR table, so we just call ar_update. `arg` is passed to ar_update and it has the `hash_aset_str` function pointer. We also pass `tbl_update_modify` as a callback.

4. **rb_hash_stlike_update -> ar_update**
   
   `ar_update` does some checks, then eventually calls `tbl_update_modify` as a callback, passing `arg`.

5. **ar_update -> tbl_update_modify**
   
   `tbl_update_modify` finally calls `hash_aset_str` as a callback.

6. **tbl_update_modify -> hash_aset_str**
   
   `hash_aset_str` checks the key. The key doesn't exist (it's new), but it's also not frozen. So we allocate a new frozen string object and assign that to `*key` so that the caller can read it.

   Now we return up the stack. `hash_aset_str` returns control to `tbl_update_modify`.

7. **tbl_update_modify**
   
   `tbl_update_modify` executes a write barrier on the key and value.

   This is odd because the key and value haven't been written yet. Neither are reachable via the hash yet. Since we have allocated a new string, the string's liveness depends on being found in the C stack or a register. It's not referenced from the hash yet.

   I think it's OK if the write barrier happens here, but it is odd that we execute the barrier but the reference isn't "real" yet.

   `tbl_update_modify` returns control to ar_update.

8. **ar_update**
ar_update finally adds the reference to the hash.

I think this is where the bug is. Adding a reference to the hash could cause the hash to expand which would cause xmalloc to execute, and this could cause GC to run. The compiler could have optimized this code in such a way that the reference to the key is not on the stack and no longer in a register.

#4 - 01/26/2022 07:52 AM - byroot (Jean Boussier)
Thanks @tenderlove, your explanation makes sense to me.

From your description we should be able to reproduce with GC.stress = true no?

#5 - 02/04/2022 09:07 AM - byroot (Jean Boussier)
- Backport changed from 2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN to 2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: REQUIRED

tenderlovemaking (Aaron Patterson) implemented a patch for this: https://github.com/ruby/ruby/pull/5525, we deployed it on our infra and so far the bug is gone. I'd like to let it run for a few more days to be fully certain though.

tenderlovemaking (Aaron Patterson), do you think this bug might have been present on any older version?

#6 - 02/04/2022 10:36 PM - tenderlovemaking (Aaron Patterson)
- Status changed from Open to Closed

Applied in changeset git|2a76440fac62bb0f6e53ccada07ca4f4b47b78cf9.

[Bug #18501] Fire write barrier after hash has been written

Before this change the write barrier was executed before the key and value were actually reachable via the Hash. This could cause inconsistencies in object coloration which would lead to accidental collection of dup'd keys.

Example:
1. Object O is grey, Object P is white.
2. Write barrier fires O -> P
3. Write barrier does nothing
4. Malloc happens, which starts GC
5. GC colors O black
6. P is written in to O (now we have O -> P reference)
7. P is now accidentally treated as garbage

#7 - 02/04/2022 10:41 PM - byroot (Jean Boussier)
- Backport changed from 2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: REQUIRED to 2.6: DONTNEED, 2.7: DONTNEED, 3.0: DONTNEED, 3.1: REQUIRED

I'd like to let it run for a few more days to be fully certain though.

We ran it quite enough and saw no more crashes, so we're quite confident the patches work.

I'm marking 3.0 and older as DONTNEED because we never noticed that crash when we were running those versions.

#8 - 02/07/2022 10:58 AM - naruse (Yui NARUSE)
- Backport changed from 2.6: DONTNEED, 2.7: DONTNEED, 3.0: DONTNEED, 3.1: REQUIRED to 2.6: DONTNEED, 2.7: DONTNEED, 3.0: DONTNEED, 3.1: DONE

ruby_3_1 86c8e15170484fe23b311e567717053f147f1d9c merged revision(s) 2a76440fac62b.

Files
0001-Guard-hash-k-v.patch 881 Bytes 01/25/2022 tenderlovemaking (Aaron Patterson)