Ruby - Bug #18796

GC compaction gets stuck on Mac OS when a debugger is attached

05/23/2022 12:05 AM - kjtsanaktsidis (KJ Tsanaktsidis)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 3.2.0dev (2022-05-18T05:33:00Z ktsanaktsidis/skip.. 97c12c5f69) [arm64-darwin21]	**Backport:**	2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN

Description

The GC compaction mechanism uses a kind of read barrier to detect attempts to read objects from pages that have already been moved from (or, at least, I *think* that's what it does). This is implemented by marking the OS pages as unreadable using mprotect, and installing a SIGBUS/SIGSEGV handler to detect attempts to access them. When they are accessed, the page move is invalidated and the page is marked as accessible again, allowing the faulted access to proceed.

This was added in 67b2c21c327c96d80b8a0fe02a96d417e85293e8 (issue https://bugs.ruby-lang.org/issues/17176).

Unfortunately, when a debugger is attached to the Ruby interpreter on Mac OS, the debugger will trap the EXC_BAD_ACCESS mach exception before the runtime can transform that into a SIGBUS signal and dispatch it. Thus, execution gets stuck; any attempt to continue from the debugger re-executes the line that caused the exception without invoking the SIGBUS signal handler and no forward progress can be made.

This makes it impossible to debug either the ruby interpreter or a C extension whilst compaction is in use.

I don't *really* understand mach stuff in MacOS that well, but I have attached a patch which seems to fix the issue. It disables the EXC_BAD_ACCESS exception handler when installing the SIGBUS/SIGSEGV handlers, and re-enables it when compaction is done. The debugger will still trap on the attempt to read the protected page, but it will be trapping the SIGBUS signal, rather than the EXC_BAD_ACCESS mach exception. It's possible to continue from this in the debgugger, which invokes the SIGBUS signal handler and thus allows forward progress to be made.

For example, this is what happens before applying this patch:

```
Process 96555 stopped
* thread #9, name = 'repro_rack.rb:43', stop reason = EXC_BAD_ACCESS (code=2, address=0x107dbbfc0)
    frame #0: 0x00000001000af834 ruby`vm_ccs_free at imemo.h:170:49
   167          const VALUE mask = (IMEMO_MASK << FL_USHIFT) | RUBY_T_MASK;
   168          const VALUE expected_type = (imemo_type << FL_USHIFT) | T_IMEMO;
   169          /* fixed at runtime. */
-> 170          return expected_type == (RBASIC(imemo)->flags & mask);
   171      }
   172      else {
   173          return 0;
Target 0: (ruby) stopped.
(lldb) bt
* thread #9, name = 'repro_rack.rb:43', stop reason = EXC_BAD_ACCESS (code=2, address=0x107dbbfc0)
  * frame #0: 0x00000001000af834 ruby`vm_ccs_free at imemo.h:170:49
    frame #1: 0x00000001000af820 ruby`vm_ccs_free(ccs=0x00006000003222c0, alive=<unavailable>, obj
space=0x0000000101008c00, klass=4383646800) at gc.c:3102:21
    frame #2: 0x00000001000c3a5c ruby`cc_table_free_i(ccs_ptr=<unavailable>, data_ptr=<unavailable
>) at gc.c:3181:5
    frame #3: 0x00000001001f3334 ruby`rb_id_table_foreach_values(tbl=0x00006000002f6fc0, func=(rub
y`cc_table_free_i at gc.c:3177), data=0x0000000170bd9e20) at id_table.c:296:45
    frame #4: 0x00000001000c0354 ruby`obj_free at gc.c:3196:9
    frame #5: 0x00000001000c0330 ruby`obj_free(objspace=0x0000000101008c00, obj=4383646800) at gc.
c:3312:9
    frame #6: 0x00000001000c00b8 ruby`gc_sweep_page at gc.c:5371:25
    frame #7: 0x00000001000c002c ruby`gc_sweep_page(objspace=0x0000000101008c00, heap=0x0000000101
008df0, ctx=0x0000000170bd9f60) at gc.c:5455:13
    frame #8: 0x00000001000bfcd0 ruby`gc_compact_move(objspace=0x0000000101008c00, heap=0x00000001
01008df0, src=4423151000) at gc.c:8197:9
```

```
    frame #9: 0x00000001000bdf40 ruby`gc_sweep at gc.c:8223:18
    frame #10: 0x00000001000bdeec ruby`gc_sweep at gc.c:8260:18
    frame #11: 0x00000001000bde14 ruby`gc_sweep at gc.c:8303:18
    frame #12: 0x00000001000bdaf0 ruby`gc_sweep(objspace=<unavailable>) at gc.c:5949:9
    frame #13: 0x00000001000c3524 ruby`gc_start [inlined] gc_marks(objspace=0x0000000101008c00, fu
ll_mark=<unavailable>) at gc.c:8380:9
    frame #14: 0x00000001000c2c84 ruby`gc_start(objspace=0x0000000101008c00, reason=<unavailable>)
 at gc.c:9205:2
    frame #15: 0x00000001000b9c14 ruby`gc_compact at gc.c:9086:15
    frame #16: 0x00000001000b9be4 ruby`gc_compact [inlined] gc_start_internal(ec=<unavailable>, se
lf=<unavailable>, full_mark=20, immediate_mark=20, immediate_sweep=20, compact=20) at gc.c:9519:5
    frame #17: 0x00000001000b9bd0 ruby`gc_compact(ec=0x00000001069b9480, self=4354993720) at gc.c:
10491:5
    frame #18: 0x0000000100233674 ruby`vm_exec_core [inlined] builtin_invoker0(ec=0x00000001069b94
80, self=<unavailable>, argv=0x0000000000000000, funcptr=<unavailable>) at vm_insnhelper.c:5826:12
    frame #19: 0x000000010023366c ruby`vm_exec_core [inlined] invoke_bf(ec=0x00000001069b9480, reg
_cfp=<unavailable>, argv=0x0000000000000000) at vm_insnhelper.c:5966:17
    frame #20: 0x0000000100233668 ruby`vm_exec_core [inlined] vm_invoke_builtin_delegate(ec=0x0000
0001069b9480, bf=<unavailable>, start_index=<unavailable>) at vm_insnhelper.c:5989:16
    frame #21: 0x0000000100233624 ruby`vm_exec_core(ec=<unavailable>, initial=<unavailable>) at in
sns.def:1510:11
    frame #22: 0x0000000100243384 ruby`rb_vm_exec(ec=0x00000001069b9480, mjit_enable_p=<unavailabl
e>) at vm.c:0:21
    frame #23: 0x0000000100252854 ruby`invoke_block_from_c_bh [inlined] invoke_block(ec=0x00000001
069b9480, iseq=0x0000000104f3b1a0, self=4302163400, captured=0x0000000170cdacd8, cref=0x0000000000
000000, type=<unavailable>, opt_pc=<unavailable>) at vm.c:1319:12
    frame #24: 0x00000001002527b4 ruby`invoke_block_from_c_bh [inlined] invoke_iseq_block_from_c(e
c=0x00000001069b9480, captured=0x0000000170cdacd8, self=4302163400, argc=<unavailable>, argv=<unav
ailable>, kw_splat=<unavailable>, passed_block_handler=<unavailable>, cref=0x0000000000000000, is_
lambda=<unavailable>, me=0x0000000000000000) at vm.c:1375:9
    frame #25: 0x0000000100252684 ruby`invoke_block_from_c_bh(ec=0x00000001069b9480, block_handler
=<unavailable>, argc=<unavailable>, argv=<unavailable>, kw_splat=<unavailable>, passed_block_handl
er=<unavailable>, cref=<unavailable>, is_lambda=<unavailable>, force_blockarg=0) at vm.c:1393:13
    frame #26: 0x0000000100251ff0 ruby`loop_i [inlined] vm_yield_with_cref(ec=<unavailable>, argc=
0, argv=0x0000000000000000, kw_splat=0, cref=0x0000000000000000, is_lambda=0) at vm.c:1430:12
    frame #27: 0x0000000100251fa8 ruby`loop_i [inlined] vm_yield(ec=<unavailable>, argc=0, argv=0x
0000000000000000, kw_splat=0) at vm.c:1438:12
    frame #28: 0x0000000100251fa8 ruby`loop_i [inlined] rb_yield_0(argc=0, argv=0x0000000000000000
) at vm_eval.c:1347:12
    frame #29: 0x0000000100251f9c ruby`loop_i(_=<unavailable>) at vm_eval.c:1446:2
    frame #30: 0x000000010009f554 ruby`rb_vrescue2(b_proc=(ruby`loop_i at vm_eval.c:1444), data1=0
, r_proc=(ruby`loop_stop at vm_eval.c:1453), data2=0, args=<unavailable>) at eval.c:904:11
    frame #31: 0x000000010009f3cc ruby`rb_rescue2(b_proc=<unavailable>, data1=<unavailable>, r_pro
c=<unavailable>, data2=<unavailable>) at eval.c:885:17
    frame #32: 0x0000000100023eb94 ruby`rb_f_loop(self=4302163400) at vm_eval.c:1497:12
    frame #33: 0x0000000100024bf6c ruby`vm_call_cfunc_with_frame(ec=0x00000001069b9480, reg_cfp=0x0
000000170cdacc0, calling=<unavailable>) at vm_insnhelper.c:3037:11
    frame #34: 0x0000000100024e560 ruby`vm_sendish(ec=0x00000001069b9480, reg_cfp=0x0000000170cdacc
0, cd=0x000060000000ac40, block_handler=<unavailable>, method_explorer=<unavailable>) at vm_insnhe
lper.c:0
    frame #35: 0x0000000100022f658 ruby`vm_exec_core(ec=<unavailable>, initial=<unavailable>) at in
sns.def:759:11
    frame #36: 0x0000000100243384 ruby`rb_vm_exec(ec=0x00000001069b9480, mjit_enable_p=<unavailabl
e>) at vm.c:0:21
    frame #37: 0x000000010024105c ruby`rb_vm_invoke_proc(ec=0x00000001069b9480, proc=<unavailable>
, argc=<unavailable>, argv=<unavailable>, kw_splat=<unavailable>, passed_block_handler=<unavailabl
e>) at vm.c:0
    frame #38: 0x00000001001ff358 ruby`thread_do_start_proc(th=0x00000001069b92d0) at thread.c:712
:16
    frame #39: 0x00000001001feb60 ruby`thread_start_func_2 [inlined] thread_do_start(th=0x00000001
069b92d0) at thread.c:731:18
    frame #40: 0x00000001001fea38 ruby`thread_start_func_2(th=0x00000001069b92d0, stack_start=<una
vailable>) at thread.c:806:9
    frame #41: 0x00000001001fe558 ruby`thread_start_func_1(th_ptr=<unavailable>) at thread_pthread
.c:1077:9
    frame #42: 0x000000019160626c libsystem_pthread.dylib`_pthread_start + 148
(lldb) cont
```

```
Process 96555 resuming
Process 96555 stopped
* thread #9, name = 'repro_rack.rb:43', stop reason = EXC_BAD_ACCESS (code=2, address=0x107dbbfc0)
    frame #0: 0x00000001000af834 ruby`vm_ccs_free at imemo.h:170:49
   167          const VALUE mask = (IMEMO_MASK << FL_USHIFT) | RUBY_T_MASK;
   168          const VALUE expected_type = (imemo_type << FL_USHIFT) | T_IMEMO;
   169          /* fixed at runtime. */
-> 170          return expected_type == (RBASIC(imemo)->flags & mask);
   171      }
   172      else {
   173          return 0;
Target 0: (ruby) stopped.
(lldb) cont
Process 96555 resuming
Process 96555 stopped
* thread #9, name = 'repro_rack.rb:43', stop reason = EXC_BAD_ACCESS (code=2, address=0x107dbbfc0)
    frame #0: 0x00000001000af834 ruby`vm_ccs_free at imemo.h:170:49
   167          const VALUE mask = (IMEMO_MASK << FL_USHIFT) | RUBY_T_MASK;
   168          const VALUE expected_type = (imemo_type << FL_USHIFT) | T_IMEMO;
   169          /* fixed at runtime. */
-> 170          return expected_type == (RBASIC(imemo)->flags & mask);
   171      }
   172      else {
   173          return 0;
Target 0: (ruby) stopped.
(lldb) cont
Process 96555 resuming
Process 96555 stopped
* thread #9, name = 'repro_rack.rb:43', stop reason = EXC_BAD_ACCESS (code=2, address=0x107dbbfc0)
    frame #0: 0x00000001000af834 ruby`vm_ccs_free at imemo.h:170:49
   167          const VALUE mask = (IMEMO_MASK << FL_USHIFT) | RUBY_T_MASK;
   168          const VALUE expected_type = (imemo_type << FL_USHIFT) | T_IMEMO;
   169          /* fixed at runtime. */
-> 170          return expected_type == (RBASIC(imemo)->flags & mask);
   171      }
   172      else {
   173          return 0;
Target 0: (ruby) stopped.
```

Note that when continuing, it keeps trapping the same access to 0x107dbbfc0.

After applying this patch, instead of breaking for EXC_BAD_ACCESS, it breaks for SIGBUS:

```
Process 18736 stopped
* thread #9, name = 'repro_rack.rb:43', stop reason = signal SIGBUS
    frame #0: 0x00000001000af6c4 ruby`vm_ccs_free at imemo.h:170:49
   167          const VALUE mask = (IMEMO_MASK << FL_USHIFT) | RUBY_T_MASK;
   168          const VALUE expected_type = (imemo_type << FL_USHIFT) | T_IMEMO;
   169          /* fixed at runtime. */
-> 170          return expected_type == (RBASIC(imemo)->flags & mask);
   171      }
   172      else {
   173          return 0;
Target 0: (ruby) stopped.
(lldb) cont
Process 18736 resuming
Process 18736 stopped
* thread #9, name = 'repro_rack.rb:43', stop reason = signal SIGBUS
    frame #0: 0x00000001000af6c4 ruby`vm_ccs_free at imemo.h:170:49
   167          const VALUE mask = (IMEMO_MASK << FL_USHIFT) | RUBY_T_MASK;
   168          const VALUE expected_type = (imemo_type << FL_USHIFT) | T_IMEMO;
   169          /* fixed at runtime. */
-> 170          return expected_type == (RBASIC(imemo)->flags & mask);
   171      }
   172      else {
   173          return 0;
Target 0: (ruby) stopped.
```

```
(lldb) cont
Process 18736 resuming
Process 18736 stopped
* thread #9, name = 'repro_rack.rb:43', stop reason = signal SIGBUS
    frame #0: 0x00000001000af6c4 ruby`vm_ccs_free at imemo.h:170:49
   167          const VALUE mask = (IMEMO_MASK << FL_USHIFT) | RUBY_T_MASK;
   168          const VALUE expected_type = (imemo_type << FL_USHIFT) | T_IMEMO;
   169          /* fixed at runtime. */
-> 170          return expected_type == (RBASIC(imemo)->flags & mask);
   171      }
   172      else {
   173          return 0;
Target 0: (ruby) stopped.
(lldb) cont
Process 18736 resuming
Process 18736 stopped
* thread #9, name = 'repro_rack.rb:43', stop reason = signal SIGBUS
    frame #0: 0x00000001000af6c4 ruby`vm_ccs_free at imemo.h:170:49
   167          const VALUE mask = (IMEMO_MASK << FL_USHIFT) | RUBY_T_MASK;
   168          const VALUE expected_type = (imemo_type << FL_USHIFT) | T_IMEMO;
   169          /* fixed at runtime. */
-> 170          return expected_type == (RBASIC(imemo)->flags & mask);
   171      }
   172      else {
   173          return 0;
Target 0: (ruby) stopped.
(lldb) cont
Process 18736 resuming
```

And continuing past the trapped signals now allows execution to continue. In fact, it's convenient to skip through all SIGBUS signals automatically, by running something like process handle -p true -s false SIGBUS in lldb (assuming, of course, that the bug you're trying to debug isn't itself causing a SIGBUS!).

Anyway, I'd like to solve this problem somehow because I'm developing a C extension and trying to ensure it's handling GC moves properly; my extension does have some kind of crash caused by moves, but I couldn't debug it without this patch because I would get stuck in this expected trap inside the compaction read barrier instead. Thanks!

History

#1 - 05/23/2022 12:09 AM - kjtsanaktsidis (KJ Tsanaktsidis)

- Description updated

#2 - 06/08/2022 08:25 AM - kjtsanaktsidis (KJ Tsanaktsidis)

I opened a PR with this patch: https://github.com/ruby/ruby/pull/5991

#3 - 11/10/2022 09:27 AM - kjtsanaktsidis (KJ Tsanaktsidis)

The fix for this was merged.

#4 - 11/10/2022 09:39 AM - ioquatix (Samuel Williams)

- Status changed from Open to Closed

I can confirm it was merged.

Files

0001-Disable-Mach-exception-handlers-when-read-barriers-i.patch	4.93 KB	05/23/2022	kjtsanaktsidis (KJ Tsanaktsidis)