Ruby master - Bug #14867

Process.wait can wait for MJIT compiler process

06/23/2018 07:39 AM - k0kubun (Takashi Kokubun)

Status:	Closed		
Priority:	Normal		
Assignee:	normalperson (Eric Wong)		
Target version:			
ruby -v:		**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

If Ruby tries to wait for any child process, MJIT's gcc/clang process could be caught by the method call. It's not convenient for both Ruby's user and MJIT worker thread, so Process.wait and its families should somehow avoid waiting for it.

Related issues:

Related to Ruby master - Feature #14830: RubyVM::MJIT.pause / RubyVM::MJIT.re...	**Closed**
Related to Ruby master - Bug #15320: IO.popen with MJIT worker thread may dea...	**Closed**

Associated revisions

Revision 359dd59d - 06/23/2018 08:29 AM - k0kubun (Takashi Kokubun)

spec: skip Process wait specs on MJIT

until [Bug #14867] is fixed. I want to start running CI with MJIT
enabled before fixing the problem.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63731 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63731 - 06/23/2018 08:29 AM - k0kubun (Takashi Kokubun)

spec: skip Process wait specs on MJIT

until [Bug #14867] is fixed. I want to start running CI with MJIT
enabled before fixing the problem.

Revision 63731 - 06/23/2018 08:29 AM - k0kubun (Takashi Kokubun)

spec: skip Process wait specs on MJIT

until [Bug #14867] is fixed. I want to start running CI with MJIT
enabled before fixing the problem.

Revision ce2a3b40 - 06/30/2018 12:51 AM - normal

use SIGCHLD_LOSSY to enable waitpid polling mode

Some systems lack SIGCHLD or have incomplete SIGCHLD
implementations. So enable polling mode for them.

[ruby-core:87705] [Bug #14867]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63795 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63795 - 06/30/2018 12:51 AM - normalperson (Eric Wong)

use SIGCHLD_LOSSY to enable waitpid polling mode

Some systems lack SIGCHLD or have incomplete SIGCHLD
implementations. So enable polling mode for them.

[ruby-core:87705] [Bug #14867]

Revision 63795 - 06/30/2018 12:51 AM - normal

use SIGCHLD_LOSSY to enable waitpid polling mode

Some systems lack SIGCHLD or have incomplete SIGCHLD

implementations. So enable polling mode for them.

[ruby-core:87705] [Bug #14867]

Revision e4600b87 - 06/30/2018 03:50 AM - normal

mjit: provide more diagnostics for waitpid failures

Also, enable check for defined(_WIN32) macro for SIGCHLD_LOSSY, too.

[Bug #14867]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63796 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63796 - 06/30/2018 03:50 AM - normalperson (Eric Wong)

mjit: provide more diagnostics for waitpid failures

Also, enable check for defined(_WIN32) macro for SIGCHLD_LOSSY, too.

[Bug #14867]

Revision 63796 - 06/30/2018 03:50 AM - normal

mjit: provide more diagnostics for waitpid failures

Also, enable check for defined(_WIN32) macro for SIGCHLD_LOSSY, too.

[Bug #14867]

Revision 63803 - 06/30/2018 11:56 AM - nobu (Nobuyoshi Nakada)

use sigsetjmp on macOS

SIGCHLD is used internally since r63758, the signal masks need to
be restored.

Revision 79f01d39 - 07/06/2018 06:50 AM - normal

mjit.c: fix waitpid macro return value for win32

We started checking return value of waitpid, so it needs
to be correct for win32 platforms for MJIT to work.

Thanks-to: MSP-Greg (Greg L) Greg.mpls@gmail.com

[ruby-core:87832] [Bug #14867]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63869 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63869 - 07/06/2018 06:50 AM - normalperson (Eric Wong)

mjit.c: fix waitpid macro return value for win32

We started checking return value of waitpid, so it needs
to be correct for win32 platforms for MJIT to work.

Thanks-to: MSP-Greg (Greg L) Greg.mpls@gmail.com

[ruby-core:87832] [Bug #14867]

Revision 63869 - 07/06/2018 06:50 AM - normal

mjit.c: fix waitpid macro return value for win32

We started checking return value of waitpid, so it needs
to be correct for win32 platforms for MJIT to work.

Thanks-to: MSP-Greg (Greg L) Greg.mpls@gmail.com

[ruby-core:87832] [Bug #14867]

Revision ac41c2c1 - 07/07/2018 11:59 PM - normal

process.c (rb_f_exec): pause MJIT before replacing process

Non-parallel "make test-spec" caused
spec/ruby/core/process/wait2_spec.rb failures because mspec
uses "exec" in single-process mode, so there's no chance
the post-exec state could know about the MJIT child process
from its pre-exec state.

[ruby-core:87846] [Bug #14867]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63877 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63877 - 07/07/2018 11:59 PM - normalperson (Eric Wong)

process.c (rb_f_exec): pause MJIT before replacing process

Non-parallel "make test-spec" caused
spec/ruby/core/process/wait2_spec.rb failures because mspec
uses "exec" in single-process mode, so there's no chance
the post-exec state could know about the MJIT child process
from its pre-exec state.

[ruby-core:87846] [Bug #14867]

Revision 63877 - 07/07/2018 11:59 PM - normal

process.c (rb_f_exec): pause MJIT before replacing process

Non-parallel "make test-spec" caused
spec/ruby/core/process/wait2_spec.rb failures because mspec
uses "exec" in single-process mode, so there's no chance
the post-exec state could know about the MJIT child process
from its pre-exec state.

[ruby-core:87846] [Bug #14867]

Revision bc4ecaa6 - 07/08/2018 01:46 AM - normal

test/ruby/test_signal.rb: skip ensure if test is skipped

Thanks to Greg for the fix. [ruby-core:87860] [Bug #14867]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63880 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63880 - 07/08/2018 01:46 AM - normalperson (Eric Wong)

test/ruby/test_signal.rb: skip ensure if test is skipped

Thanks to Greg for the fix. [ruby-core:87860] [Bug #14867]

Revision 63880 - 07/08/2018 01:46 AM - normal

test/ruby/test_signal.rb: skip ensure if test is skipped

Thanks to Greg for the fix. [ruby-core:87860] [Bug #14867]

Revision 953b7a20 - 08/08/2018 07:26 AM - normal

mjit.c: remove old comment about WNOHANG and SIGCHLD [ci skip]

[Bug #14867] implemented exactly what was needed (for POSIX
platforms, at least).

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64225 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64225 - 08/08/2018 07:26 AM - normalperson (Eric Wong)

mjit.c: remove old comment about WNOHANG and SIGCHLD [ci skip]

[Bug #14867] implemented exactly what was needed (for POSIX
platforms, at least).

Revision 64225 - 08/08/2018 07:26 AM - normal

mjit.c: remove old comment about WNOHANG and SIGCHLD [ci skip]

[Bug #14867] implemented exactly what was needed (for POSIX

platforms, at least).

Revision 8ae44386 - 08/23/2018 07:13 PM - normal

NEWS: add entries for rb_waitpid and timer-thread [ci skip]

Some of these changes may affect debugging and tracing tools

[Bug #14867] [ruby-core:88199] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64522 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64522 - 08/23/2018 07:13 PM - normalperson (Eric Wong)

NEWS: add entries for rb_waitpid and timer-thread [ci skip]

Some of these changes may affect debugging and tracing tools

[Bug #14867] [ruby-core:88199] [Misc #14937]

Revision 64522 - 08/23/2018 07:13 PM - normal

NEWS: add entries for rb_waitpid and timer-thread [ci skip]

Some of these changes may affect debugging and tracing tools

[Bug #14867] [ruby-core:88199] [Misc #14937]

History

#1 - 06/23/2018 07:40 AM - k0kubun (Takashi Kokubun)

- Related to Feature #14830: RubyVM::MJIT.pause / RubyVM::MJIT.resume added

#2 - 06/23/2018 08:14 AM - akr (Akira Tanaka)

I think it is bit difficult to solve.

It seems Solaris have good feature: POSIX_SPAWN_WAITPID_NP for posix_spawn.
https://www.unix.com/man-page/all/3C/posix_spawn/
But it is not portable enough for us.

One idea that MJIT invokes compiler via a invoker process:

- ruby forks a temporary process
- the temporary process forks invoker process
- the temporary process exits
- ruby requests the invoker process to invoke compiler (via pipe)

Since the invoker process and compiler processes are not a child process of ruby,
Process.wait doesn't wait them.

or

Deprecate Process.wait.

#3 - 06/23/2018 08:29 AM - k0kubun (Takashi Kokubun)

- Status changed from Open to Closed

Applied in changeset trunk|r63731.

---

spec: skip Process wait specs on MJIT

until [Bug #14867] is fixed. I want to start running CI with MJIT
enabled before fixing the problem.

#4 - 06/23/2018 08:43 AM - k0kubun (Takashi Kokubun)

- Status changed from Closed to Open

#5 - 06/23/2018 09:03 AM - k0kubun (Takashi Kokubun)

> One idea that MJIT invokes compiler via a invoker process

I prefer the idea in general, rather than deprecating Process.wait. Not only the failing tests, but also LeakChecker is reacting to MJIT compiler processes and it can also solve the problem.

An orphan process that continuously spawns compiler processes could be seen as a harmful process for people unfamiliar with MJIT, but we would be able to rename the process to indicate it's Ruby's JIT compiler.

I'm not sure if the invoker process can be properly killed when the main Ruby process exits abnormally and fails to call a finalizer that usually kills the invoker process on Ruby's exit.

#6 - 06/23/2018 09:12 AM - normalperson (Eric Wong)

akr@fsij.org wrote:

> Deprecate Process.wait.

Please don't.

We can implement Process.wait with a global SIGCHLD handler
and mask out PIDs used by MJIT. I already did that from
Feature #13618 and we can extract the work out.
I can work on it.

I'm already extracting timeout handling out separately in
[Feature #14859]

#7 - 06/23/2018 09:23 AM - k0kubun (Takashi Kokubun)

> We can implement Process.wait with a global SIGCHLD handler and mask out PIDs used by MJIT

If it's portable enough compared to Solaris's POSIX_SPAWN_WAITPID_NP, I'm totally fine with the idea to patch Process.{wait,wait2,waitall} since I have a concern on killing the invoker process.

#8 - 06/23/2018 10:26 AM - Eregon (Benoit Daloze)

I think Process.wait can simply ignore pids of MJIT-workers and just keep waiting, that seems the easiest solution to me.
I'm sure normalperson (Eric Wong) knows more about this than I, so +1 for his solution.

#9 - 06/23/2018 01:42 PM - k0kubun (Takashi Kokubun)

- Status changed from Open to Closed

Applied in changeset commit:ruby-git|359dd59db2512d801bb983f98bede4fc598f139a.

---

spec: skip Process wait specs on MJIT

until [Bug #14867] is fixed. I want to start running CI with MJIT
enabled before fixing the problem.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63731 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

#10 - 06/23/2018 02:29 PM - k0kubun (Takashi Kokubun)

- Status changed from Closed to Open

#11 - 06/23/2018 04:02 PM - normalperson (Eric Wong)

- File 0001-hijack-SIGCHLD-handler-for-internal-use.patch added

Tested on FreeBSD 11.1 and Linux

https://80x24.org/spew/20180623155826.29681-1-e@80x24.org/raw

#12 - 06/24/2018 04:04 AM - normalperson (Eric Wong)

File 0001-hijack-SIGCHLD-handler-for-internal-use.patch added

Scratch that, slightly buggy. Will fix in a bit.

#13 - 06/24/2018 12:22 PM - normalperson (Eric Wong)

File 0001-hijack-SIGCHLD-handler-for-internal-use.patch added

Scratch that, slightly buggy. Will fix in a bit.

Testing with this amendment:
https://80x24.org/spew/20180624115501.19651-1-e@80x24.org/raw

It's a bit tricky, and I introduced yet another sleep function
(rb_thread_sleep_interruptible) to make it work.

Original problem in my patch was holding a native mutex while
forking is dangerous, so I went to normal ruby sleep methods
(same as Mutex and autoload).

However, it's impossible to resume main thread from signal
handler because rb_threadptr_execute_interrupts always restores
th->status after trap handlers. In other words, this doesn't work:

```
main = Thread.current
trap(:USR1) { main.run }
Thread.new { sleep 0.1; Process.kill(:USR1, $$) }
sleep
```

So I tried moving rb_sigchld/rb_waitpid_all into timer-thread;
but that's racy and not doable unless the Ruby thread holds
a native mutex while sleeping on a native condvar (w/o GVL).
And that brings us back to the original problem...
(Well, I could use futex :P)

Now, back to using the main thread to handle SIGCHLD...

It seems to be working.

I suspect there's an old bug here, since we lose wakeups when
the timer-thread is stopped:

```
--- a/process.c
+++ b/process.c
@@ -1348,6 +1348,9 @@ after_exec_non_async_signal_safe(void)
 {
     rb_thread_reset_timer_thread();
     rb_thread_start_timer_thread();
+    if (rb_signal_buff_size()) {
+        rb_thread_wakeup_timer_thread();
+    }
 }

 static void
```

But maybe we can also block signals before stopping timer-thread.

#14 - 06/24/2018 12:42 PM - normalperson (Eric Wong)

```
+++ b/process.c
@@ -1348,6 +1348,9 @@ after_exec_non_async_signal_safe(void)
 {
     rb_thread_reset_timer_thread();
     rb_thread_start_timer_thread();
+    if (rb_signal_buff_size()) {
+        rb_thread_wakeup_timer_thread();
+    }
```

NAK. Reordering should be sufficient:
https://bugs.ruby-lang.org/issues/14868 r63741

#15 - 06/25/2018 12:33 AM - normalperson (Eric Wong)

I'm pretty sure this series is sorted (at least on Linux +
FreeBSD) And with [PATCH 3/5] probably won't break on Win32
(does it have SIGCHLD?). At least I tried to make it
easy-to-fix for win32 maintainers.

I plan to squash patches 1-3 before committing, and separate out 4 and 5

Shall I commit?

The following changes since commit 8c8247c6eb554d32fdb03f199df9c14d18bf71dc:

- 2018-06-25 (2018-06-24 22:08:16 +0000)

are available in the Git repository at:

git://80x24.org/ruby.git chld-hijack

for you to fetch changes up to 02cd6168da97d63a0a9e181706ad3c1732328267:

Revert "spec: skip Process wait specs on MJIT" (2018-06-25 00:18:54 +0000)

Eric Wong (5):
hijack SIGCHLD handler for internal use
https://80x24.org/spew/20180625002220.29490-2-e@80x24.org/raw
fix SIGCHLD hijacking race conditions
https://80x24.org/spew/20180625002220.29490-3-e@80x24.org/raw
mjit.c: allow working on platforms without SIGCHLD
https://80x24.org/spew/20180625002220.29490-4-e@80x24.org/raw
Revert "test_process.rb: skip tests for Bug 14867"
https://80x24.org/spew/20180625002220.29490-5-e@80x24.org/raw
Revert "spec: skip Process wait specs on MJIT"
https://80x24.org/spew/20180625002220.29490-6-e@80x24.org/raw

```
mjit.c                                |  68 ++++++++++---
process.c                             | 181 +++++++++++++++++++++++++-------
signal.c                              |  56 ++++++++---
spec/mspec/lib/mspec/guards/feature.rb |   6 --
spec/ruby/core/process/wait2_spec.rb   |  26 +++--
spec/ruby/core/process/wait_spec.rb    | 122 ++++++++++++++----------
spec/ruby/core/process/waitall_spec.rb |  66 ++++++++-----
test/ruby/test_process.rb              |   3 -
thread.c                              |  13 +++
vm_core.h                             |   3 +
10 files changed, 355 insertions(+), 189 deletions(-)
```

#16 - 06/25/2018 02:03 AM - normalperson (Eric Wong)

> Shall I commit?

Well, it works well only for good C extensions and programs
which avoids NFS or slow HDD in the main thread.

However, handling SIGCHLD in timer-thread will have the
advantage of working well even when main thread is stuck in
an uninterruptible state.

So maybe I will retry moving waitpid to work in timer-thread as
I mentioned [ruby-core:87621]. That will allow waitpid to
function in sub-Threads if the main thread is stuck in 'D'
state. The race conditions with timer-thread can be avoided
with futex in Linux and pipe on other OSes. Or maybe use
interrupt_lock like native_sleep for thread_pthread.c

#17 - 06/25/2018 01:14 PM - k0kubun (Takashi Kokubun)

- Assignee changed from k0kubun (Takashi Kokubun) to normalperson (Eric Wong)

Thanks to work on this. The skipped tests you reverted was for running with "./configure cppflags=-DMJIT_FORCE_ENABLE", and some of them failed and some SEGV-ed on my machine.
Here is the log:
https://gist.githubusercontent.com/k0kubun/9ce75493545fa7aa5b6db89f43dd592c/raw/c72af68c5be760675e6fecc7db0e2bc9facfa834/chld-hijack.log

I don't investigate the cause yet but it doesn't happen at least on ko1's CI server. So probably it's not ready to be committed yet.

#18 - 06/25/2018 11:22 PM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> Thanks to work on this. The skipped tests you reverted was for running with "./configure cppflags=-DMJIT_FORCE_ENABLE", and some of them failed and some SEGV-ed on my machine.
> Here is the log:
> https://gist.githubusercontent.com/k0kubun/9ce75493545fa7aa5b6db89f43dd592c/raw/c72af68c5be760675e6fecc7db0e2bc9facfa834/chld-hijack.log

Thanks, investigating... I think removing the
`rb_native_mutex_destroy(&vm->waitpid_lock);'
call fixes the bug at shutdown...

ext/pty overrides SIGCHLD, so that might have to be changed
somehow or made incompatible with MJIT (smaller price than
deprecating Process.wait)

#19 - 06/26/2018 01:42 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> takashikkbn@gmail.com wrote:
>
> > Thanks to work on this. The skipped tests you reverted was for running with "./configure cppflags=-DMJIT_FORCE_ENABLE", and some of them failed and some SEGV-ed on my machine.
> > Here is the log:
> > https://gist.githubusercontent.com/k0kubun/9ce75493545fa7aa5b6db89f43dd592c/raw/c72af68c5be760675e6fecc7db0e2bc9facfa834/chld-hijack.log
>
> Thanks, investigating... I think removing the
> `rb_native_mutex_destroy(&vm->waitpid_lock);'
> call fixes the bug at shutdown...

Tweaked ordering to favor targetted PID and preserve errno,
seems closer to passing tests (takes forever on my system)
https://80x24.org/spew/20180625235051.66045-9-e@80x24.org/raw

On top of cleanups + timer-thread waitpid handling:
https://80x24.org/spew/20180625235051.66045-5-e@80x24.org/raw
https://80x24.org/spew/20180625235051.66045-6-e@80x24.org/raw

"git request-pull" output + links to all patches:
https://80x24.org/spew/20180625235051.66045-1-e@80x24.org/

#20 - 06/26/2018 01:42 PM - normalperson (Eric Wong)

```
Up to 14 patches, now, but I think most problems are solved,
at least on Linux.  FreeBSD 11.1 is close

Links within:

https://80x24.org/spew/20180626093817.1533-1-e@80x24.org/

One potential problem is stuff like:

pids = []
pids << Process.fork { Process.exit! 2 }
pids << Process.fork { Process.exit! 1 }
pids << Process.fork { Process.exit! 0 }
Process.waitall

Can return extra PIDs with MJIT enabled, because exit! cannot
```

```
cleanup and wait for any processes.  I haven't been able to
reproduce the problem lately, though...

* Implemented rb_grantpt for ext/pty use (to temporarily disable SIGCHLD)

* reinstated non-SIGCHLD code path for rb_waitpid

* split Process.wait(PID > 0) and Process.wait(PID <= 0) threads
to prioritize (PID > 0) waiters.  This was an important change.

* avoid spurious wakeups in main thread if no trap(:CHLD) users
(needed to get OpenSSL::PKey::RSA.new working with rubygems test
and test/-ext-/gvl/test_last_thread.rb to pass on Linux)

In retrospect, we could probably be avoiding spurious main thread
wakeups for all signals when nobody registers trap(sig).
However, this isn't effective on FreeBSD...  We may need to block
non-VTALRM signals in main thread, and only enable interrupts in
tiner-thread.

The actual history is a mess, will need to cleanup and do more
documenting before committing.  MJIT takes forever to test :<

I am happy that ext/pty/pty.c is cleaner with the rb_grantpt change, now :)

The following changes since commit 4444025d16ae1a586eee6a0ac9bdd09e33833f3c:

mjit_compile.inc.erb: drop unnecessary variable (2018-06-25 14:15:26 +0000)

are available in the Git repository at:

git://80x24.org/ruby.git mjit-chld-wip

for you to fetch changes up to 6545efeee931baa42f38d9b86941c0e5d30c1044:

process.c (rb_waitpid): reimplement non-SIGCHLD code path (2018-06-26 09:09:21 +0000)
```

#21 - 06/27/2018 04:17 AM - k0kubun (Takashi Kokubun)

Hi Eric. After you committed r63758 and r63760, 2 tests for Process.wait2 failed in ko1's CI server:
http://ci.rvm.jp/results/trunk-mjit@silicon-docker/1051216
https://gist.github.com/ko1/90104a101b9fa2d7adbd880dbead6beb

By the way, are you interested in receiving test failure alerts by ko1's CI servers? It's sometimes too noisy since it sends a lot of emails when it fails, but it's convenient to know some random test failures (it runs tests many times for the same commit).

#22 - 06/27/2018 10:18 AM - normalperson (Eric Wong)

r63764 probably fixes the specs better. At least my misguided attempt to
set ECHILD is removed.

Feel free to add me to CI notifications and Cc: me directly when mailing list or
Redmine are broken. I'm used to getting thousands of emails a day :>

#23 - 06/27/2018 01:13 PM - k0kubun (Takashi Kokubun)

Since r63758, mswin builds are failing like https://ci.appveyor.com/project/ruby/ruby/build/1.0.8700, and the build error is changed like https://ci.appveyor.com/project/ruby/ruby/build/1.0.8703 since r63761. Could you check this? Probably #14871 is related as well.

#24 - 06/27/2018 01:33 PM - nobu (Nobuyoshi Nakada)

Since r63758, rb_waitpid() hangs up frequently (not always) on macOS (darwin17).
Seems SIGCHLD is not delivered.

#25 - 06/27/2018 03:32 PM - normalperson (Eric Wong)

> Up to 14 patches, now, but I think most problems are solved,
> at least on Linux. FreeBSD 11.1 is close

Seems OK, hopefully it doesn't cause regressions for platforms
w/o SIGCHLD.

Squashed to 3 commits on r63758-r63760

hijack SIGCHLD handler for internal use
Revert "test_process.rb: skip tests for Bug 14867"
Revert "spec: skip Process wait specs on MJIT"

#26 - 06/27/2018 04:07 PM - MSP-Greg (Greg L)

normalperson (Eric Wong),

Eric,

Thanks for all the work on this and elsewhere. Some issues with Windows, both MSWin & MinGW.

https://bugs.ruby-lang.org/issues/14872 MSWin
https://bugs.ruby-lang.org/issues/14873 MinGW

Greg

#27 - 06/28/2018 11:06 AM - Eregon (Benoit Daloze)

Would it be simpler to track a set of pids created by MJIT, ignore those in waitpid() and synchronize around both creating GCC processes and when checking the result of waitpid()?
Signals feel unreliable to me.

#28 - 06/28/2018 03:42 PM - normalperson (Eric Wong)

nobu@ruby-lang.org wrote:

> Since r63758, rb_waitpid() hangs up frequently (not always) on macOS (darwin17).
> Seems SIGCHLD is not delivered.

Does removing some of the pthread_sigmask changes from that
alleviate the problem? I try to force SIGCHLD to only be
delivered to timer-thread (to avoid spurious wakeups for
test/-ext-/gvl/test_last_thread.rb)

#29 - 06/29/2018 02:32 AM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> Would it be simpler to track a set of pids created by MJIT, ignore those in waitpid() and synchronize around both creating GCC processes and when checking the result of waitpid()?

Not possible, if there's a thread running in waitpid(-1, 0),
it can steal the result of waitpid(mjit_used_by_pid).

> Signals feel unreliable to me.

Maybe there's a race condition somewhere (including kernel).

#30 - 06/29/2018 05:52 AM - normalperson (Eric Wong)

I can't find output of yes-test-all in any of the
links from r63763..r63771 on
http://ci.rvm.jp/results/trunk-mjit@silicon-docker

Is it because r63763? ("give up insn attr handles_frame",
commit 6b534134a78e3e43c344682c3585e1abab015216).

It seems to cause problems for me with MJIT_FORCE_ENABLE

#31 - 06/29/2018 08:12 AM - spatulasnout (B Kelly)

Eric Wong wrote:

> eregontp@gmail.com wrote:
>
> > Would it be simpler to track a set of pids created by MJIT, ignore those in
> > waitpid() and synchronize around both creating GCC processes and when checking
> > the result of waitpid()?

> Not possible, if there's a thread running in waitpid(-1, 0),
> it can steal the result of waitpid(mjit_used_by_pid).

Could it help at all to delegate to a single "JIT Controller" daemon?

E.g.

- At startup, Ruby forks a single JIT Controller daemon,
  meant to persist for the duration of the Ruby process
- JITC daemon removes itself from Ruby's process group
- Ruby communicates with JITC daemon via pipes, and JITC
  in turn spawns and waits for JIT compiler processes

Just a thought. (Apologies if not applicable; I'm not well-versed in
the current JIT architecture.)

Regards,

Bill

#32 - 06/29/2018 09:42 AM - normalperson (Eric Wong)

Bill Kelly billk@cts.com wrote:

> Could it help at all to delegate to a single "JIT Controller" daemon?

That's akr's initial idea [ruby-core:87608] but that uses more
memory. We already have resource problems with timer-thread
creation in the test suite.

A SIGCHLD-based implementation will also integrate better for
proposed features such as auto-Fibers.

Maybe platforms with lossy/racy signals (OSX? [ruby-core:87657])
can put timer-thread in polling mode when there are threads
calling waitpid. Perhaps win32 can do that, too, since their
timer-thread is always polling regardless.

I'm pretty sure all other BSD, Solaris and Linux systems get
SIGCHLD correctly.

#33 - 06/29/2018 10:52 AM - normalperson (Eric Wong)

nobu@ruby-lang.org wrote:

> Since r63758, rb_waitpid() hangs up frequently (not always) on macOS (darwin17).
> Seems SIGCHLD is not delivered.

Does moving timer-thread into polling mode when there are
waitpid threads help?

```
diff --git a/signal.c b/signal.c
index 238679e1bd..ffdf645e9e 100644
--- a/signal.c
+++ b/signal.c
@@ -1066,7 +1066,7 @@ void ruby_waitpid_all(rb_vm_t *); /* process.c */
void
ruby_sigchld_handler(rb_vm_t *vm)
{
-    if (sigchld_hit) {
+    if (sigchld_hit || SIGCHLD_LOSSY) {
sigchld_hit = 0;
ruby_waitpid_all(vm);
}
diff --git a/thread_pthread.c b/thread_pthread.c
index 6ac7728ad9..004c9ecfe0 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
```

```
@@ -1375,6 +1375,13 @@ timer_thread_sleep(rb_global_vm_lock_t* gvl)

 need_polling = !ubf_threads_empty();

+    if (SIGCHLD_LOSSY) {
+        rb_vm_t *vm = container_of(gvl, rb_vm_t, gvl);
+        if (!list_empty(&vm->waiting_pids) || !list_empty(&vm->waiting_grps)) {
+            need_polling = 1;
+        }
+    }
+
 if (gvl->waiting > 0 || need_polling) {
    /* polling (TIME_QUANTUM_USEC usec) */
    result = poll(pollfds, 1, TIME_QUANTUM_USEC/1000);
diff --git a/vm_core.h b/vm_core.h
index bdbe87287b..6ea3293d70 100644
--- a/vm_core.h
+++ b/vm_core.h
@@ -100,6 +100,12 @@
  #  define RUBY_SIGCHLD    (0)
  #endif

+#if defined(__APPLE__)
+#  define SIGCHLD_LOSSY (1)
+#else
+#  define SIGCHLD_LOSSY (0)
+#endif
+
  #ifdef HAVE_STDARG_PROTOTYPES
  #include <stdarg.h>
  #define va_init_list(a,b) va_start((a),(b))
```

#34 - 06/29/2018 11:25 AM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

> eregontp@gmail.com wrote:
>
> > Would it be simpler to track a set of pids created by MJIT, ignore those in waitpid() and synchronize around both creating GCC processes and when checking the result of waitpid()?
>
> Not possible, if there's a thread running in waitpid(-1, 0),
> it can steal the result of waitpid(mjit_used_by_pid).

Thank you for the reply.
I am not sure to understand why it is not possible, could you elaborate a bit more?
Could the thread running waitpid(-1, 0) then pass that wait result to the MJIT thread, and retry waitpid(-1, 0) in such a case?
(with careful synchronization as PIDs can be reused, to ensure to correctly classify as MJIT/non-MJIT pid)

#35 - 06/29/2018 06:52 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> normalperson (Eric Wong) wrote:
>
> > eregontp@gmail.com wrote:
> >
> > > Would it be simpler to track a set of pids created by MJIT, ignore those in waitpid() and synchronize around both creating GCC processes and when checking the result of waitpid()?
> >
> > Not possible, if there's a thread running in waitpid(-1, 0),
> > it can steal the result of waitpid(mjit_used_by_pid).
>
> Thank you for the reply.
> I am not sure to understand why it is not possible, could you elaborate a bit more?
> Could the thread running waitpid(-1, 0) then pass that wait result to the MJIT thread, and retry waitpid(-1, 0) in such a case?
> (with careful synchronization as PIDs can be reused, to ensure to correctly classify as MJIT/non-MJIT pid)

I don't think that synchronization is possible; or it'd be far more complex
than it is now. Calling waitpid to put a thread to sleep can't atomically

release a mutex like pthread_cond_wait can, and potentially sleeping on two
different non-FD objects is deadlock-prone.

Right now, any sleeping is done with cond_wait in the Ruby or MJIT thread.
The timer thread has no changes in the way it sleeps, as it only does
waitpid with WNOHANG

If SIGCHLD exists but it's reliability is a problem on your platform,
perhaps try the polling patch I proposed at [ruby-core:87682]

#36 - 06/29/2018 10:20 PM - MSP-Greg (Greg L)

normalperson (Eric Wong)

FYI, both mswin & mingw are building. I don't believe mswin runs JIT tests, but when I run them locally on MinGW, I just get a frozen flashing cursor. No output at all.

I'm happy to help with the problem, but I'm not quite sure what to do...

Thanks, Greg

#37 - 06/29/2018 10:52 PM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> FYI, both mswin & mingw are building. I don't believe mswin
> runs JIT tests, but when I run them locally on MinGW, I just
> get a frozen flashing cursor. No output at all.

Right, the fallback code for non-SIGCHLD doesn't benefit MJIT.
However, maybe polling will work, since timer-thread in
thread_win32.c is always polling anyways...

> I'm happy to help with the problem, but I'm not quite sure
> what to do...

Can you try the following patch? It is a newer version of
[ruby-core:87682] and needs r63793[*]

https://80x24.org/spew/20180629224058.20652-1-e@80x24.org/raw

[*] ("signal.c: use ATOMIC_EXCHANGE for sigchld_hit")
r63793 may solve OSX problem with CI machines, as I suspect
pthread_sigmask is broken on that platform. Still waiting
on results from rubyci.org.

#38 - 06/30/2018 12:27 AM - MSP-Greg (Greg L)

normalperson (Eric Wong)

Thank you for all the work & help.

With the patch, ruby-loco is now green on 63794.

Greg

#39 - 06/30/2018 12:53 AM - normalperson (Eric Wong)

- Status changed from Open to Closed

Applied in changeset trunk|r63795.

---

use SIGCHLD_LOSSY to enable waitpid polling mode

Some systems lack SIGCHLD or have incomplete SIGCHLD
implementations. So enable polling mode for them.

[ruby-core:87705] [Bug #14867]

#40 - 06/30/2018 01:15 AM - MSP-Greg (Greg L)

- File JIT-test-all.log added

normalperson (Eric Wong),

Long story, bouncing between local building & Appveyor, etc.

I was mistaken, almost all of the TestJITs test failed, both when run parallel and when retried.

Sorry for the mixup. I've attached a parsed log from test-all.

Thanks, Greg

#41 - 06/30/2018 02:12 AM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> File JIT-test-all.log added
>
> I was mistaken, almost all of the TestJITs test failed, both when run parallel and when retried.
>
> Sorry for the mixup. I've attached a parsed log from test-all.

Is that with the SIGCHLD_LOSSY (polling) patch applied on top of r63794?
JIT-test-all.log only says r63794.

I committed the the SIGCHLD_LOSSY (polling) patch as r64795
after your previous message. Is that tested? I'm not sure if
the defined(**WIN32**) guard is correct or if defined(_WIN32)
also needs to be there, or if the waitpid() implementation in
win32/win32.c works in timer thread...

Maybe usa (Usaku NAKAMURA) can help.

#42 - 06/30/2018 03:19 AM - MSP-Greg (Greg L)

normalperson (Eric Wong) wrote:

> Is that with the SIGCHLD_LOSSY (polling) patch applied on top of r63794?
> JIT-test-all.log only says r63794.

Yes. I added the patch file to the build. Since removed for r64795.

> I committed the the SIGCHLD_LOSSY (polling) patch as r64795
> after your previous message.

Again, sorry for that. I saw the green, and forgot that I left some code that subtracted only NJIT failures from the total to determine green/red. IOW, it ran the tests, but disregarded them.

> I'm not sure if the defined(**WIN32**) guard is correct or if defined(_WIN32)
> also needs to be there

I suspect that mjit.c is some of the newest code in the repo, and quick search seemed to show
#ifdef _WIN32 or defined(_WIN32), but I have no idea.

> or if the waitpid() implementation in win32/win32.c works in timer thread...

All I know is config shows checking for waitpid... (cached) yes

Wish I could be of more help, and again, sorry for the incorrect info, Greg

#43 - 06/30/2018 04:53 AM - MSP-Greg (Greg L)

- File mjit_test-all_63796.log added

normalperson (Eric Wong)

Attached is the log for r63796. Still have the fails...

Thanks again, Greg

#44 - 06/30/2018 06:32 AM - normalperson (Eric Wong)

OK, I realized mjit.c and process.c have totally different ideas
of what a PID is on win32. For process.c, win32/win32.c
abstracts things while mjit.c uses the raw OS return value (I think).

So trying r63797...

If that doesn't work, I trust someone (usa or nobu) to figure
this out easily.

#45 - 06/30/2018 07:22 AM - normalperson (Eric Wong)

> OK, I realized mjit.c and process.c have totally different ideas
> of what a PID is on win32. For process.c, win32/win32.c
> abstracts things while mjit.c uses the raw OS return value (I think).
>
> So trying r63797...

Wait, that won't work because timeout is zero :x I think
somebody with win32 knowledge just need to check the return
value and retry.

#46 - 06/30/2018 04:27 PM - MSP-Greg (Greg L)

normalperson (Eric Wong)

Running TestJIT locally, using the last good build ruby 2.6.0dev (2018-06-27 trunk 63757) [x64-mingw32], no files are left in TMP.

Running with a recent release (with failures), over 60 files remain, all approx. 56 mb in size, with names like _ruby_mjit_*.h.gch

Don't know if it sheds any more info on the issue...

Thanks, Greg

#47 - 07/02/2018 09:42 PM - normalperson (Eric Wong)

- Assignee changed from normalperson (Eric Wong) to k0kubun (Takashi Kokubun)

- Status changed from Closed to Assigned

k0kubun / usa / nobu: can you check the return value handling of
_WIN32 waitpid_sys/WaitForSingleObject in process.c as of r63797?
It needs to retry on the timeout==0 (WNOHANG) case.

Thanks.

#48 - 07/02/2018 11:33 PM - normalperson (Eric Wong)

nobu@ruby-lang.org wrote:

> Since r63758, rb_waitpid() hangs up frequently (not always) on macOS (darwin17).
> Seems SIGCHLD is not delivered.

I guess sigsetjmp in r63803 did not help? I don't think it
would, once a thread starts, the signal mask won't change.

My polling-for-lossy-SIGCHLD implementation was incomplete,
r63829 should complete it for **APPLE**. Seems like a strange
kernel bug, but all the other BSDs seem fine...

Only thing left is somebody to figure out the non-blocking
waitpid/WaitForSingleObject [ruby-core:87759] for win32.

#49 - 07/04/2018 10:54 PM - k0kubun (Takashi Kokubun)

- Assignee changed from k0kubun (Takashi Kokubun) to normalperson (Eric Wong)

As it was reverted on r63852 by naruse (Yui NARUSE), assigning this back to Eric. I'm guessing it was reverted since it was not portable mainly for Windows. Whenever you commit things, you can check at least http://mswinci.japaneast.cloudapp.azure.com/vc12-x64/ruby-trunk/recent.html status and revert it if it fails. When you want a fast feedback, https://ci.appveyor.com/project/ruby/ruby/history can show you the result more quickly.

I really appreciate your work on this, but sometimes I can't immediately take time to fix your work for Windows and in that case master CI failure for a

long time disrupts other committers.

#50 - 07/05/2018 01:48 AM - MSP-Greg (Greg L)

normalperson (Eric Wong)

I'm not sure what you look at on the web (GitHub, Appveyor), so briefly:

1. The mswin builds do not test MJIT, but mingw (ruby-loco) does. Note that mingw tests 3 time a day, so I may not know what revision caused what. The mingw test history is at https://ci.appveyor.com/project/MSP-Greg/ruby-loco/history.

2. When you first added commits (r63758), mingw wouldn't build, then once that was fixed, tests crashed. As of r63794, it built, tests completed, but all JIT tests failed (they previously passed). At that point, I removed the JIT results from the pass/fail logic.

3. The mswin build was functioning up until r63816. The changes in r63820 (get rid of a compiler warning of VC) interacted with your commits to cause the mswin build to lock up. If that interaction can be fixed, mswin may pass.

4. If I'm not AFK, I can certainly start a new build job to test a particular commit. I can also easily add a patch to the build system.
   Greg.mpls@gmail.com

I don't know if this info helps, but since you and everyone else here have always been helpful with windows issues, I certainly don't want Windows testing to jam up trunk improvements... Thanks, Greg

#51 - 07/05/2018 03:12 AM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> I'm not sure what you look at on the web (GitHub, Appveyor), so briefly:
>
> 1. The mswin builds do not test MJIT, but mingw (ruby-loco) does. Note that mingw tests 3 time a day, so I may not know what revision caused what. The mingw test history is at https://ci.appveyor.com/project/MSP-Greg/ruby-loco/history.

I only see plain-text and sometimes tables. Javascript doesn't
show up to me, so that has no data for me. I tried watching
rubyci.org and managed to fix OSX in r63829. I wasn't sure
which was the win32 build and there were other failures
unrelated to RUBYLIB, but I will look at k0kubun's link.

> 1. When you first added commits (r63758), mingw wouldn't build, then once that was fixed, tests crashed. As of r63794, it built, tests completed, but all JIT tests failed (they previously passed). At that point, I removed the JIT results from the pass/fail logic.

OK, I will keep that in mind; r63794 only changed a test;
so I'm not sure about JIT.

> 1. The mswin build was functioning up until r63816. The changes in r63820 (get rid of a compiler warning of VC) interacted with your commits to cause the mswin build to lock up. If that interaction can be fixed, mswin may pass.

Maybe r63820 needs to be reverted separately.
I think the problem is mjit.c code has a different
idea of PID from process.c for win32.

#52 - 07/05/2018 03:32 AM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> As it was reverted on r63852 by naruse (Yui NARUSE), assigning this back
> to Eric. I'm guessing it was reverted since it was not
> portable mainly for Windows. Whenever you commit things, you
> can check at least
> http://mswinci.japaneast.cloudapp.azure.com/vc12-x64/ruby-trunk/recent.html

OK, I will keep an eye on that one (on rubyci.org page, it
wasn't obvious to me vc12 is Windows)

> status and revert it if it fails. When you want a fast
> feedback, https://ci.appveyor.com/project/ruby/ruby/history
> can show you the result more quickly.

Requires JS, so I can't view with w3m or lynx.

> I really appreciate your work on this, but sometimes I can't immediately take time to fix your work for Windows and in that case master CI failure for a long time disrupts other committers.

Anyways I'll keep an eye on the vc12 and try to keep my changes to non-w32.

#53 - 07/05/2018 04:47 AM - MSP-Greg (Greg L)

Importantly, mswin (ruby's vc12) passed r63856.

Unfortunately, ruby-loco locked up on test-all. So, before the reversion (r63852), test-all completed, but all JIT tests failed. After the reversion, all JIT tests passed.

Ok, I'll remember the no js...

> OK, I will keep that in mind; r63794 only changed a test;

That's why I mentioned the 'builds 3 times a day' (3:00, 9:00 and 16:00 UTC, or Noon, 6:00 P and 1:00 A JST). The Ruby core mswin CI runs on every commit.

In the build previous to r63794, which was r63790, the tests crashed. As of r63794, they completed.

Anyway, since mingw and mswin mean nothing to people unfamiliar with Windows builds, how about I'll call the Ruby core CI build 'mswin', and the mingw build 'ruby-loco' (which is my build)? mswin is straight MSVC compiler, ruby-loco is using gcc 7.3.0 ported to windows via the MSYS2/MinGW project.

Gotta go AFK. Thanks, Greg

#54 - 07/05/2018 05:42 AM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> Importantly, mswin (ruby's vc12) passed r63856.

Good to know. I saw your message before http://mswinci.japaneast.cloudapp.azure.com/vc12-x64/ruby-trunk/recent.html got updated, I guess there is a delay.

> Unfortunately, ruby-loco locked up on test-all. So, before the reversion (r63852), test-all completed, but all JIT tests failed. After the reversion, all JIT tests passed.

Any compiler warnings or helpful output before lokcup?

> Ok, I'll remember the no js...
>
> > OK, I will keep that in mind; r63794 only changed a test;
>
> That's why I mentioned the 'builds 3 times a day' (3:00, 9:00 and 16:00 UTC, or Noon, 6:00 P and 1:00 A JST). The Ruby core mswin CI runs on every commit.
>
> In the build previous to r63794, which was r63790, the tests crashed. As of r63794, they completed.
>
> Anyway, since mingw and mswin mean nothing to people unfamiliar with Windows builds, how about I'll call the Ruby core CI build 'mswin', and the mingw build 'ruby-loco' (which is my build)? mswin is straight MSVC compiler, ruby-loco is using gcc 7.3.0 ported to windows via the MSYS2/MinGW project.

OK. So I wonder if ruby-loco tries to define SIGCHLD and fails because of it. The different behaviors of various Windows builds is confusing, because usa (Usaku NAKAMURA) was happy about r63790.

Does the following help? If WAITPID_USE_SIGCHLD is zero, it should fall back to old code and none of my changes should have

any effect.

```
--- a/vm_core.h
+++ b/vm_core.h
@@ -110,6 +110,11 @@
/* define to 0 to test old code path */
 #define WAITPID_USE_SIGCHLD (RUBY_SIGCHLD || SIGCHLD_LOSSY)

+#if defined(__MINGW32__) || defined(_WIN32)
+#  undef WAITPID_USE_SIGCHLD
+#  define WAITPID_USE_SIGCHLD (0)
+#endif
+
 #ifdef HAVE_STDARG_PROTOTYPES
 #include <stdarg.h>
 #define va_init_list(a,b) va_start((a),(b))
```

Thanks for your patience.

#55 - 07/05/2018 02:02 PM - MSP-Greg (Greg L)

normalperson (Eric Wong)

Thanks for the patch, I added it, and a build with r63859 locked/froze on test-all. I'm going to see if I can get any more info by building/testing locally. I'm surprised and wondering if something reverted wasn't re-applied, as ruby-loco should be completing test-all with MJIT tests failing…

One question – would the config portion of a ruby-loco build be helpful for you to see? I can attach it as a text file…

Some more Windows background:

The ruby/ruby repo on GitHub runs two CI builds. Travis builds on Unbuntu, and Appveyor builds on Windows.

The Appveyor build is what I refer to as mswin. If both Travis & Appveyor pass, then the ruby/ruby mirror on GitHub shows a green pass flag for each commit. When either or both fail, people have to look further to see what the cause of the failure was.

The Appveyor mswin build's history is shown at https://ci.appveyor.com/project/ruby/ruby/history.

RubyCI.org shows has many builds reported on, and the link to http://mswinci.japaneast.cloudapp.azure.com/vc12-x64/ruby-trunk/recent.html is one of them. Generally, if mswin passes, so should this build.

So, mswin is used by committers to check whether the Widows VC build succeeded.

Ruby-loco is built using the same build system (MSYS2/MinGW) that the majority of Windows Ruby releases use. These releases/builds are available for download, as many (new) Windows users/coders are unfamiliar with building/compiling.

Many gems/repos test on Appveyor using the same builds, and many of those also test against trunk.

Some use ruby-loco for that, as I believe it is the only trunk build available with full testing. The way I have the build script set up, the 'trunk download' used for CI testing will always be the most recent passing build. Because of that, I have an interest in making sure it continues to build and pass all tests.

Thanks again, Greg

#56 - 07/05/2018 09:22 PM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> normalperson (Eric Wong)
>
> Thanks for the patch, I added it, and a build with r63859 locked/froze on test-all. I'm going to see if I can get any more info by building/testing locally. I'm surprised and wondering if something reverted wasn't re-applied, as ruby-loco should be completing test-all with MJIT tests failing…
>
> One question – would the config portion of a ruby-loco build be helpful for you to see? I can attach it as a text file…

Maybe, just about any text info helps.

Just to reiterate, "make test" (not "test-all") passes?

Do you get any output at all from test-all? That would be most
useful. Perhaps use "TESTS=-v" in the env for test-all to get
more verbose output as to which test it's stuck on
("make test-all TESTS=-v")

It's a major accessibility problem for Appveyor to require JS to

view plain text :< Can you get them to fix it?

The patch below should dump out more info about the system
which might help me diagnose it. If the "ruby" executable
builds, it should run and show some info we care about for
this issue:

You can also download the patch at:
https://80x24.org/spew/20180705204805.7670-1-e@80x24.org/raw

Usage: ./ruby --jit -e exit

Output (on GNU/Linux):

process.c: sigchld=17 getpid.size=4 rb_pid_t.size=4
mjit.c: sigchld=17 getpid.size=4 pid_t.size=4
mjit.c: !WIN32 !MINGW32_ !**MINGW64** !**CYGWIN**

I expect at least some of those '!' to be gone on your system.

```
diff --git a/mjit.c b/mjit.c
index 232424fa5d..07221fc714 100644
--- a/mjit.c
+++ b/mjit.c
@@ -1503,6 +1503,31 @@ mjit_init(struct mjit_options *opts)
rb_id_table_foreach(RCLASS_CONST_TBL(rb_cObject), valid_class_serials_add_i, NULL);
}

+    fprintf(stderr, "mjit.c: sigchld=%d getpid.size=%"PRIuSIZE" pid_t.size=%"PRIuSIZE"\n",
+            RUBY_SIGCHLD, sizeof(getpid()), sizeof(pid_t));
+    fprintf(stderr, "mjit.c: %s %s %s %s\n",
+#if defined(_WIN32)
+            "_WIN32",
+#else
+            "!_WIN32",
+#endif
+#if defined(__MINGW32__)
+            "__MINGW32__",
+#else
+            "!__MINGW32__",
+#endif
+#if defined(__MINGW64__)
+            "__MINGW64__",
+#else
+            "!__MINGW64__",
+#endif
+#if defined(__CYGWIN__)
+            "__CYGWIN__"
+#else
+            "!__CYGWIN__"
+#endif
+        ); /* fprintf */
+
/* Overwrites RUBY_DESCRIPTION constant */
rb_const_remove(rb_cObject, rb_intern("RUBY_DESCRIPTION"));
rb_description = rb_usascii_str_new_static(ruby_description_with_jit, strlen(ruby_description_with_jit));
diff --git a/process.c b/process.c
index 12cceba934..9c98c20b3e 100644
--- a/process.c
+++ b/process.c
@@ -8318,4 +8318,7 @@ Init_process(void)
id_exception = rb_intern("exception");

InitVM(process);
+
+    fprintf(stderr, "process.c: sigchld=%d getpid.size=%"PRIuSIZE" rb_pid_t.size=%"PRIuSIZE"\n",
+            RUBY_SIGCHLD, sizeof(getpid()), sizeof(rb_pid_t));
}
```

#57 - 07/06/2018 12:50 AM - MSP-Greg (Greg L)

normalperson (Eric Wong)

I've got builds running both locally and on Appveyor, when they finish I'll have more info.

> Just to reiterate, "make test" (not "test-all") passes?

Actually, btest, test-basic, & test-spec all pass.

> Do you get any output at all from test-all?

If test-all locks/freezes, I don't get any output. Long story, I need to change it. Appveyor has a time limit on build jobs. If the build hits the time limit, that's it, no script can clean up, etc. I'm trying to get test-all to stop locking up...

I added you patch, and I can't yet run ruby --jit -e exit, but I am seeing the line

```
process.c: sigchld=0 getpid.size=8 rb_pid_t.size=8
```

Thanks, and I'll update when the builds finish, Greg

#58 - 07/06/2018 01:01 AM - MSP-Greg (Greg L)

normalperson (Eric Wong)

Below is the out from ruby --jit -e exit

```
C:\Greg\GitHub> ruby --jit -e exit
process.c: sigchld=0 getpid.size=8 rb_pid_t.size=8
mjit.c: sigchld=0 getpid.size=8 pid_t.size=8
mjit.c: _WIN32 __MINGW32__ __MINGW64__ !__CYGWIN__
```

Hope that helps. Both Appveyor build & local build have the same output.

Thanks again, Greg

#59 - 07/06/2018 01:21 AM - MSP-Greg (Greg L)

- File config_ruby-loco_mingw.log added

normalperson (Eric Wong)

Attached is the config portion of ruby-loco mingw build log.

Thanks, Greg

#60 - 07/06/2018 01:22 AM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> Below is the out from ruby --jit -e exit
>
> ```
> C:\Greg\GitHub> ruby --jit -e exit
> process.c: sigchld=0 getpid.size=8 rb_pid_t.size=8
> mjit.c: sigchld=0 getpid.size=8 pid_t.size=8
> mjit.c: _WIN32 __MINGW32__ __MINGW64__ !__CYGWIN__
> ```

```
Thanks.  Were there any compiler warnings building process.c or mjit.c?

> Hope that helps.  Both Appveyor build & local buidl have the same output.

So nothing showed up from test-all?  Did you try verbose (TESTS=-v) output?
Perhaps trying a quick individual tests (e.g. test_pp.rb).
Something like:

    # should be an easy pass, especially if other suites are passing
    make test-all TESTS='test/test_pp.rb -v'

    make test-all TESTS='-v' # entire suite

Or maybe even add "V=1" to the make command to show more info
about what is executing:

    make test-all TESTS=-v V=1

Not sure what version of make you run, but maybe even "-d" flag
can help:
```

```
    make -d test-all TESTS=-v V=1
```

```
But yeah, really strange that even test-spec passes for you.
```

#61 - 07/06/2018 02:03 AM - MSP-Greg (Greg L)

- File test_jit_results.txt added

normalperson (Eric Wong)

> Were there any compiler warnings building process.c or mjit.c?

No. Only warning were for warning: flip-flop is deprecated in another file...

> So nothing showed up from test-all?

Yes, but it's a mess with STDOUT having the line(s) added by the patch.

Running test_jit.rb locally using runner.rb, the tests timeout. Attached is the console output.

BTW, something seems to have changed, as previously I would get emails for any post to a thread I 'watch'. Now, I only get email when I create a message. So I have to keep checking the web page to see your responses...

#62 - 07/06/2018 03:04 AM - normalperson (Eric Wong)

Greg: oh, I think I finally spotted it. The waitpid() macro for
_WIN32 doesn't set a return value. The old code assumed waitpid
always succeeds, and the new code checks the return value.

So I think we need to fake the return value, for now.
Can you try this? (Won't need the other patches)

```
 --- a/mjit.c
 +++ b/mjit.c
 @@ -132,7 +132,7 @@ rb_pid_t ruby_waitpid_locked(rb_vm_t *, rb_pid_t, int *status, int options,
  #define dlclose(handle) (FreeLibrary(handle))
  #define RTLD_NOW  -1

 -#define waitpid(pid,stat_loc,options) (WaitForSingleObject((HANDLE)(pid), INFINITE), GetExitCodeProcess((HAN
DLE)(pid), (LPDWORD)(stat_loc)))
 +#define waitpid(pid,stat_loc,options) (WaitForSingleObject((HANDLE)(pid), INFINITE), GetExitCodeProcess((HAN
DLE)(pid), (LPDWORD)(stat_loc)), (pid))
  #define WIFEXITED(S) ((S) != STILL_ACTIVE)
  #define WEXITSTATUS(S) (S)
  #define WIFSIGNALED(S) (0)
```

#63 - 07/06/2018 04:38 AM - MSP-Greg (Greg L)

Eric,

Well done. The build is 'green' (passing). Script generated result summary is:

```
_____________________________________________________________________________ Test Results
0 Total Failures/Errors                          Build No 912    Job Id kcd9gvig2qdykyi1
  ruby 2.6.0dev (2018-07-06 trunk 63867) [x64-mingw32]
  2018-07-06 04:31:02 UTC

test-all  19246 tests, 2239900 assertions, 0 failures, 0 errors, 110 skips, 110 skips shown

test-spec  3607 files, 27957 examples, 209417 expectations, 0 failures, 0 errors, 0 tagged
mspec      3607 files, 27959 examples, 209315 expectations, 0 failures, 0 errors, 0 tagged

test-basic test succeeded
btest      PASS all 1385 tests
```

Thanks for all your work on this, Greg

#64 - 07/06/2018 06:50 AM - normalperson (Eric Wong)

- Status changed from Assigned to Closed

Applied in changeset trunk|r63869.

mjit.c: fix waitpid macro return value for win32

We started checking return value of waitpid, so it needs
to be correct for win32 platforms for MJIT to work.

Thanks-to: MSP-Greg (Greg L) Greg.mpls@gmail.com

[ruby-core:87832] [Bug #14867]

#65 - 07/06/2018 06:52 AM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> Well done. The build is 'green' (passing).

Thanks! Committed as r63869.

#66 - 07/07/2018 02:58 PM - k0kubun (Takashi Kokubun)

- Status changed from Closed to Assigned

Hi Eric. I fixed [Bug #14892] on r63875, but only the rubyspec check you added is failing with -DMJIT_FORCE_ENABLE like:

```
$ make test-spec
$ /home/k0kubun/src/github.com/ruby/ruby-svn/.ruby-force/miniruby -I/home/k0kubun/src/github.com/ruby/ruby-svn
/lib /home/k0kubun/src/github.com/ruby/ruby-svn/tool/runruby.rb --archdir=/home/k0kubun/src/github.com/ruby/ru
by-svn/.ruby-force --extout=.ext -- /home/k0kubun/src/github.com/ruby/ruby-svn/spec/mspec/bin/mspec-run -B ../
spec/default.mspec
ruby 2.6.0dev (2018-07-07 trunk 63874) +JIT [x86_64-linux]
[| | ================  40%                   | 00:02:03]      0F      0E leaked before wait2 specs: [[31406,
#<Process::Status: pid 31406 exit 0>]]

1)
An exception occurred during: before :all FAILED
Expected [[31406, #<Process::Status: pid 31406 exit 0>]] to be empty
/home/k0kubun/src/github.com/ruby/ruby-svn/spec/ruby/core/process/wait2_spec.rb:18:in `block (3 levels) in <to
p (required)>'
/home/k0kubun/src/github.com/ruby/ruby-svn/spec/ruby/core/process/wait2_spec.rb:16:in `block (2 levels) in <to
p (required)>'
/home/k0kubun/src/github.com/ruby/ruby-svn/spec/ruby/core/process/wait2_spec.rb:3:in `<top (required)>'
[- | ==================100%================== | 00:00:00]      1F      0E

Finished in 103.288794 seconds

3607 files, 28545 examples, 208272 expectations, 1 failure, 0 errors, 0 tagged
uncommon.mk:777: recipe for target 'yes-test-spec' failed
make: *** [yes-test-spec] Error 1
```

So I skipped the check in r63876. I'm not sure if it's reproducible on your environment, but could you check it?

#67 - 07/07/2018 10:52 PM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> Hi Eric. I fixed [Bug #14892] on r63875, but only the rubyspec check you added is failing with -DMJIT_FORCE_ENABLE like:
>
> ```
> $ make test-spec
> ```

Thanks, I think it's a bug of rb_f_exec and it seems the problem
goes away with parallel make. This should fix it:

https://80x24.org/spew/20180707224314.28350-1-e@80x24.org/raw

(taking forever to test, and I have at least two other potential
bugfixes pending)

#68 - 07/08/2018 12:01 AM - normalperson (Eric Wong)

- Status changed from Assigned to Closed

Applied in changeset trunk|r63877.

process.c (rb_f_exec): pause MJIT before replacing process

Non-parallel "make test-spec" caused
spec/ruby/core/process/wait2_spec.rb failures because mspec
uses "exec" in single-process mode, so there's no chance
the post-exec state could know about the MJIT child process
from its pre-exec state.

[ruby-core:87846] [Bug #14867]

#69 - 07/08/2018 12:06 AM - k0kubun (Takashi Kokubun)

Thanks! I confirmed that the patch works well on my machine as well.

#70 - 07/08/2018 01:07 AM - MSP-Greg (Greg L)

normalperson (Eric Wong)

Eric,

Both mswin & ruby-loco failed the test test_sigchld_ignore in r63879.

Having recently made the exact same mistake, ensure runs with a skip. Easy fix would be

```
trap(:CHLD, old) if Signal.list['CHLD']
```

Obviously several ways around it. Thanks again, Greg

#71 - 07/08/2018 01:52 AM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> Both mswin & ruby-loco failed the test test_sigchld_ignore in r63879.
>
> Having recently made the exact same mistake, ensure runs with a skip. Easy fix would be
>
> ```
> trap(:CHLD, old) if Signal.list['CHLD']
> ```

Oops, thanks for the quick spot. r63880 should fix it.
rubyci still hadn't gotten to it r63879 :<

#72 - 07/08/2018 02:45 AM - MSP-Greg (Greg L)

Eric,

r63880 fixed it, Appveyor mswin is green. I will ask them about a 'no javascript' fallback for at least some of the content.

#73 - 10/22/2018 04:37 PM - k0kubun (Takashi Kokubun)

- Status changed from Closed to Assigned

normalperson (Eric Wong) You added RUBY_VM_CHECK_INTS(ec) in mjit.c's stop_worker() on r63855, but that seems to randomly cause SEGV like this:

```
-- C level backtrace information -------------------------------------------
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(rb_print_backtrace+0x14) [0x127357f] vm_dump.c:71
5
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(rb_vm_bugreport) vm_dump.c:985
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(bug_report_end+0x0) [0x10cdae9] error.c:610
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(rb_bug_context) error.c:610
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(sig_do_nothing+0x0) [0x11dacc0] signal.c:998
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(sigsegv) (null):0
/lib/libthr.so.3(0x801fb3954) [0x801fb3954]
/lib/libthr.so.3(0x801fb2eb2) [0x801fb2eb2]
[0x7fffffff193]
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(tokadd_escape+0x140b73) [0x802dfc916] parse.y:551
0
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(tokadd_string) parse.y:5717
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(rb_postponed_job_flush+0x12d) [0x127737d] vm_trac
e.c:1655
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(rb_threadptr_execute_interrupts+0x81) [0x1218a61]
 thread.c:2134
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(stop_worker+0xcb) [0x1134bdb] ./vm_core.h:1773
```

```
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(mjit_finish) mjit.c:726
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(ruby_finalize_1+0x0) [0x10d829b] eval.c:236
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(ruby_cleanup) eval.c:238
/usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(main+0x72) [0x104be92] ./main.c:42
```

https://rubyci.org/logs/rubyci.s3.amazonaws.com/freebsd11zfs/ruby-trunk/log/20181022T153002Z.fail.html.gz

I think rb_threadptr_execute_interrupts() is designed to be called in safe places on VM and calling it from ruby_cleanup() may not be expected. Could you remove RUBY_VM_CHECK_INTS() from the stop_worker() and try to solve the original issue in another way?

#74 - 10/22/2018 06:32 PM - normalperson (Eric Wong)

```
 takashikkbn@gmail.com wrote:
 > @normalperson You added `RUBY_VM_CHECK_INTS(ec)` in mjit.c's `stop_worker()` on r63855, but that seems to r
andomly cause SEGV like this:

 OK, investigating...

 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(sigsegv) (null):0
 > /lib/libthr.so.3(0x801fb3954) [0x801fb3954]
 > /lib/libthr.so.3(0x801fb2eb2) [0x801fb2eb2]
 > [0x7fffffff193]
 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(tokadd_escape+0x140b73) [0x802dfc916] parse.y:
5510
 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(tokadd_string) parse.y:5717
 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(rb_postponed_job_flush+0x12d) [0x127737d] vm_t
race.c:1655

 So rb_postponed_job_flush is calling mjit_copy_job_handler?  I'm
 not sure how we're entering the parser from
 rb_postponed_job_flush, actually.

 If mjit_copy_job_handler is the culprit, maybe having
 mjit_copy_job go out-of-scope from mjit-worker while main thread
 is using the copy job is wrong.  So maybe we need to remove
 stop_worker_p check from worker, here:

 --- a/mjit_worker.c
 +++ b/mjit_worker.c
 @@ -1182,7 +1182,7 @@ copy_cache_from_main_thread(struct mjit_copy_job *job)
 return FALSE;

 CRITICAL_SECTION_START(3, "in MJIT copy job wait");
 -    while (!job->finish_p && !stop_worker_p) {
 +    while (!job->finish_p) {
 rb_native_cond_wait(&mjit_worker_wakeup, &mjit_engine_mutex);
 verbose(3, "Getting wakeup from client");
 }

 Otherwise, the Ruby thread will SEGV because mjit_copy_job goes
 out-of-scope in mjit_worker.

 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(rb_threadptr_execute_interrupts+0x81) [0x1218a
61] thread.c:2134
 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(stop_worker+0xcb) [0x1134bdb] ./vm_core.h:1773
 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(mjit_finish) mjit.c:726
 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(ruby_finalize_1+0x0) [0x10d829b] eval.c:236
 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(ruby_cleanup) eval.c:238
 > /usr/home/hsbt/chkbuild/tmp/build/20181022T153002Z/ruby/ruby(main+0x72) [0x104be92] ./main.c:42
 > ~~~
 >
 > https://rubyci.org/logs/rubyci.s3.amazonaws.com/freebsd11zfs/ruby-trunk/log/20181022T153002Z.fail.html.gz
 >
 > I think `rb_threadptr_execute_interrupts()` is designed to be called in safe places on VM and calling it fr
om `ruby_cleanup()` may not be expected. Could you remove `RUBY_VM_CHECK_INTS()` from the `stop_worker()` and
try to solve the original issue in another way?

 Not sure if removing interrupt checking here is right, either.
 We want to ensure Ctrl-C works during shutdown.
```

#75 - 10/22/2018 11:44 PM - k0kubun (Takashi Kokubun)

- Status changed from Assigned to Closed

Thanks for your quick response.

> So rb_postponed_job_flush is calling mjit_copy_job_handler? I'm
> not sure how we're entering the parser from
> rb_postponed_job_flush, actually.

Sometimes (1) mjit_copy_job_handler could be also called and it causes a problem. But in such a case, at lest memcpy appears on the SEGV, and in this stack trace memcpy is not called but (2) tokadd_string() which seems not called inside mjit_copy_job_handler is called.

So, if the problem is only (1), your patch

```
-    while (!job->finish_p && !stop_worker_p) {
+    while (!job->finish_p) {
```

could work, but at the same time this patch is an invert of r65299 r65300 r65301. As stop_worker blocks until MJIT worker is stopped and MJIT worker blocks until the postponed job is flushed with your patch (because you skip to stopping on stop_worker_p), that results in dead lock.

So, for (1), I'll check stop_worker_p inside mjit_copy_job_handler and skip everything if it's being stopped. I think that wouldn't have a race condition as long as RUBY_VM_CHECK_INTS is called only when stop_worker_p is already set to TRUE in ruby_cleanup situation.

> Not sure if removing interrupt checking here is right, either.
> We want to ensure Ctrl-C works during shutdown.

I understood the intention of the code. I want to support that too. Since (2) could be just a side effect of (1) failure (I'm not sure), let me try fixing (1) first, and let's see whether (2) is fixed by that or not. Thanks.

#76 - 10/23/2018 12:10 AM - k0kubun (Takashi Kokubun)

Ah, one thing I forgot to note which was found on watching CI is that looping in stop_worker() (calling rb_postponed_job_flush) may not flush mjit_copy_job_handler when EC_EXEC_TAG() is not TAG_NONE. That's why I think stop_worker_p should be checked inside copy_cache_from_main_thread().

My current attempt to fix this issue is r65312. I'll watch CI results.

#77 - 10/23/2018 01:22 AM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> Ah, one thing I forgot to note which was found on watching CI is that looping in stop_worker() (calling rb_postponed_job_flush) may not flush mjit_copy_job_handler when EC_EXEC_TAG() is not TAG_NONE. That's why I think stop_worker_p should be checked inside copy_cache_from_main_thread().
>
> My current attempt to fix this issue is r65312. I'll watch CI results.

OK, also, I don't think rb_postponed_job_register is safe to
call without GVL because vm->postponed_jobs needs protecting.

#78 - 10/23/2018 02:28 AM - k0kubun (Takashi Kokubun)

The original issue which I mentioned seems to have been fixed by r65312.

> I don't think rb_postponed_job_register is safe to call without GVL because vm->postponed_jobs needs protecting.

I thought the 100% same thing before implementing MJIT copy job, but ko1 (Koichi Sasada) said it's safe and so it's made as such. I'll confirm about that with him again.

#79 - 10/23/2018 02:59 AM - ko1 (Koichi Sasada)

"multi-thread-safe" is not correct. It was my mistake and it should be "signal handler safe" because this API is used by stackprof (used with SIGPROF).

however, I recognize this assumption is broken with the following ticket.

```
commit 5a1dfb04bc2b09fcf8f3427cac72d0ce52a45eb2
Author: normal <normal@b2dd03c8-39d4-4d8f-98ff-823fe69b080e>
Date:   Thu May 17 04:20:33 2018 +0000

    vm_trace: implement postponed_jobs as st_table

    st_table allows the use of st_shift to act as an order-preserving
    queue while allowing fast lookups to prevent duplicate jobs.

    In typical Ruby apps, this table will only have one entry
```

```
    for gc_finalize_deferred_register.
```

I believe this commit should be reverted (and I need to write a comment about assumption).

#80 - 10/23/2018 03:52 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> Issue #14867 has been updated by ko1 (Koichi Sasada).
>
> "multi-thread-safe" is not correct. It was my mistake and it should be "signal handler safe" because this API is used by stackprof (used with SIGPROF).
>
> however, I recognize this assumption is broken with the following ticket.
>
> ```
> commit 5a1dfb04bc2b09fcf8f3427cac72d0ce52a45eb2
> ```

Oops :x nobody told me about stackprof and SIGPROF usage :x

> ```
> vm_trace: implement postponed_jobs as st_table
> ```
>
> I believe this commit should be reverted (and I need to write a comment about assumption).

Reverted and commented in r65316

#81 - 10/23/2018 04:54 AM - ko1 (Koichi Sasada)

Thank you for your apply.

> Oops :x nobody told me about stackprof and SIGPROF usage :x

sorry I needed to write documents.
(I thought nobody touch there :p)

#82 - 10/27/2018 05:10 AM - k0kubun (Takashi Kokubun)

- Status changed from Closed to Assigned

Recently we see 2 types of deadlocks on CI with --jit or --jit-wait.

1. waitpid on #system, #`, or Process.waitpid

http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1431775
http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1431394
http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1430875

The stack trace on deadlock is like this:

```
0x00007fbd6585ecf6 in __GI_ppoll (fds=fds@entry=0x7ffd30485118, nfds=nfds@entry=1, timeout=<optimized out>, si
gmask=sigmask@entry=0x0) at ../sysdeps/unix/sysv/linux/ppoll.c:39
>>> Machine level backtrace

Thread 3 (Thread 0x7fbd66c90700 (LWP 30964)):
#0  0x00007fbd667af384 in __libc_read (fd=6, buf=0x7fbd58047c70, nbytes=1024) at ../sysdeps/unix/sysv/linux/re
ad.c:27
#1  0x000055ac93a5c433 in rb_thread_io_blocking_region (func=func@entry=0x55ac93925110 <internal_read_func>, d
ata1=data1@entry=0x7fbd66c8f4c0, fd=<optimized out>) at /home/ko1/ruby/src/trunk-mjit-wait/thread.c:1529
#2  0x000055ac93924ebe in rb_read_internal (count=<optimized out>, buf=<optimized out>, fd=<optimized out>) at
 /home/ko1/ruby/src/trunk-mjit-wait/io.c:987
#3  read_internal_call (arg=arg@entry=140451449992880) at /home/ko1/ruby/src/trunk-mjit-wait/io.c:2732
#4  0x000055ac938fa0d3 in rb_ensure (b_proc=0x55ac93924e80 <read_internal_call>, data1=140451449992880, e_proc
=e_proc@entry=0x55ac93a286a0 <rb_str_unlocktmp>, data2=<optimized out>) at /home/ko1/ruby/src/trunk-mjit-wait/
eval.c:1052
#5  0x000055ac93a3c490 in rb_str_locktmp_ensure (str=<optimized out>, func=func@entry=0x55ac93924e80 <read_int
ernal_call>, arg=arg@entry=140451449992880) at /home/ko1/ruby/src/trunk-mjit-wait/string.c:2654
#6  0x000055ac9393b99b in io_getpartial (argc=<optimized out>, argv=<optimized out>, io=<optimized out>, opts=
opts@entry=8, nonblock=nonblock@entry=0) at /home/ko1/ruby/src/trunk-mjit-wait/io.c:2782
#7  0x000055ac9393bf51 in io_readpartial (argc=<optimized out>, argv=<optimized out>, io=<optimized out>) at /
home/ko1/ruby/src/trunk-mjit-wait/io.c:2872
#8  0x000055ac93a90455 in vm_call_cfunc_with_frame (ci=0x55ac95418c40, cc=<optimized out>, calling=<optimized
out>, reg_cfp=0x7fbd5f34bfa0, ec=0x55ac97772258) at /home/ko1/ruby/src/trunk-mjit-wait/vm_insnhelper.c:1914
```

```
... (snip) ...
#16 rb_vm_invoke_proc (ec=<optimized out>, proc=proc@entry=0x55ac99d5f2e0, argc=<optimized out>, argv=<optimiz
ed out>, passed_block_handler=passed_block_handler@entry=0) at /home/ko1/ruby/src/trunk-mjit-wait/vm.c:1191
#17 0x000055ac93a59615 in thread_do_start (args=94199785452720, th=0x55ac97a8d040) at /home/ko1/ruby/src/trunk
-mjit-wait/thread.c:667
#18 thread_start_func_2 (th=th@entry=0x55ac97a8d040, stack_start=<optimized out>) at /home/ko1/ruby/src/trunk-
mjit-wait/thread.c:706
#19 0x000055ac93a59acb in thread_start_func_1 (th_ptr=<optimized out>) at /home/ko1/ruby/src/trunk-mjit-wait/t
hread_pthread.c:1017
#20 0x00007fbd667a56db in start_thread (arg=0x7fbd66c90700) at pthread_create.c:463
#21 0x00007fbd6586b88f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 2 (Thread 0x7fbd65749700 (LWP 30944)):
#0  0x00007fbd667ab9f3 in futex_wait_cancelable (private=<optimized out>, expected=0, futex_word=0x55ac93e3942
8 <mjit_worker_wakeup+40>) at ../sysdeps/unix/sysv/linux/futex-internal.h:88
#1  __pthread_cond_wait_common (abstime=0x0, mutex=0x55ac93e394c0 <mjit_engine_mutex>, cond=0x55ac93e39400 <mj
it_worker_wakeup>) at pthread_cond_wait.c:502
#2  __pthread_cond_wait (cond=cond@entry=0x55ac93e39400 <mjit_worker_wakeup>, mutex=mutex@entry=0x55ac93e394c0
 <mjit_engine_mutex>) at pthread_cond_wait.c:655
#3  0x000055ac93a566a9 in rb_native_cond_wait (cond=cond@entry=0x55ac93e39400 <mjit_worker_wakeup>, mutex=mute
x@entry=0x55ac93e394c0 <mjit_engine_mutex>) at /home/ko1/ruby/src/trunk-mjit-wait/thread_pthread.c:496
#4  0x000055ac9395cc19 in copy_cache_from_main_thread (job=0x7fbd65748e60) at /home/ko1/ruby/src/trunk-mjit-wa
it/mjit_worker.c:1189
#5  mjit_worker () at /home/ko1/ruby/src/trunk-mjit-wait/mjit_worker.c:1247
#6  0x000055ac93a51d0a in mjit_worker (arg=0x55ac9395c770 <mjit_worker>) at /home/ko1/ruby/src/trunk-mjit-wait
/thread_pthread.c:1866
#7  0x00007fbd667a56db in start_thread (arg=0x7fbd65749700) at pthread_create.c:463
#8  0x00007fbd6586b88f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 1 (Thread 0x7fbd66df3740 (LWP 30930)):
#0  0x00007fbd6585ecf6 in __GI_ppoll (fds=fds@entry=0x7ffd30485118, nfds=nfds@entry=1, timeout=<optimized out>
, sigmask=sigmask@entry=0x0) at ../sysdeps/unix/sysv/linux/ppoll.c:39
#1  0x000055ac93a58f5d in native_ppoll_sleep (th=th@entry=0x55ac952bde60, rel=rel@entry=0x0) at /home/ko1/ruby
/src/trunk-mjit-wait/thread_pthread.c:2015
#2  0x000055ac93a5b22a in native_sleep (th=th@entry=0x55ac952bde60, rel=0x0) at /home/ko1/ruby/src/trunk-mjit-
wait/thread_pthread.c:2051
#3  0x000055ac93a5baeb in rb_thread_sleep_interruptible () at /home/ko1/ruby/src/trunk-mjit-wait/thread.c:1270
#4  0x000055ac939bb4b5 in waitpid_sleep (x=x@entry=140725413499744) at /home/ko1/ruby/src/trunk-mjit-wait/proc
ess.c:1099
#5  0x000055ac938fa0d3 in rb_ensure (b_proc=b_proc@entry=0x55ac939bb4a0 <waitpid_sleep>, data1=data1@entry=140
725413499744, e_proc=e_proc@entry=0x55ac939bb450 <waitpid_cleanup>, data2=data2@entry=140725413499744) at /hom
e/ko1/ruby/src/trunk-mjit-wait/eval.c:1052
#6  0x000055ac939c2cf9 in waitpid_wait (w=0x7ffd30485360) at /home/ko1/ruby/src/trunk-mjit-wait/process.c:1158
#7  rb_waitpid (pid=<optimized out>, st=0x7ffd304853c4, flags=<optimized out>) at /home/ko1/ruby/src/trunk-mji
t-wait/process.c:1197
#8  0x000055ac939c4e6e in rb_f_system (argc=3, argv=0x7fbd66cf2338) at /home/ko1/ruby/src/trunk-mjit-wait/proc
ess.c:4371
#9  0x000055ac93a90455 in vm_call_cfunc_with_frame (ci=0x55ac98d41800, cc=<optimized out>, calling=<optimized
 out>, reg_cfp=0x7fbd66df1cc8, ec=0x55ac952be2e8) at /home/ko1/ruby/src/trunk-mjit-wait/vm_insnhelper.c:1914
... (snip) ...
#35 0x000055ac938f90fb in ruby_exec_node (n=<optimized out>) at /home/ko1/ruby/src/trunk-mjit-wait/eval.c:325
#36 ruby_run_node (n=<optimized out>) at /home/ko1/ruby/src/trunk-mjit-wait/eval.c:317
#37 0x000055ac938f41df in main (argc=<optimized out>, argv=<optimized out>) at /home/ko1/ruby/src/trunk-mjit-w
ait/main.c:42
>>> Dump Ruby level backtrace
th: 0x55ac952bde60, native_id: 0x7fbd66df3740
-- Control frame information -----------------------------------------------
c:0016 p:---- s:0107 e:000106 CFUNC  :system
c:0015 p:0024 s:0100 e:000099 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/fileutils/test_fileutils.rb:148
c:0014 p:0098 s:0095 e:000094 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/fileutils/test_fileutils.rb:167
c:0013 p:0044 s:0090 e:000089 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/lib/minitest/unit.rb:1289
c:0012 p:0015 s:0081 e:000080 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/lib/test/unit/testcase.rb:18
c:0011 p:0082 s:0076 e:000075 BLOCK  /home/ko1/ruby/src/trunk-mjit-wait/test/lib/minitest/unit.rb:952 [FINISH]
c:0010 p:---- s:0069 e:000068 CFUNC  :map
c:0009 p:0133 s:0065 e:000064 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/lib/minitest/unit.rb:945
c:0008 p:0045 s:0053 e:000052 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/lib/test/unit.rb:1044
c:0007 p:0095 s:0046 E:0020b0 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/lib/test/unit/parallel.rb:50
c:0006 p:0009 s:0030 e:000029 BLOCK  /home/ko1/ruby/src/trunk-mjit-wait/test/lib/test/unit/parallel.rb:25 [FIN
ISH]
c:0005 p:---- s:0026 e:000025 CFUNC  :map
c:0004 p:0006 s:0022 e:000021 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/lib/test/unit/parallel.rb:24
c:0003 p:0263 s:0016 E:001aa0 METHOD /home/ko1/ruby/src/trunk-mjit-wait/test/lib/test/unit/parallel.rb:122
c:0002 p:0128 s:0006 E:0021f0 EVAL   /home/ko1/ruby/src/trunk-mjit-wait/test/lib/test/unit/parallel.rb:207 [FI
NISH]
c:0001 p:0000 s:0003 E:0021f0 (none) [FINISH]
```

```
th: 0x55ac97a8d040, native_id: 0x7fbd66c90700
-- Control frame information -----------------------------------------------
c:0003 p:---- s:0012 e:000011 CFUNC  :readpartial
c:0002 p:0036 s:0007 e:000006 BLOCK  /home/ko1/ruby/src/trunk-mjit-wait/test/lib/test/unit/parallel.rb:39 [FIN
ISH]
c:0001 p:---- s:0003 e:000002 (none) [FINISH]
```

In this case, 3 threads are blocking in:

1. rb_thread_io_blocking_region called from rb_read_internal called from io_readpartial
2. native_ppoll_sleep called inside rb_waitpid
3. (MJIT worker) rb_native_cond_wait called from copy_cache_from_main_thread

I think 3's lock is completely independent from blocking in 1 and 2, and I have no idea why 1 and 2 are blocking in that place forever.

2. in ruby_cleanup

http://ci.rvm.jp/results/trunk-mjit@silicon-docker/1428895
http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1429135

The stack trace on deadlock is like this:

```
0x00005587e954031c in verbose (level=3, format=format@entry=0x5587e973f692 "Unlocked %s", level=3) at /home/ko
1/ruby/src/trunk-mjit/mjit_worker.c:289
289 verbose(int level, const char *format, ...)
>>> Machine level backtrace

Thread 3 (Thread 0x7fd7f439d700 (LWP 11123)):
#0  0x00007fd7f3e57ed9 in futex_reltimed_wait_cancelable (private=<optimized out>, reltime=0x7fd7f439cd70, exp
ected=0, futex_word=0x7fd7f439ce88) at ../sysdeps/unix/sysv/linux/futex-internal.h:142
#1  __pthread_cond_wait_common (abstime=0x7fd7f439ce50, mutex=0x5587e9a1f660 <thread_cache_lock>, cond=0x7fd7f
439ce60) at pthread_cond_wait.c:533
#2  __pthread_cond_timedwait (cond=cond@entry=0x7fd7f439ce60, mutex=mutex@entry=0x5587e9a1f660 <thread_cache_l
ock>, abstime=abstime@entry=0x7fd7f439ce50) at pthread_cond_wait.c:667
#3  0x00005587e963eca8 in native_cond_timedwait (abs=<synthetic pointer>, mutex=0x5587e9a1f660 <thread_cache_l
ock>, cond=0x7fd7f439ce60) at /home/ko1/ruby/src/trunk-mjit/thread_pthread.c:516
#4  register_cached_thread_and_wait (altstack=0x7fd7e0000b20) at /home/ko1/ruby/src/trunk-mjit/thread_pthread.
c:1079
#5  thread_start_func_1 (th_ptr=<optimized out>) at /home/ko1/ruby/src/trunk-mjit/thread_pthread.c:1024
#6  0x00007fd7f3e516db in start_thread (arg=0x7fd7f439d700) at pthread_create.c:463
#7  0x00007fd7f2f1788f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 2 (Thread 0x7fd7f2df5700 (LWP 17017)):
#0  0x00007fd7f2f0acf6 in __GI_ppoll (fds=fds@entry=0x7fd7f2def8e8, nfds=nfds@entry=1, timeout=<optimized out>
, sigmask=sigmask@entry=0x0) at ../sysdeps/unix/sysv/linux/ppoll.c:39
#1  0x00005587e963c757 in rb_sigwait_sleep (th=th@entry=0x0, sigwait_fd=sigwait_fd@entry=3, rel=rel@entry=0x0)
 at /home/ko1/ruby/src/trunk-mjit/thread_pthread.c:1959
#2  0x00005587e95a7a7c in ruby_waitpid_locked (vm=vm@entry=0x5587ea98a950, pid=pid@entry=11124, status=status@
entry=0x7fd7f2def9dc, options=options@entry=0, cond=cond@entry=0x7fd7f2def9e0) at /home/ko1/ruby/src/trunk-mji
t/process.c:1070
#3  0x00005587e953fd02 in exec_process (path=<optimized out>, argv=argv@entry=0x7fd7ec01ced0) at /home/ko1/rub
y/src/trunk-mjit/mjit_worker.c:657
#4  0x00005587e9541276 in compile_c_to_o (o_file=0x7fd7f2defa80 "/tmp/_ruby_mjit_p17016u76.o", c_file=0x7fd7f2
defb70 "/tmp/_ruby_mjit_p17016u76.c") at /home/ko1/ruby/src/trunk-mjit/mjit_worker.c:842
#5  convert_unit_to_func (unit=0x5587eaf601f0, cc_entries=<optimized out>, is_entries=<optimized out>) at /hom
e/ko1/ruby/src/trunk-mjit/mjit_worker.c:1122
#6  0x00005587e9541af6 in mjit_worker () at /home/ko1/ruby/src/trunk-mjit/mjit_worker.c:1253
#7  0x00005587e9636dfa in mjit_worker (arg=0x5587e9541830 <mjit_worker>) at /home/ko1/ruby/src/trunk-mjit/thre
ad_pthread.c:1866
#8  0x00007fd7f3e516db in start_thread (arg=0x7fd7f2df5700) at pthread_create.c:463
#9  0x00007fd7f2f1788f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 1 (Thread 0x7fd7f449f740 (LWP 17016)):
#0  0x00005587e954031c in verbose (level=3, format=format@entry=0x5587e973f692 "Unlocked %s", level=3) at /hom
e/ko1/ruby/src/trunk-mjit/mjit_worker.c:289
#1  0x00005587e95437a3 in CRITICAL_SECTION_FINISH (msg=0x5587e973f8e8 "in stop_worker", level=3) at /home/ko1/
ruby/src/trunk-mjit/mjit_worker.c:448
#2  stop_worker () at /home/ko1/ruby/src/trunk-mjit/mjit.c:659
#3  mjit_finish () at /home/ko1/ruby/src/trunk-mjit/mjit.c:734
#4  0x00005587e94dde81 in ruby_cleanup (ex=<optimized out>) at /home/ko1/ruby/src/trunk-mjit/eval.c:236
#5  0x00005587e94de102 in ruby_run_node (n=<optimized out>) at /home/ko1/ruby/src/trunk-mjit/eval.c:317
#6  0x00005587e94d91df in main (argc=<optimized out>, argv=<optimized out>) at /home/ko1/ruby/src/trunk-mjit/m
ain.c:42
>>> Dump Ruby level backtrace
```

```
th: 0x5587ea98ae60, native_id: 0x7fd7f449f740
-- Control frame information -----------------------------------------------
c:0001 p:0000 s:0003 E:002360 (none) [FINISH]
```

In this case, 3 threads are blocking in:

1. native_cond_timedwait called from register_cached_thread_and_wait
2. (MJIT worker) rb_sigwait_sleep called from ruby_waitpid_locked called from compile_c_to_o
3. (main thread) looping inside stop_worker called from ruby_cleanup

1 looks innocent and ignoreable. In 2, somehow it seems to have lost the process to wait, or locked with VM's lock. If the situation is the former, sometimes this CI machine is overloaded and thus it may happen on such an environment. And if the situation is the latter, I have no idea why it's locked.

normalperson (Eric Wong) While I'm taking a look at this by myself, it would be helpful if you have any insight about this, since waitpid thing is involved in both issues. For now I have no approach to fix this because it's not reproductive on my local machine...

#83 - 10/27/2018 08:52 AM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> In this case, 3 threads are blocking in:
>
> 1. rb_thread_io_blocking_region called from rb_read_internal called from io_readpartial
> 2. native_ppoll_sleep called inside rb_waitpid
> 3. (MJIT worker) rb_native_cond_wait called from copy_cache_from_main_thread

rb_postponed_job_register only sets a flag, but doesn't wake up
sleeping the thread in 1. or 2. by calling ubf.func (via
rb_threadptr_interrupt).

This is a tricky situation...

Calling ubf.func is NOT async-signal-safe, so rb_postponed_job_register
may not use it by default, either.

Also ruby_current_execution_context_ptr variable is unstable
between setting ec->interrupt_flag (via
RUBY_VM_SET_POSTPONED_JOB_INTERRUPT) and ubf.func calls since we
make them without GVL

This is a similar situation to [Bug #14939] r64062

> I think 3's lock is completely independent from blocking in 1
> and 2, and I have no idea why 1 and 2 are blocking in that
> place forever.

I'm not sure if rb_postponed_job_register is the right tool
in a multi-threaded situation. It seems like the "postponed"
part is a bad fit for MJIT anyways.

Anyways, the above issue is pretty straightforward, I think.

2. in ruby_cleanup

> In this case, 3 threads are blocking in:

Not sure about this one, yet:

> 1. native_cond_timedwait called from register_cached_thread_and_wait
> 2. (MJIT worker) rb_sigwait_sleep called from ruby_waitpid_locked called from compile_c_to_o
> 3. (main thread) looping inside stop_worker called from ruby_cleanup
>
> 1 looks innocent and ignoreable.

Not sure, this is a timeout situation?

THREAD_CACHE_TIME is only 3 seconds, so I think the cache entry
would've timed out if a whole test hits timeout.

> In 2, somehow it seems to have lost the process to wait, or

> locked with VM's lock. If the situation is the former,
> sometimes this CI machine is overloaded and thus it may happen
> on such an environment. And if the situation is the latter, I
> have no idea why it's locked.

Can you tell if the process 2. is waiting on is still a zombie?
To debug, maybe always return &busy_wait from sigwait_sleep_time
and check the contents of vm->waiting_pids periodically.

You may also periodically kill(pid, 0) to see if the process is
killable.

#84 - 10/27/2018 11:55 AM - k0kubun (Takashi Kokubun)

- Assignee changed from normalperson (Eric Wong) to k0kubun (Takashi Kokubun)

Thank you for your quick and detailed reply. For 1, I'll try to acquire GVL from MJIT worker thread to resolve this issue. In that case possibly we could just synchronously finish the copy job during the lock and avoid using postponed_job.

I'll take a look at 2 after that, but your comment made sense.

#85 - 10/27/2018 09:43 PM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> Thank you for your quick and detailed reply. For 1, I'll try
> to acquire GVL from MJIT worker thread to resolve this issue.
> In that case possibly we could just synchronously finish the
> copy job during the lock and avoid using postponed_job.

Acquire GVL from non-Ruby Thread doesn't seem possible, but
making MJIT thread into a Ruby Thread may simplify the code,
actually.

However, I don't understand why copy_cache_from_main_thread
needs to be done at all. I haven't looked at it in detail, but
it adds extra synchronization overhead for a slightly more
up-to-date cache, right?

Back-and-forth synchronization between threads kills parallelism
and likely defeats the purpose of MJIT worker having its own
thread.

So I think two options are worth experimenting with:

1. Copy cache when enqueueing during mjit_add_iseq_to_process,
   give up on up-to-dateness of IC/CC. This should be easy
   and simplify existing code, even.

2. Get rid of MJIT worker thread and rely on SIGCHLD + non-blocking waitpid.
   I don't know the portability problems in Windows, though.

#86 - 10/28/2018 12:32 AM - k0kubun (Takashi Kokubun)

> I haven't looked at it in detail, but it adds extra synchronization overhead for a slightly more up-to-date cache, right?

Correct.

> Back-and-forth synchronization between threads kills parallelism and likely defeats the purpose of MJIT worker having its own thread.

This part may not be correct. The introduction of synchronization did not spoil the benchmark result of Optcarrot. I assume that this is because RUBY_VM_CHECK_INTS calls are frequent enough (MJIT worker is not waiting for a long time), copy job (2 memcpy calls) is fast enough, and the time taken for compilation by C compiler is much longer than the time until being enqueued so that necessary caches are filled until being JIT-ed.

1. Copy cache when enqueueing during mjit_add_iseq_to_process, give up on up-to-dateness of IC/CC. This should be easy and simplify existing code, even.

I agree that it's worth experimenting with, and actually I tried the approach first. Unfortunately it didn't work well at least for Optcarrot because its hotspot requires a lot of calls to fill inline caches to reproduce the current benchmark result. Requiring 100,000 calls to enqueue doesn't work for Optcarrot because it renders a lot of pixels even in the first frame. We're not sure if the workload is so common, but having the latest cache is desired if possible.

But still this would be the simplest approach for this issue.

1. Get rid of MJIT worker thread and rely on SIGCHLD + non-blocking waitpid. I don't know the portability problems in Windows, though.

It sounds interesting and it could be reasonable since the major time of JIT is taken for waiting for C compiler process. But we already achieved a complete support of mswin MJIT and I'm reluctant to reduce the portability.

#87 - 10/28/2018 01:32 AM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> 1. Get rid of MJIT worker thread and rely on SIGCHLD + non-blocking waitpid. I don't know the portability problems in Windows, though.
>
> It sounds interesting and it could be reasonable since the major time of JIT is taken for waiting for C compiler process. But we already achieved a complete support of mswin MJIT and I'm reluctant to reduce the portability.

I think we'll have to support non-blocking/event-based waitpid
in Windows for auto-fiber/Thread::Light, anyways. So making
MJIT worker thread be a Ruby Thread (or Thread::Light) would
allow it to hook into process.c APIs. I don't think the
existing code is too far off from being able to support
Windows...

#88 - 10/28/2018 09:02 AM - k0kubun (Takashi Kokubun)

> I think we'll have to support non-blocking/event-based waitpid in Windows for auto-fiber/Thread::Light, anyways. So making MJIT worker thread be a Ruby Thread (or Thread::Light) would allow it to hook into process.c APIs. I don't think the existing code is too far off from being able to support Windows...

I see. Dropping MJIT worker by non-blocking/event-based waitpid would be an option once it's supported for auto-fiber/Thread::Light. At this moment, I'm not sure which is larger, synchronization cost between threads or translating ISeq to C code synchronously. So in a very short term, I wanna fix the 1 (waitpid) case with postponed_job unless it has a critical blocker to fix the issue.

Note that the current synchronization design is already achieving the better benchmark result than one by copying inline caches on enqueueing an ISeq. To simplify the situation by stop using postponed_job, I experimented to have another global variable for the synchronization job which is to be checked and invoked by RUBY_VM_CHECK_INTS, but the benchmark result was not good. So for now using postponed_job is the only possible approach that achieves the current performance.

> rb_postponed_job_register only sets a flag, but doesn't wake up sleeping the thread in 1. or 2. by calling ubf.func (via rb_threadptr_interrupt).
>
> This is a tricky situation...
>
> Calling ubf.func is NOT async-signal-safe, so rb_postponed_job_register may not use it by default, either.

I'm not still understanding why this could be related to the deadlock 1 (waitpid on #system, #`, or Process.waitpid). I assume that the sleeping threads "1." and "2." should be waken up by something prepared in threads "1." and "2.", not by MJIT worker thread "3.". So the fact that rb_postponed_job_register (called by MJIT worker thread "3.") may not wakeup the thread "1." or "2." should not be the cause of this deadlock, in my understanding.

I actually don't understand how threads "1." and "2." are usually waken up. For waitpid thread "2.", I'm guessing that a signal handler (of SIGCHLD?) registered by the thread "2." itself somehow *replaces* native_ppoll_sleep and finishes it, but not sure.

> Also ruby_current_execution_context_ptr variable is unstable between setting ec->interrupt_flag (via RUBY_VM_SET_POSTPONED_JOB_INTERRUPT) and ubf.func calls since we make them without GVL
>
> This is a similar situation to [Bug #14939] r64062

I recognize this issue. For this issue, ko1 (Koichi Sasada) suggested to repeatedly calling rb_postponed_job_register_one (that registers the same job at most only once) to survive the ec switches. I can fix this issue, but I'm NOT thinking that this could be the solution for this deadlock if unblocking thread "3." waiting for postponed job may not wake up threads "1." and "2." as I understand.

I *assume* that threads "1." and "2." are not waken up because something that should wake up "1." and "2." is not working somehow with MJIT enabled. At first ko1 (Koichi Sasada) guessed that rb_postponed_job_register swaps ec->interrupt_flag in a bad way and a flag for the signal handler for waitpid is skipped to be set, but actually we're always using ATOMIC_OR for modifying ec->interrupt_flag, and so it may not be the cause either.

#89 - 10/29/2018 01:23 AM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> Issue #14867 has been updated by k0kubun (Takashi Kokubun).
>
> > I think we'll have to support non-blocking/event-based waitpid in Windows for auto-fiber/Thread::Light, anyways. So making MJIT worker thread be a Ruby Thread (or Thread::Light) would allow it to hook into process.c APIs. I don't think the existing code is too far off from being able to support Windows...
>
> I see. Dropping MJIT worker by non-blocking/event-based waitpid would be an option once it's supported for auto-fiber/Thread::Light. At this moment, I'm not sure which is larger, synchronization cost between threads or translating ISeq to C code synchronously. So in a very short term, I wanna fix the 1 (waitpid) case with postponed_job unless it has a critical blocker to fix the issue.

I suppose MJIT has priority over Thread::Light at the moment
since MJIT is already in trunk and people know about it.
I can take some time to investigate moving MJIT over to
event-based waitpid in a platform-independent way.

The other thing is MJIT multi-threading/synchronization seems
tricky-to-debug right now, and making it event-based should
be beneficial for reliability.

> Note that the current synchronization design is already achieving the better benchmark result than one by copying inline caches on enqueueing an ISeq. To simplify the situation by stop using postponed_job, I experimented to have another global variable for the synchronization job which is to be checked and invoked by RUBY_VM_CHECK_INTS, but the benchmark result was not good. So for now using postponed_job is the only possible approach that achieves the current performance.

OK, thank you for that information. So checking one extra global
variables with RUBY_VM_CHECK_INTS introduces a measurable perf
hit?

> > rb_postponed_job_register only sets a flag, but doesn't wake up sleeping the thread in 1. or 2. by calling ubf.func (via rb_threadptr_interrupt).
> >
> > This is a tricky situation...
> >
> > Calling ubf.func is NOT async-signal-safe, so rb_postponed_job_register may not use it by default, either.
>
> I'm not still understanding why this could be related to the deadlock 1 (waitpid on #system, #`, or Process.waitpid). I assume that the sleeping threads "1." and "2." should be waken up by something prepared in threads "1." and "2.", not by MJIT worker thread "3.". So the fact that rb_postponed_job_register (called by MJIT worker thread "3.") may not wakeup the thread "1." or "2." should not be the cause of this deadlock, in my understanding.

Ah, so the waitpid from #system is on /bin/rm (I missed that earlier)
Is `rm' stuck (NFS, drive error, etc)? But yes, I also wonder
if there's a bug in the current system+waitpid implementation.

Perhaps #system needs to use similar mechanism as mjit_worker to
prevent work-stealing.

Regardless, ubf.func need to be called after registering postponed
job, so there was still a problem, there.

> I actually don't understand how threads "1." and "2." are usually waken up. For waitpid thread "2.", I'm guessing that a signal handler (of SIGCHLD?) registered by the thread "2." itself somehow *replaces* native_ppoll_sleep and finishes it, but not sure.

System-level signal handler is only registered at startup, and Ruby Signal.trap handler is process-wide and only runs in main thread. native_ppoll_sleep can be called by any thread (which exclusively acquires sigwait_fd).

> > Also ruby_current_execution_context_ptr variable is unstable between setting ec->interrupt_flag (via RUBY_VM_SET_POSTPONED_JOB_INTERRUPT) and ubf.func calls since we make them without GVL
> >
> > This is a similar situation to [Bug #14939] r64062

> I recognize this issue. For this issue, ko1 (Koichi Sasada) suggested to repeatedly calling rb_postponed_job_register_one (that registers the same job at most only once) to survive the ec switches. I can fix this issue, but I'm NOT thinking that this could be the solution for this deadlock if unblocking thread "3." waiting for postponed job may not wake up threads "1." and "2." as I understand.

Right.

> I *assume* that threads "1." and "2." are not waken up because something that should wake up "1." and "2." is not working somehow with MJIT enabled. At first ko1 (Koichi Sasada) guessed that rb_postponed_job_register swaps ec->interrupt_flag in a bad way and a flag for the signal handler for waitpid is skipped to be set, but actually we're always using ATOMIC_OR for modifying ec->interrupt_flag, and so it may not be the cause either.

Agreed, and the self-pipe/eventfd would've woken native_ppoll_sleep, anyways.

(But I need to fix a car problem, first :<)

#90 - 10/29/2018 05:48 AM - k0kubun (Takashi Kokubun)

> The other thing is MJIT multi-threading/synchronization seems tricky-to-debug right now, and making it event-based should be beneficial for reliability.

Yeah, if the performance regression by moving to event-based implementation is not so big, using that would be reasonable. I want MJIT implementation to be reliable too.

> So checking one extra global variables with RUBY_VM_CHECK_INTS introduces a measurable perf hit?

Correct. With Optcarrot on my machine, the trunk --jit is 87fps at maximum, extra global variable check on all RUBY_VM_CHECK_INTS was 83fps, and disabling the check on JIT-ed code (only VM checks it) was 84fps. Copying inline caches on enqueueing was 80~84fps-ish.

> Ah, so the waitpid from #system is on /bin/rm (I missed that earlier)

For http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1431394, yes. Others http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1431775 http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1430875 are on waitpid for different things.

> Is `rm' stuck (NFS, drive error, etc)?

Unfortunately I can't know that now. I'll try to check that once I see another failure notification while I'm online. But at least I can say that it looks not stucking on similar but non-MJIT CIs.

> But yes, I also wonder if there's a bug in the current system+waitpid implementation.

> Perhaps #system needs to use similar mechanism as mjit_worker to prevent work-stealing.
>
> the self-pipe/eventfd would've woken native_ppoll_sleep, anyways.

I see. So that would be the direct fix for this deadlock.

> System-level signal handler is only registered at startup, and Ruby Signal.trap handler is process-wide and only runs in main thread. native_ppoll_sleep can be called by any thread (which exclusively acquires sigwait_fd).

Thanks for the explanation around that. That helps my understanding.

> (But I need to fix a car problem, first :<)

Oh, sorry to hear that :<

#91 - 10/29/2018 03:22 PM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> > Ah, so the waitpid from #system is on /bin/rm (I missed that earlier)
>
> For http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1431394, yes. Others http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1431775 http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1430875 are on waitpid for different things.

OK, first patch for #system is here:

https://80x24.org/spew/20181029151106.9474-1-e@80x24.org/raw

It took me a few iterations because of error handling, so I
shoved the lock as deep into where fork/vfork is actually called
to avoid deadlocks and stalls on FIFOs.

I plan to build on this for MJIT...

Greg: everything should be confined to *nix for now, but can you
make sure I didn't break anything on Windows? Thanks.

#92 - 10/29/2018 04:07 PM - k0kubun (Takashi Kokubun)

- Assignee changed from k0kubun (Takashi Kokubun) to normalperson (Eric Wong)

Today I took a deeper look at how rb_waitpid is currently working. After reading that, while I couldn't exactly figure out which part is wrong, my bets are:

- There's a race condition around some of SIGCHLD by gcc/clang process spawn by MJIT, rb_waitpid, signal handler for SIGCHLD, RUBY_VM_CHECK_INTS, and mjit_worker.c's exec_process (especially around vm->waitpid_lock and ruby_waitpid_locked), which prevents the main thread in native_ppoll_sleep from waking up forever.
- That had been *hidden* until MJIT postponed_job was introduced and MJIT stopped to spuriously wake up the main thread by SIGCHLD while the main thread is in native_ppoll_sleep.

Given that, your description

> Following how mjit_worker.c currently works, rb_f_system now ensures the VM-wide waitpid lists is locked before creating a new process via fork/vfork.

and the patch that locks waitpid_lock before fork made sense to me.

While I'm not sure in what kind of actual steps with thread interleaving rb_waitpid callers could steal work, your patch seems to improve the situation. Let's commit that and see what happens to trunk-mjit and trunk-mjit-wait on ko1's CIs, at least for failing inside rb_f_system.

The recent failure is http://ci.rvm.jp/results/trunk-mjit@silicon-docker/1435576, which hangs on Process.wait2 for pid created by Kernel#spawn in EnvUtil.invoke_ruby and appears more frequently than #system one. For this case, I guess changes similar to rb_f_system (creating and passing waitpid_state) are needed for rb_f_spawn (and all related families that fire rb_execarg_spawn) as well?

By the way, thanks to take a look at this. It would take a lot of time if I tried to resolve this alone.

#93 - 10/29/2018 04:52 PM - MSP-Greg (Greg L)

- File 14867_91_mingw_build_log_.txt added

normalperson (Eric Wong) wrote:

> Greg: everything should be confined to *nix for now, but can you
> make sure I didn't break anything on Windows? Thanks.

Eric,

I normally just build mingw, so I'm not testing the mswin/vc builds. I added the patch, and several build errors, starting at line 2254 of the attached log. I then removed the patch, and all built fine...

Thanks, Greg

#94 - 10/29/2018 11:42 PM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> normalperson (Eric Wong) wrote:
>
> > Greg: everything should be confined to *nix for now, but can you
> > make sure I didn't break anything on Windows? Thanks.
>
> Eric,
>
> I normally just build mingw, so I'm not testing the mswin/vc builds. I added the patch, and several build errors, starting at line 2254 of the attached log. I then removed the patch, and all built fine...

Greg: very odd, it's breaking in bigdecimal which isn't
affected by any changes in internal.h

I see process.c compiled fine and the patch only changed
process.c and internal.h from your log, so I'm going to assume
there's something else amiss with labs + bigdecimal unrelated
in your build...

#95 - 10/30/2018 01:22 AM - normalperson (Eric Wong)

> I normally just build mingw, so I'm not testing the mswin/vc builds. I added the patch, and several build errors, starting at line 2254 of the attached log. I then removed the patch, and all built fine...

> Greg: very odd, it's breaking in bigdecimal which isn't
> affected by any changes in internal.h
>
> I see process.c compiled fine and the patch only changed
> process.c and internal.h from your log, so I'm going to assume
> there's something else amiss with labs + bigdecimal unrelated
> in your build...

sigh reverted r65434
None of these errors seem related, even
http://mswinci.japaneast.cloudapp.azure.com/vc12-x64/ruby-trunk/log/20181030T003541Z.fail.html.gz

#96 - 10/30/2018 01:42 AM - normalperson (Eric Wong)

> *sigh* reverted r65434
> None of these errors seem related, even
> http://mswinci.japaneast.cloudapp.azure.com/vc12-x64/ruby-trunk/log/20181030T003541Z.fail.html.gz

Trying r65437

```
diff --git a/9ee245cba9 b/13af9d182c
index 9ee245cba9..13af9d182c 100644
--- a/9ee245cba9
+++ b/13af9d182c
@@ -4294,7 +4294,9 @@ rb_spawn_process(struct rb_execarg *eargp, char *errmsg, size_t errmsg_buflen)
rb_last_status_set((status & 0xff) << 8, 0);
```

```
pid = 1;            /* dummy */
 # endif
-
+    if (eargp->waitpid_state) {
+        eargp->waitpid_state->pid = pid;
+    }
rb_execarg_run_options(&sarg, NULL, errmsg, errmsg_buflen);
 #endif
return pid;
```

#97 - 10/30/2018 02:29 AM - k0kubun (Takashi Kokubun)

Congrats, while mswinci's vc12-x64 is not finished yet, r65437 at least worked on AppVeyor CI (mswin, mingw). That seems to work on Windows now.

#98 - 10/30/2018 03:04 AM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> Congrats, while mswinci's vc12-x64 is not finished yet, r65437
> at least worked on AppVeyor CI (mswin, mingw). That seems to
> work on Windows now.

Thank you for update, I was refreshing rubyci.org and getting
impatient :<

I am going to move more data into rb_mjit_unit struct for state
management using callbacks and events. It looks like
rb_mjit_unit_node is useless so I will get rid of it and
embed ccan/list node into rb_mjit_unit.

compact_units list is useless and you have no plans for it right?

#99 - 10/30/2018 03:12 AM - normalperson (Eric Wong)

> I am going to move more data into rb_mjit_unit struct for state
> management using callbacks and events. It looks like
> rb_mjit_unit_node is useless so I will get rid of it and
> embed ccan/list node into rb_mjit_unit.
>
> compact_units list is useless and you have no plans for it right?

I'm testing this patch for ccan/list in rb_mjit_unit:

https://80x24.org/spew/20181030030441.10036-1-e@80x24.org/raw

Howeever, I think I found an old bug. Accessing
unit->iseq->body outside of critical section seems wrong and
I hit a segfault:

```
CRITICAL_SECTION_FINISH(3, "before mjit_compile to wait GC finish");

{
VALUE s = rb_iseq_path(unit->iseq);
const char *label = RSTRING_PTR(unit->iseq->body->location.label);
const char *path = RSTRING_PTR(s);
int lineno = FIX2INT(unit->iseq->body->location.first_lineno);
verbose(2, "start compilation: %s@%s:%d -> %s", label, path, lineno, c_file);
fprintf(f, "/* %s@%s:%d */\n\n", label, path, lineno);
}
success = mjit_compile(f, unit->iseq->body, funcname, cc_entries, is_entries);

/* release blocking mjit_gc_start_hook */
CRITICAL_SECTION_START(3, "after mjit_compile to wakeup client for GC");
```

#100 - 10/30/2018 03:15 AM - k0kubun (Takashi Kokubun)

> compact_units list is useless and you have no plans for it right?

Oh, it's intended to be "List of compacted so files which will be deleted in mjit_finish()" so the current behavior is a bug. I'll fix it shortly, so please leave it as is.

#101 - 10/30/2018 04:00 AM - MSP-Greg (Greg L)

normalperson (Eric Wong)

r65437 passed.

> very odd, it's breaking in bigdecimal which isn't affected by any changes in internal.h

I saw that, and knew I could look at it all day and never find the cause. The builds with and without the patch both ran on r65429. Regardless, it's good now...

Thanks, Greg

#102 - 10/30/2018 05:31 AM - k0kubun (Takashi Kokubun)

> Howeever, I think I found an old bug. Accessing unit->iseq->body outside of critical section seems wrong and I hit a segfault:

Thanks to report, I'll take a look at that once I go home.

#103 - 10/30/2018 12:36 PM - k0kubun (Takashi Kokubun)

> I'm testing this patch for ccan/list in rb_mjit_unit:
> https://80x24.org/spew/20181030030441.10036-1-e@80x24.org/raw

It's okay to merge it. Migration to ccan/list was one of things I wanted to do later. Though, as said above, I'll resurrect compact_units to clean up them on ruby_cleanup anyway after you commit that.

> Howeever, I think I found an old bug. Accessing unit->iseq->body outside of critical section seems wrong and I hit a segfault:

Accessing iseq->body in the part you quote was completely intentional. Prior to leaving the critical section before mjit_compile, it waits for the current GC finish and set in_jit. With in_jit set to TRUE, other Ruby threads may not start GC due to mjit_gc_start_hook. Even while we're running tests with --jit for 24 hours on ko1's CI, I don't remember SEGV on that line.

If the SEGV is reproductive on trunk without your patch, please let me know so that I can start looking at it.

#104 - 10/30/2018 03:31 PM - k0kubun (Takashi Kokubun)

normalperson (Eric Wong) By the way, is there any plan to apply rb_f_system-like changes to rb_f_spawn as well? Many of "1. waitpid" deadlocks seem to come from a process created by rb_f_spawn, and also I guess those waitpid-related race conditions may also result in "2. in ruby_cleanup" deadlocks as well. So it would be worth taking a look since we may be releasing 2.6.0 preview3 shortly.

I briefly took a look, but at least we can't use alloca for waitpid_state on rb_f_spawn and so the code for it would be slightly different from rb_f_system's one. I'll leave it to you since you're more familiar with the current implementation (and possible race conditions) around waitpid.

#105 - 10/30/2018 04:22 PM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> normalperson (Eric Wong) By the way, is there any plan to apply rb_f_system-like changes to rb_f_spawn as well? Many of "1. waitpid" deadlocks seem to come from a process created by rb_f_spawn, and also I guess those waitpid-related race conditions may also result in "2. in ruby_cleanup" deadlocks as well. So it would be worth taking a look since we may be releasing 2.6.0 preview3 shortly.

I'm not sure that's the problem, actually; and holding vm->waitpid_lock across two Ruby method calls won't work.

Fwiw, the missing locks ([ruby-core:89629]) leading to data corruption could be causing some these timeouts/pauses.

> I briefly took a look, but at least we can't use alloca for waitpid_state on rb_f_spawn and so the code for it would be slightly different from rb_f_system's one. I'll leave it to you since you're more familiar with the current implementation (and possible race conditions) around waitpid.

AFAIK, the waitpid code has been good for a while until you started using postponed job, right? I was mostly away for a few weeks along with several computer hardware problems.

Since the waitpid problems seems new, that leads me to more strongly suspect MJIT is clobbering some memory which the waitpid/locking stuff relies on.

#106 - 10/30/2018 11:48 PM - k0kubun (Takashi Kokubun)

> holding vm->waitpid_lock across two Ruby method calls won't work.

Oh, I see.

> AFAIK, the waitpid code has been good for a while until you started using postponed job, right?

Kind of right. But as replied previously, my assumption is:

"That had been hidden until MJIT postponed_job was introduced and MJIT stopped to spuriously wake up the main thread by SIGCHLD while the main thread is in native_ppoll_sleep."

Prior to MJIT postponed_job, MJIT worker thread could infinitely spawn gcc processes while Ruby main thread is sleeping under waitpid. And I'm thinking that firing SIGCHLD handlers (after actual SIGCHLD which Ruby main thread was waiting for but missed) had been *accidentally* succeeded to wake up the Ruby main thread.

Since I'm not so familiar around the current waipid mechanism, my assumptions may go wrong. So please let me know if the above guess was not valid at all. At least my understanding of possible effects to waitpid by my recent change around MJIT is just "it may stop JIT-ing continously, but it doesn't change how it spawns/waits for a gcc process". We may be able to experiment to spuriously send SIGCHLD to main thread to let it call ruby_waitpid_all after some timeout of waiting for MJIT postponed_job, or finishing postponed_job during main thread sleep and firing gcc may let it work again.

And still recent builds are waiting for a process created by Process.spawn forever in native_ppoll_sleep of Process.wait2 and Process.waitpid2.
http://ci.rvm.jp/results/trunk-mjit@silicon-docker/1437559
http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1437673

About #system, the way of deadlock seems to be changed after r65437:
http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1437880

Could you take a look at at least the last one? Is it fine to call ubf_select inside ruby_waitpid_locked call in MJIT worker thread?

#107 - 10/31/2018 12:01 AM - k0kubun (Takashi Kokubun)

By the way,

> holding vm->waitpid_lock across two Ruby method calls won't work.

first of all I may not need to do this, since just locking from fork/vfork to waiting_pids modification could work. I'll take a look at that part again.

#108 - 10/31/2018 02:42 AM - normalperson (Eric Wong)

> About #system, the way of deadlock seems to be changed after r65437:
> http://ci.rvm.jp/results/trunk-mjit-wait@silicon-docker/1437880
>
> Could you take a look at at least the last one? Is it fine to call ubf_select inside ruby_waitpid_locked call in MJIT worker thread?

ubf_select inside ruby_waitpid_locked is safe, but contention on vm->gvl.lock seems wrong. Made r65465 which should fix rb_f_system.

Also taking a look at native_ppoll_sleep for spawn; but I would

feel more comfortable investigating issues if possible race
condition around iseq->body access in MJIT worker could be
eliminated as source of data corruption.

I also started a patch series (two so far) to simplify MJIT and
make it more event-oriented:

https://80x24.org/spew/20181030184614.3830-1-e@80x24.org/raw
https://80x24.org/spew/20181030184614.3830-2-e@80x24.org/raw

So perhaps mjit worker thread can be eliminated in the future...

#109 - 10/31/2018 02:55 AM - k0kubun (Takashi Kokubun)

> ubf_select inside ruby_waitpid_locked is safe, but contention on vm->gvl.lock seems wrong. Made r65465 which should fix rb_f_system.
>
> Also taking a look at native_ppoll_sleep for spawn;

Thanks for taking a look at them :)

> but I would feel more comfortable investigating issues if possible race condition around iseq->body access in MJIT worker could be eliminated as source of data corruption.

Ok. As said above, I believe it's properly guarded with locks and in_jit, but I'll take a look at that after https://80x24.org/spew/20181030184614.3830-1-e@80x24.org/raw is committed. Could you check in ccan/list one first? I'll check https://80x24.org/spew/20181030184614.3830-2-e@80x24.org/raw one later, but ccan change should be safe to merge separately.

> So perhaps mjit worker thread can be eliminated in the future...

I'm hoping to see that too for better reliability. Please let me confirm the benchmark results prior to committing it once it's made possible.

#110 - 10/31/2018 03:12 AM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> > ubf_select inside ruby_waitpid_locked is safe, but contention on vm->gvl.lock seems wrong. Made r65465 which should fix rb_f_system.
> >
> > Also taking a look at native_ppoll_sleep for spawn;
>
> Thanks for taking a look at them :)

OK, I think it's the lack of locking/atomics around
checking waitpid_state.ret and I introduced the bug
in commit 9d4027b50334ef804f6f138fba1d342fe188826b
("process.c: simplify SIGCHLD-based waitpid").

I redesign rb_thread_sleep_interruptible tomorrow. Really tired and stressed
now from crappy keyboard.

> > but I would feel more comfortable investigating issues if possible race condition around iseq->body access in MJIT worker could be eliminated as source of data corruption.
>
> Ok. As said above, I believe it's properly guarded with locks and in_jit, but I'll take a look at that after https://80x24.org/spew/20181030184614.3830-1-e@80x24.org/raw is committed. Could you check in ccan/list one first? I'll check https://80x24.org/spew/20181030184614.3830-2-e@80x24.org/raw one later, but ccan change should be safe to merge separately.

Sorry, I missed your message about in_jit. Trying ccan/list now
as r65468

#111 - 10/31/2018 06:52 AM - normalperson (Eric Wong)

> Sorry, I missed your message about in_jit. Trying ccan/list now
> as r65468

Reverted r65468 for now because all MJIT CIs failed (sorry, was tired and needed to go afk for a bit)

#112 - 10/31/2018 07:04 AM - normalperson (Eric Wong)

> Sorry, I missed your message about in_jit. Trying ccan/list now as r65468
>
> Reverted r65468 for now because all MJIT CIs failed (sorry, was tired and needed to go afk for a bit)

Possible fix on top of r65468 (if unreverted) but I'm too tired to wait for local tests to finish a few times right now...

```
diff --git a/mjit_worker.c b/mjit_worker.c
index 5008e6d8bc..b8ebb29812 100644
--- a/mjit_worker.c
+++ b/mjit_worker.c
@@ -1041,6 +1041,13 @@ convert_unit_to_func(struct rb_mjit_unit *unit, struct rb_call_cache *cc_entries
in_jit = TRUE;
CRITICAL_SECTION_FINISH(3, "before mjit_compile to wait GC finish");

+    /* We need to check again here because we could've waited on GC above */
+    if (unit->iseq == NULL) {
+        if (!mjit_opts.save_temps)
+            remove_file(c_file);
+        free_unit(unit);
+        return (mjit_func_t)NOT_COMPILED_JIT_ISEQ_FUNC;
+    }
{
VALUE s = rb_iseq_path(unit->iseq);
const char *label = RSTRING_PTR(unit->iseq->body->location.label);
```

#113 - 10/31/2018 12:28 PM - k0kubun (Takashi Kokubun)

> I'm too tired to wait for local tests to finish a few times right now...

Take your time!

> Possible fix on top of r65468

Thanks, that looks good for the fix of previous SEGV. To avoid our conflict, I re-applied your work with that fix on r65473. I'm watching trunk-mjit and trunk-mjit-wait CIs.

#114 - 10/31/2018 01:15 PM - k0kubun (Takashi Kokubun)

r65473 didn't work either. So I reverted that again on r65474.

I think doing 2 separate complicated things are hard like this, so the refactoring change (migration to ccan) and behavior change ("rb_mjit_unit can either exist in unit_queue or active_units, but not both") should be done separately.

#115 - 10/31/2018 07:32 PM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> r65473 didn't work either. So I reverted that again on r65474.
>
> I think doing 2 separate complicated things are hard like this, so the refactoring change (migration to ccan) and behavior change ("rb_mjit_unit can either exist in unit_queue or active_units, but not both") should be done separately.

I think the original problem is related to GC, my original patch did not work and got stuck because it left in_jit in a bad state (and leaked FILE *f).
This is a refined version which works with and without my ccan/list change:

https://80x24.org/spew/20181031191919.26547-1-e@80x24.org/raw

The ccan/list patch seems fine, but I'll commit the above one, first.

#116 - 10/31/2018 10:44 PM - k0kubun (Takashi Kokubun)

Looks good. Please commit that first.

#117 - 10/31/2018 11:03 PM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> Looks good. Please commit that first.

Actually, I'm pretty sure that extra check is a noop because
unit_queue is a GC mark root. The difference is ccan/list
change removes from unit_queue during get_from_list, so
it won't be marked...

(Still testing)

#118 - 11/01/2018 01:27 AM - k0kubun (Takashi Kokubun)

Cool, r65475 looks working on trunk-mjit and trunk-mjit-wait CIs so far. Thanks.

#119 - 11/01/2018 01:44 AM - k0kubun (Takashi Kokubun)

Btw thanks to leave compact_units in it. It looks like it's already valuable for cleaning up units by free_list on mjit_unit.

#120 - 11/01/2018 02:20 AM - k0kubun (Takashi Kokubun)

I tried to apply an easy fix similar to rb_f_system's one to rb_f_spawn but I got Errno::ECHILD: No child processes. So,

> > Made r65465 which should fix rb_f_system.
>
> OK, I think it's the lack of locking/atomics around checking waitpid_state.ret and I introduced the bug in commit 9d4027b50334ef804f6f138fba1d342fe188826b ("process.c: simplify SIGCHLD-based waitpid").
>
> I redesign rb_thread_sleep_interruptible tomorrow. Really tired and stressed now from crappy keyboard.

I'll just wait for this one for deadlock issue by rb_f_spawn and rb_waitpid.

Also, this thread is already too long and so hard to follow this ticket on the browser. Let's file another ticket for talking about the topic for simplifying MJIT worker and moving to event-based MJIT.

#121 - 11/01/2018 11:03 AM - normalperson (Eric Wong)

I wrote:

> > http://ci.rvm.jp/results/trunk-mjit@silicon-docker/1437559
> > OK, I think it's the lack of locking/atomics around
> > checking waitpid_state.ret and I introduced the bug
> > in commit 9d4027b50334ef804f6f138fba1d342fe188826b
> > ("process.c: simplify SIGCHLD-based waitpid").
>
> I redesign rb_thread_sleep_interruptible tomorrow.

Nope, only native_ppoll_sleep path in native_sleep was
vulnerable to that, I think. Instead, I will add an extra
FD which only the main thread sleeps on:

https://80x24.org/spew/20181101103147.18505-1-e@80x24.org/raw

One downside is (rare?) pthreads platforms without POSIX timers
will now have six pipe FDs (3 pipes) at startup (Ruby 2.5 had 4
pipe FDs).

Now, I don't know which platforms lack POSIX timers. I know
FreeBSD has them; and Linux can even use cheaper eventfd.
(we also had CI failures due to system-wide pipe limit, so
I favor eventfd when possible)

#122 - 11/01/2018 02:30 PM - k0kubun (Takashi Kokubun)

Thanks for your continuous effort on it. We saw another failure on rb_f_system http://ci.rvm.jp/results/trunk-mjit@silicon-docker/1440570 on r65493, but I hope timer solves the issue as well on r65495.

#123 - 11/03/2018 07:39 AM - k0kubun (Takashi Kokubun)

- Status changed from Assigned to Closed

The CIs have been stable since r65495. I think this issue seems to be fixed now. Thank you so much!

I close this ticket. Please file another ticket if you work on moving to event-based MJIT worker.

#124 - 11/03/2018 12:42 PM - normalperson (Eric Wong)

takashikkbn@gmail.com wrote:

> I close this ticket. Please file another ticket if you work on
> moving to event-based MJIT worker.

Thanks, but I don't know if it can happen in time for 2.6 :<

Judging by how often processes are stuck doing read(2) on
sockets and pipes. Bug #14968
should make a big difference in usability/performance of
event-based MJIT. However, non-blocking portability is
a concern...

One tangential goal is parallel MJIT builds. ccan/list +
get_from_list change in r65475 is a big step in making that
happen, at least.

#125 - 11/19/2018 01:10 PM - k0kubun (Takashi Kokubun)

- Related to Bug #15320: IO.popen with MJIT worker thread may deadlock added

Files

0001-hijack-SIGCHLD-handler-for-internal-use.patch	13.8 KB	06/23/2018	normalperson (Eric Wong)
JIT-test-all.log	39.9 KB	06/30/2018	MSP-Greg (Greg L)
mjit_test-all_63796.log	40.4 KB	06/30/2018	MSP-Greg (Greg L)
config_ruby-loco_mingw.log	27 KB	07/06/2018	MSP-Greg (Greg L)
test_jit_results.txt	41.2 KB	07/06/2018	MSP-Greg (Greg L)
14867_91_mingw_build_log_.txt	127 KB	10/29/2018	MSP-Greg (Greg L)