Ruby master - Misc #14937

[PATCH] thread_pthread: lazy-spawn timer-thread only on contention

07/24/2018 11:48 PM - normalperson (Eric Wong)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

[ruby-core:87773]

thread_pthread: lazy-spawn timer-thread only on contention

To reduce resource use and reduce CI failure; lazy spawn timer-thread only in processes which use Ruby Threads AND those Ruby Threads hit contention. Single-threaded Ruby processes (including forked children) will never have timer-thread overhead.

To simplify the thread_pthread.c code, I eliminated busy timer thread [Misc #14851]. Maybe the thread_win32.c code can use self-pipe, too; and they won’t need busy wakeups.

There is only one self-pipe, now, as wakeups for timeslice are handled via condition variables. This reduces FD pressure slightly.

Signal handling is handled directly by one Ruby Thread (instead of timer-thread) by exposing signal self-pipe to callers of rb_thread_fd_select, native_sleep, rb_wait_for_single_fd, etc...

Acquiring, using, and releasing the self-pipe is exposed via 4 new internal functions:

1) rb_sigwait_fd_get - exclusively acquire timer_thread_pipe.normal[0]
2) rb_sigwait_fd_sleep - sleep and wait for signal (and no other FDs)
3) rb_sigwait_fd_put - release acquired result from rb_sigwait_fd_get
4) rb_sigwait_fd_migrate - migrate signal handling to another thread after calling rb_sigwait_fd_put.

rb_sigwait_fd_migrate is necessary for waitpid callers because only one thread can wait on self-pipe at a time, otherwise a deadlock will occur if threads fight over the self-pipe.

TRAP_INTERRUPT_MASK is now set for the main thread directly in signal handler via rb_thread_wakeup_timer_thread.

Originally, I wanted to use POSIX timers (timer_create/timer_settime) and avoid timer-thread completely. Unfortunately, this proved unfeasible for one reason: Mutex#sleep resumes on spurious wakeups and test/thread/test_cv.rb::test_condvar_timed_wait failed.

In the future, I hope [Feature #14717] is accepted so Threads may be made non-preemptible. This will allow users to prevent timer-thread creation completely.

git repository also available at:
https://80x24.org/ruby.git tt-lazy
(commit a2990cefcdba55300ad44275ee44df18e6f95ce)
timer-thread may set trap interrupt with rb_threadptr_check_signal at any time independent of GVL. This means timer-thread may set the trap interrupt flag on the previous execution context; causing the flag to be unnoticed until a future ec switch (or lost completely if the ec is done).

Note: I avoid relying on th->interrupt_lock here and use atomics because we won't be able to rely on it for proposed lazy timer-thread [Misc #14937].

This regression affects Ruby 2.5 as it was introduced by moving interrupt_flag to `ec' which is an unstable pointer. Ruby <= 2.4 was unaffected because vm->main_thread->interrupt_flag never changed.

[ruby-core:88119] [Bug #14939]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64062 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64062 - 07/26/2018 08:30 AM - normalperson (Eric Wong)

timer-thread may set trap interrupt with rb_threadptr_check_signal at any time independent of GVL. This means timer-thread may set the trap interrupt flag on the previous execution context; causing the flag to be unnoticed until a future ec switch (or lost completely if the ec is done).

Note: I avoid relying on th->interrupt_lock here and use atomics because we won't be able to rely on it for proposed lazy timer-thread [Misc #14937].

This regression affects Ruby 2.5 as it was introduced by moving interrupt_flag to `ec' which is an unstable pointer. Ruby <= 2.4 was unaffected because vm->main_thread->interrupt_flag never changed.

[ruby-core:88119] [Bug #14939]

Revision 64062 - 07/26/2018 08:30 AM - normal

timer-thread may set trap interrupt with rb_threadptr_check_signal at any time independent of GVL. This means timer-thread may set the trap interrupt flag on the previous execution context; causing the flag to be unnoticed until a future ec switch (or lost completely if the ec is done).

Note: I avoid relying on th->interrupt_lock here and use atomics because we won't be able to rely on it for proposed lazy timer-thread [Misc #14937].

This regression affects Ruby 2.5 as it was introduced by moving interrupt_flag to `ec' which is an unstable pointer. Ruby <= 2.4 was unaffected because vm->main_thread->interrupt_flag never changed.

[ruby-core:88119] [Bug #14939]

Revision 708bfd21 - 07/29/2018 08:47 PM - normal

timer-thread: remove timer-thread by restructuring GVL

To reduce resource use and reduce CI failure; remove timer-thread. Single-threaded Ruby processes (including forked children) will never see extra thread overhead. This prevents glibc and jemalloc from going into multi-threaded mode and initializing locks or causing fragmentation via arena explosion.
The GVL implements its own wait-queue as a ccan/list to permit controlling wakeup order. Timeslice under contention is handled by a designated timer thread (similar to choosing a "patrol_thread" for current deadlock checking).

There is only one self-pipe, now, as wakeups for timeslice are done independently using condition variables. This reduces FD pressure slightly.

Signal handling is handled directly by a Ruby Thread (instead of timer-thread) by exposing signal self-pipe to callers of rb_thread_fd_select, native_sleep, rb_wait_for_single_fd, etc...

Acquiring, using, and releasing the self-pipe is exposed via 4 new internal functions:

1) rb_sigwait_fd_get - exclusively acquire timer_thread_pipe.normal[0]
2) rb_sigwait_fd_sleep - sleep and wait for signal (and no other FDs)
3) rb_sigwait_fd_put - release acquired result from rb_sigwait_fd_get
4) rb_sigwait_fd_migrate - migrate signal handling to another thread after calling rb_sigwait_fd_put.

rb_sigwait_fd_migrate is necessary for waitpid callers because only one thread can wait on self-pipe at a time, otherwise a deadlock will occur if threads fight over the self-pipe.

TRAP_INTERRUPT_MASK is now set for the main thread directly in signal handler via rb_thread_wakeup_timer_thread.

Originally, I wanted to use POSIX timers (timer_create/timer_settime) for this. Unfortunately, this proved unfeasible as Mutex#sleep resumes on spurious wakeups and test/thread/test_cv.rb::test_condvar_timed_wait failed. Using pthread_sigmask to mask out SIGVTALRM fixed that test, but test/fiddle/test_function.rb::test_nogvl_poll proved there'd be some unavoidable (and frequent) incompatibilities from that approach.

Finally, this allows us to drop thread_destruct_lock and interrupt current ec directly.

We don't need to rely on vm->thread_destruct_lock or a coherent vm->running_thread on any platform. Separate timer-thread for time slice and signal handling is relegated to thread_win32.c, now.

[ruby-core:88088] [Misc #14937]

Revision 64107 - 07/29/2018 08:47 PM - normalperson (Eric Wong)

thread_pthread: remove timer-thread by restructuring GVL

To reduce resource use and reduce CI failure; remove timer-thread. Single-threaded Ruby processes (including forked children) will never see extra thread overhead. This prevents glibc and jemalloc from going into multi-threaded mode and initializing locks or causing fragmentation via arena explosion.

The GVL is implements its own wait-queue as a ccan/list to permit controlling wakeup order. Timeslice under contention is handled by a designated timer thread (similar to choosing a "patrol_thread" for current deadlock checking).

There is only one self-pipe, now, as wakeups for timeslice are done independently using condition variables. This reduces FD pressure slightly.

Signal handling is handled directly by a Ruby Thread (instead of timer-thread) by exposing signal self-pipe to callers of rb_thread_fd_select, native_sleep, rb_wait_for_single_fd, etc...

Acquiring, using, and releasing the self-pipe is exposed via 4...
new internal functions:

1) rb_sigwait_fd_get - exclusively acquire timer_thread_pipe.normal[0]
2) rb_sigwait_fd_sleep - sleep and wait for signal (and no other FDs)
3) rb_sigwait_fd_put - release acquired result from rb_sigwait_fd_get
4) rb_sigwait_fd_migrate - migrate signal handling to another thread
   after calling rb_sigwait_fd_put.

rb_sigwait_fd_migrate is necessary for waitpid callers because
only one thread can wait on self-pipe at a time, otherwise a
deadlock will occur if threads fight over the self-pipe.

TRAP_INTERRUPT_MASK is now set for the main thread directly in
signal handler via rb_thread_wakeup_timer_thread.

Originally, I wanted to use POSIX timers
(timer_create/timer_settime) for this. Unfortunately, this
proved unfeasible as Mutex#sleep resumes on spurious wakeups and
test/thread/test_cv.rb::test_condvar_timed_wait failed. Using
pthread_sigmask to mask out SIGVTALRM fixed that test, but
test/fiddle/test_function.rb::test_nogvl_poll proved there'd be
some unavoidable (and frequent) incompatibilities from that
approach.

Finally, this allows us to drop thread_destruct_lock and
interrupt current ec directly.

We don't need to rely on vm->thread_destruct_lock or a coherent
vm->running_thread on any platform. Separate timer-thread for
time slice and signal handling is relegated to thread_win32.c,
now.

[ruby-core:88088] [Misc #14937]

Revision 64107 - 07/29/2018 08:47 PM - normal
thread pthread: remove timer-thread by restructuring GVL

To reduce resource use and reduce CI failure; remove
timer-thread. Single-threaded Ruby processes (including forked
children) will never see extra thread overhead. This prevents
glibc and jemalloc from going into multi-threaded mode and
initializing locks or causing fragmentation via arena explosion.

The GVL is implements its own wait-queue as a ccancel/list to
permit controlling wakeup order. Timeslice under contention is
handled by a designated timer thread (similar to choosing a
"patrol_thread" for current deadlock checking).

There is only one self-pipe, now, as wakeups for timeslice are
done independently using condition variables. This reduces FD
pressure slightly.

Signal handling is handled directly by a Ruby Thread (instead
of timer-thread) by exposing signal self-pipe to callers of
rb_thread_fd_select, native_sleep, rb_wait_for_single_fd, etc...
Acquiring, using, and releasing the self-pipe is exposed via 4
new internal functions:

1) rb_sigwait_fd_get - exclusively acquire timer_thread_pipe.normal[0]
2) rb_sigwait_fd_sleep - sleep and wait for signal (and no other FDs)
3) rb_sigwait_fd_put - release acquired result from rb_sigwait_fd_get
4) rb_sigwait_fd_migrate - migrate signal handling to another thread
   after calling rb_sigwait_fd_put.

rb_sigwait_fd_migrate is necessary for waitpid callers because
only one thread can wait on self-pipe at a time, otherwise a
deadlock will occur if threads fight over the self-pipe.
TRAP_INTERRUPT_MASK is now set for the main thread directly in signal handler via rb_thread_wakeup_timer_thread.

Originally, I wanted to use POSIX timers (timer_create/timer_settime) for this. Unfortunately, this proved unfeasible as Mutex#sleep resumes on spurious wakeups and test/thread/test_cv.rb::test_condvar_timed_wait failed. Using pthread_sigmask to mask out SIGVTALRM fixed that test, but test/fiddle/test_function.rb::test_nogvt_poll proved there'd be some unavoidable (and frequent) incompatibilities from that approach.

Finally, this allows us to drop thread_destruct_lock and interrupt current ec directly.

We don't need to rely on vm->thread_destruct_lock or a coherent vm->running_thread on any platform. Separate timer-thread for time slice and signal handling is relegated to thread_win32.c, now.

[ruby-core:88088] [Misc #14937]

Revision 4c1ab82f - 07/29/2018 10:19 PM - normal
thread_pthread.c (ubf_select): refix [Bug #5343]

We still need to to designate a timer thread after registering target thread for the ubf list.

Oops :x

Note: I was never able to reproduce
test/thread/test_queue.rb::test_thr_kill failures on my on Debian machines.

[ruby-core:88088] [Misc #14937] [Bug #5343]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64108 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64108 - 07/29/2018 10:19 PM - normalperson (Eric Wong)
thread_pthread.c (ubf_select): refix [Bug #5343]

We still need to to designate a timer thread after registering target thread for the ubf list.

Oops :x

Note: I was never able to reproduce
test/thread/test_queue.rb::test_thr_kill failures on my on Debian machines.

[ruby-core:88088] [Misc #14937] [Bug #5343]

Revision 64108 - 07/29/2018 10:19 PM - normal
thread_pthread.c (ubf_select): refix [Bug #5343]

We still need to to designate a timer thread after registering target thread for the ubf list.

Oops :x

Note: I was never able to reproduce
test/thread/test_queue.rb::test_thr_kill failures on my on Debian machines.

[ruby-core:88088] [Misc #14937] [Bug #5343]

Revision 26b8a70b - 07/30/2018 02:28 AM - normal
thread_pthread.c (rb_sigwait_sleep): re-fix [Bug #5343] harder

We can't always designate a timer thread, so any sleepers must also perform ubf wakeups. Note: a similar change needs to be
made for rb_thread_fd_select and rb_wait_for_single_fd.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64111 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64111 - 07/30/2018 02:28 AM - normalperson (Eric Wong)
thread_pthread.c (rb_sigwait_sleep): re-fix [Bug #5343] harder

We can't always designate a timer thread, so any sleepers must also perform ubf wakeups. Note: a similar change needs to be made for rb_thread_fd_select and rb_wait_for_single_fd.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64111 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64111 - 07/30/2018 02:28 AM - normal
thread_pthread.c (rb_sigwait_sleep): re-fix [Bug #5343] harder

We can't always designate a timer thread, so any sleepers must also perform ubf wakeups. Note: a similar change needs to be made for rb_thread_fd_select and rb_wait_for_single_fd.

Revision 45143629 - 07/30/2018 04:28 AM - normal
thread_pthread.c (rb_sigwait_sleep): fix uninitialized poll set in UBF case

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64113 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64113 - 07/30/2018 04:28 AM - normalperson (Eric Wong)
thread_pthread.c (rb_sigwait_sleep): fix uninitialized poll set in UBF case

Revision 64113 - 07/30/2018 04:28 AM - normal
thread_pthread.c (rb_sigwait_sleep): fix uninitialized poll set in UBF case

Revision ab47a57a - 07/30/2018 05:51 AM - normal
thread*.c: waiting on sigwait_fd performs periodic ubf wakeups

We need to be able to perform periodic ubf_list wakeups when a thread is sleeping and waiting on signals.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64115 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64115 - 07/30/2018 05:51 AM - normalperson (Eric Wong)
thread*.c: waiting on sigwait_fd performs periodic ubf wakeups

We need to be able to perform periodic ubf_list wakeups when a thread is sleeping and waiting on signals.

Revision 64115 - 07/30/2018 05:51 AM - normal
thread*.c: waiting on sigwait_fd performs periodic ubf wakeups

We need to be able to perform periodic ubf_list wakeups when a thread is sleeping and waiting on signals.
Revision 48b6bd74 - 08/13/2018 09:34 PM - normal
thread_pthread.c: eliminate timer thread by restructuring GVL

This reverts commit 194a6a2c68e9c8a3536b24db18ceac87535a6051 (r64203).

Race conditions which caused the original reversion will be fixed in the subsequent commit.

[ruby-core:88360] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64352 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64352 - 08/13/2018 09:34 PM - normalperson (Eric Wong)
thread_pthread.c: eliminate timer thread by restructuring GVL

This reverts commit 194a6a2c68e9c8a3536b24db18ceac87535a6051 (r64203).

Race conditions which caused the original reversion will be fixed in the subsequent commit.

[ruby-core:88360] [Misc #14937]

Revision 3dd6288d - 08/13/2018 09:34 PM - normal
thread_pthread: use POSIX timer or thread to get rid of races

This closes race condition where GVL is uncontended and a thread receives a signal immediately before calling the blocking function when releasing GVL:

  1) check interrupts
  2) release GVL
  3) blocking function

If signal fires after 1) but before 3), that thread may never wake up if GVL is uncontended

We also need to wakeup the ubf_list unconditionally on gvl_yield; because two threads can be yielding to each other while waiting on IO#close while waiting on threads in IO#read or IO#gets.

[ruby-core:88360] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64353 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64353 - 08/13/2018 09:34 PM - normalperson (Eric Wong)
thread_pthread: use POSIX timer or thread to get rid of races

This closes race condition where GVL is uncontended and a thread receives a signal immediately before calling the blocking function when releasing GVL:

  1) check interrupts
  2) release GVL
  3) blocking function

If signal fires after 1) but before 3), that thread may never wake up if GVL is uncontended

We also need to wakeup the ubf_list unconditionally on gvl_yield; because two threads can be yielding to each other
while waiting on IO#close while waiting on threads in IO#read or IO#gets.

[ruby-core:88360] [Misc #14937]

Revision 64353 - 08/13/2018 09:34 PM - normal
thread_pthread: use POSIX timer or thread to get rid of races

This closes race condition where GVL is uncontended and a thread receives a signal immediately before calling the blocking function when releasing GVL:

1) check interrupts
2) release GVL
3) blocking function

If signal fires after 1) but before 3), that thread may never wake up if GVL is uncontended

We also need to wakeup the ubf_list unconditionally on gvl_yield; because two threads can be yielding to each other while waiting on IO#close while waiting on threads in IO#read or IO#gets.

[ruby-core:88360] [Misc #14937]

Revision 8da12db1 - 08/13/2018 10:19 PM - normal
thread_pthread (rb_timer_arm): ignore UBF_TIMER_POSIX state 2

It looks like I forgot to account for a situation involving 3 threads.

[ruby-core:88360] [Misc #14937]
git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64354 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64354 - 08/13/2018 10:19 PM - normalperson (Eric Wong)
thread_pthread (rb_timer_arm): ignore UBF_TIMER_POSIX state 2

It looks like I forgot to account for a situation involving 3 threads.

[ruby-core:88360] [Misc #14937]

Revision 64354 - 08/13/2018 10:19 PM - normal
thread_pthread (rb_timer_arm): ignore UBF_TIMER_POSIX state 2

It looks like I forgot to account for a situation involving 3 threads.

[ruby-core:88360] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64355 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 1e769ce6 - 08/14/2018 12:31 AM - normal

test/fiddle/test_function.rb (test_nogvl_poll): stop timer hack

EINTR seems unavoidable in real programs (or MJIT), so maybe it's not worth dealing with. r64353 relies on POSIX timers to signal.

Switching pipes and sockets to non-blocking by default would let us get rid of POSIX timers, timer pthread and this hack:
https://bugs.ruby-lang.org/issues/14968

[ruby-core:88360] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64355 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64355 - 08/14/2018 12:31 AM - normalperson (Eric Wong)
test/fiddle/test_function.rb (test_nogvl_poll): stop timer hack
EINTR seems unavoidable in real programs (or MJIT), so maybe it's not worth dealing with. r64353 relies on POSIX timers to signal.

Switching pipes and sockets to non-blocking by default would let us get rid of POSIX timers, timer pthread and this hack: https://bugs.ruby-lang.org/issues/14968

[ruby-core:88360] [Misc #14937]

Revision 64355 - 08/14/2018 12:31 AM - normal

test/fiddle/test_function.rb (test_nogvl_poll): stop timer hack

EINTR seems unavoidable in real programs (or MJIT), so maybe it's not worth dealing with. r64353 relies on POSIX timers to signal.

Switching pipes and sockets to non-blocking by default would let us get rid of POSIX timers, timer pthread and this hack: https://bugs.ruby-lang.org/issues/14968

[ruby-core:88360] [Misc #14937]

Revision 291afc96 - 08/14/2018 02:24 AM - normal

thread_pthread.c: use CLOCK_REALTIME on SunOS (Solaris)

timer_create does not seem to support CLOCK_MONOTONIC on Solaris, and CLOCK_HIRES seems like it could fail with insufficient permissions:

https://docs.oracle.com/cd/E86824_01/html/E54766/timer-create-3c.html

(Only tested on Linux and FreeBSD)

[ruby-core:88360] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64356 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64356 - 08/14/2018 02:24 AM - normalperson (Eric Wong)

thread_pthread.c: use CLOCK_REALTIME on SunOS (Solaris)

timer_create does not seem to support CLOCK_MONOTONIC on Solaris, and CLOCK_HIRES seems like it could fail with insufficient permissions:

https://docs.oracle.com/cd/E86824_01/html/E54766/timer-create-3c.html

(Only tested on Linux and FreeBSD)

[ruby-core:88360] [Misc #14937]

Revision 64356 - 08/14/2018 02:24 AM - normal

thread_pthread.c: use CLOCK_REALTIME on SunOS (Solaris)

timer_create does not seem to support CLOCK_MONOTONIC on Solaris, and CLOCK_HIRES seems like it could fail with insufficient permissions:

https://docs.oracle.com/cd/E86824_01/html/E54766/timer-create-3c.html

(Only tested on Linux and FreeBSD)

[ruby-core:88360] [Misc #14937]

Revision 5dca7d86 - 08/15/2018 04:32 AM - normal

thread_pthread.c: additional UBF_TIMER == UBF_TIMER_PTHREAD guards

Hopefully this makes the code easier-to-follow

[ruby-core:88475] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64371 b2dd03c8-39d4-4d8f-98ff-823fe69b080e
Revision 64371 - 08/15/2018 04:32 AM - normal

thread_pthread.c: additional UBF_TIMER == UBF_TIMER_PTHREAD guards

Hopefully this makes the code easier-to-follow

[ruby-core:88475] [Misc #14937]

Revision 64371 - 08/15/2018 04:32 AM - normal

thread_pthread.c: additional UBF_TIMER == UBF_TIMER_PTHREAD guards

Hopefully this makes the code easier-to-follow

[ruby-core:88475] [Misc #14937]

Revision e4fd71d6 - 08/15/2018 04:32 AM - normal

thread_pthread.c: rename timer_thread_pipe to signal_self_pipe

This data structure has nothing to do with timers or threads.

[ruby-core:88475] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64372 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64372 - 08/15/2018 04:32 AM - normal

thread_pthread.c: rename timer_thread_pipe to signal_self_pipe

This data structure has nothing to do with timers or threads.

[ruby-core:88475] [Misc #14937]

Revision 64372 - 08/15/2018 04:32 AM - normal

thread_pthread.c: rename timer_thread_pipe to signal_self_pipe

This data structure has nothing to do with timers or threads.

[ruby-core:88475] [Misc #14937]

Revision 3dcf85e0 - 08/15/2018 04:32 AM - normal

thread_pthread.c: rename rb_timer_* to ubf_timer_*

These functions will not be exported outside of thread_pthread.c
and we need to clarify the timer here is used for ubf and not
timeslice.

[ruby-core:88475] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64373 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64373 - 08/15/2018 04:32 AM - normal

thread_pthread.c: rename rb_timer_* to ubf_timer_*

These functions will not be exported outside of thread_pthread.c
and we need to clarify the timer here is used for ubf and not
timeslice.

[ruby-core:88475] [Misc #14937]

Revision 64373 - 08/15/2018 04:32 AM - normal

thread_pthread.c: rename rb_timer_* to ubf_timer_*

These functions will not be exported outside of thread_pthread.c
and we need to clarify the timer here is used for ubf and not
timeslice.

[ruby-core:88475] [Misc #14937]

Revision 906ad167 - 08/15/2018 05:31 AM - normal
thread_pthread.h (native_thread_data): split list_node between ubf and gvl

Do not waste extra memory for each thread, but make thread_pthread.c easier-to-follow as a result.

[ruby-core:88475] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64375 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64375 - 08/15/2018 05:31 AM - normalperson (Eric Wong)
thread_pthread.h (native_thread_data): split list_node between ubf and gvl

Do not waste extra memory for each thread, but make thread_pthread.c easier-to-follow as a result.

[ruby-core:88475] [Misc #14937]

Revision 4610d36c - 08/15/2018 07:16 AM - normal
thread_pthread.c: hoist out do_gvl_timer and improve documentation

This hopefully clarifies the roles of UBF_TIMER and vm->gvl.timer

[ruby-core:88475] [Misc #14937]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64377 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64377 - 08/15/2018 07:16 AM - normalperson (Eric Wong)
thread_pthread.c: hoist out do_gvl_timer and improve documentation

This hopefully clarifies the roles of UBF_TIMER and vm->gvl.timer

[ruby-core:88475] [Misc #14937]

Revision 64377 - 08/15/2018 07:16 AM - normal
thread_pthread.c: hoist out do_gvl_timer and improve documentation

This hopefully clarifies the roles of UBF_TIMER and vm->gvl.timer

[ruby-core:88475] [Misc #14937]

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]

Revision 9fc4c79f - 10/11/2018 02:58 PM - nagachika (Tomoyuki Chikanaga)
merge revision(s) 64062: [Backport #14939]

cnt.c (ec_switch): prevent delayed/missed trap interrupt race

timer-thread may set trap interrupt with rb_threadptr_check_signal
at any time independent of GVL. This means timer-thread may set
the trap interrupt flag on the previous execution context; causing
the flag to be unnoticed until a future ec switch (or lost
completely if the ec is done).

Note: I avoid relying on th->interrupt_lock here and use
atomics because we won't be able to rely on it for proposed lazy
timer-thread [Misc #14937].

This regression affects Ruby 2.5 as it was introduced by moving
interrupt_flag to 'ec' which is an unstable pointer. Ruby <= 2.4
was unaffected because vm->main_thread->interrupt_flag never
changed.

Note: I avoid relying on th->interrupt_lock here and use
atomics because we won't be able to rely on it for proposed lazy
timer-thread [Misc #14937].

This regression affects Ruby 2.5 as it was introduced by moving
interrupt_flag to 'ec' which is an unstable pointer. Ruby <= 2.4
was unaffected because vm->main_thread->interrupt_flag never
changed.

History

#1 - 07/25/2018 12:04 AM - normalperson (Eric Wong)

https://bugs.ruby-lang.org/issues/14937

Hi Greg, Can you try the patch in this ticket? It shouldn't
affect win32, but maybe I made typos. I couldn't add you
as a watcher to this ticket from Redmine (since that requires
JavaScript :<)

Thanks.

#2 - 07/25/2018 12:22 AM - normalperson (Eric Wong)

git repository also available at:
https://80x24.org/ruby.git tt-lazy
(commit a2990cecfccba55300ad44275ee4adf18e6f95ece)
Now up to commit 6fb72ad7334a61e3c4341aa5fa17749671cbb867

[PATCH 2/1] thread.c (rb_threadptr_execute_interrupts): drain pipe and handle SIGCHLD
https://80x24.org/spew/20180725001947.77uutfvtkn3cq235@whir/raw

greg (Grégory Duchatelet): [PATCH 2/1] should not affect win32 users at all, either

#3 - 07/25/2018 12:42 AM - normalperson (Eric Wong)

git repository also available at:
https://80x24.org/ruby.git tl-lazy
(commit a2990cecfccba55300ad44275ee4adf18e6f95ece)

Now up to commit 6fb72ad7334a61e3c4341aa5fa17749671cbb867

Sorry, didn't finish testing the other one :x
now up to commit eac43426d79dd4d2e455d3bb8e161e014378893e

[PATCH 3/1] SQUASH: fixup FD draining
https://80x24.org/spew/20180725003101.ey2mmcu7mauvmfew@whir/raw

#4 - 07/25/2018 01:03 AM - MSP-Greg (Greg L)

normalperson (Eric Wong)

Eric,

Tested with the attached patch. When Appveyor is busy, and with test-all using retry, and something in MJIT failing or erroring, the test took 47 minutes.

Anyway, all good, ran with r64038, and all tests passed. Results below.

On Windows, Process.respond_to?(:daemon) returns false, so the changed test doesn't run...

Thanks, Greg

0 Total Failures/Errors Build No 975 Job Id e9se4873fpb4owjs
ruby 2.6.0dev (2018-07-25 trunk 64038) [x64-mingw32]
2018-07-25 00:45:29 UTC

  test-all 19258 tests, 2242240 assertions, 0 failures, 0 errors, 106 skips, 106 skips shown
  test-spec 3607 files, 27958 examples, 209426 expectations, 0 failures, 0 errors, 0 tagged
  mspec 3607 files, 27960 examples, 209308 expectations, 0 failures, 0 errors, 0 tagged

  test-basic test succeeded
  btest PASS all 1386 tests

#5 - 07/25/2018 08:12 AM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

Tested with the attached patch. When Appveyor is busy, and with test-all using retry, and something in MJIT failing or erroring, the test took 47 minutes.

 Anyway, all good, ran with r64038, and all tests passed. Results below.

Thanks for testing; I guess 47 minutes is not unusual for a busy CI machine?
It seems long... “make exam” on an ancient 1.6GHz Pentium-M (Centrino) laptop takes ~20 min on Debian 9.

#6 - 07/25/2018 08:22 AM - normalperson (Eric Wong)

Koichi Sasada ko1@atdot.net wrote:

I need to read your proposal with more concentrations, but one thing:
Thanks. I noticed a bug where it sometimes still get stuck when I run "make exam" in a loop. Will have to dig deeper to solve... (this is a tricky change :x)

On 2018/07/25 8:48, normalperson@yhbt.net wrote:

Mutex#sleep resumes on spurious wakeups and

I think Mutex#sleep should care about spurious wakeups, shouldn't?

I don't know, current behavior seems intentional and documented in RDoc:

- Note that this method can wakeup without explicit Thread#wakeup call.
- For example, receiving signal and so on. */

I don't know the reasoning behind it, maybe matz does.

#7 - 07/25/2018 08:32 AM - ko1 (Koichi Sasada)
On 2018/07/25 17:20, Eric Wong wrote:

I don't know, current behavior seems intentional and documented in RDoc:

* Note that this method can wakeup without explicit Thread#wakeup call.
* For example, receiving signal and so on. */

I don't know the reasoning behind it, maybe matz does.

Mutex#sleep is written by me :p

--
// SASADA Koichi at atdot dot net

#8 - 07/25/2018 12:21 PM - MSP-Greg (Greg L)

normalperson (Eric Wong)

Eric,

In addition to the build mentioned above, there have been two more ruby-loco builds, both with all three patches, and both passed all tests

Thanks for testing; I guess 47 minutes is not unusual for a busy CI machine?

Re Appveyor:

- 6 minutes is updating the build system
- I think each CI is run on 2 cores of a 4 core system, the system may be competing with another CI for disk IO, etc. The build for r64024 completed in 37 minutes.
- I run both make test-rubyspec and mspec.
- I think MJIT is one of the longer running suites, and with retry, it is always run twice.

Thanks again, Greg

#9 - 07/26/2018 08:30 AM - normalperson (Eric Wong)

- Status changed from Open to Closed

Applied in changeset trunk/r64062

cont.c (ec_switch): prevent delayed/missed trap interrupt race

timer-thread may set trap interrupt with rb_threadptr_check_signal at any time independent of GVL. This means timer-thread may set the trap interrupt flag on the previous execution context; causing the flag to be unnoticed until a future ec switch (or lost completely if the ec is done).
Note: I avoid relying on th->interrupt_lock here and use atomics because we won't be able to rely on it for proposed lazy timer-thread [Misc #14937].

This regression affects Ruby 2.5 as it was introduced by moving interrupt_flag to `ec` which is an unstable pointer. Ruby <= 2.4 was unaffected because vm->main_thread->interrupt_flag never changed.

[Bug #14937] [Bug #14939] lost trap interrupts during ec_switch
[Bug #14945] unregister_ubf_list ordering

And killed a lot of code from thread_pthread.c

Greg: can you test this patch instead? Thanks.

Btw, I can probably improve on this by making GVL waiter perform timeslice; so it would eliminate timer-thread completely. Will work on it tomorrow (too tired, now)

OK, new patch to eliminate timer-thread completely. It seems to pass repeated “make exam” and introduce no regressions. So far I have found and fixed two bugs while working on this.

[Bug #14939] lost trap interrupts during ec_switch
[Bug #14945] unregister_ubf_list ordering

And killed a lot of code from thread_pthread.c

Normalperson (Eric Wong)

Passed all tests using r64094. TestJIT#test_unload_units has been failing (both in parallel & retry) since it was added. Been meaning to post an issue/bug report about it...

Thanks, Greg
19273 tests, 2240557 assertions, 1 failures, 0 errors, 108 skips, 108 skips shown

Parallel Tests - 2 Failures

2 TestJIT#test_compile_insn_putstring_concatstrings_tostring = 1.32 s = F
1 TestJIT#test_unload_units = 4.77 s = F

Parallel Tests - 1 Error

0 TestIO#test_select_leak = 60.37 s = E

After Retry - 1 Failure
ruby/test_jit.rb
TestJIT#test_unload_units Line: 830
Failed to run script with JIT:

Hi Greg, thanks for testing. I'll let k0kubun deal with JIT failures. For threading, I found more code to delete (goes on top of above patch):

https://80x24.org/spew/20180729002256.22651-1-e@80x24.org/raw

Also pushed to my git repo:

The following changes since commit 443f4d583c8fe78198bee791f2ac3da0be2dfb5e:
are available in the Git repository at:
https://80x24.org/ruby.git tt-designated
for you to fetch changes up to 776e12582ba847e0d4b96fc1cf05e2cd7049ee7:
drop thread_destruct_lock and interrupt current ec directly (2018-07-29 00:16:27 +0000)

Eric Wong (2):
thread_pthread: remove timer-thread by restructuring GVL
drop thread_destruct_lock and interrupt current ec directly

<table>
<thead>
<tr>
<th>File</th>
<th>Insertions</th>
<th>Deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal.h</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>process.c</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>signal.c</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>test/ruby/test_process.rb</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>thread.c</td>
<td>327</td>
<td></td>
</tr>
<tr>
<td>thread_pthread.c</td>
<td>586</td>
<td></td>
</tr>
<tr>
<td>thread_pthread.h</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>thread_win32.c</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>vm_core.h</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

9 files changed, 526 insertions(+), 543 deletions(-)

Hi, Greg, thanks for testing. I'll let k0kubun deal with JIT failures. For threading, I found more code to delete (goes on top of above patch):

https://80x24.org/spew/20180729002256.22651-1-e@80x24.org/raw

Also pushed to my git repo:

The following changes since commit 443f4d583c8fe78198bee791f2ac3da0be2dfb5e:
are available in the Git repository at:
https://80x24.org/ruby.git tt-designated
for you to fetch changes up to 776e12582ba847e0d4b96fc1cf05e2cd7049ee7:
drop thread_destruct_lock and interrupt current ec directly (2018-07-29 00:16:27 +0000)

Eric Wong (2):
thread_pthread: remove timer-thread by restructuring GVL
drop thread_destruct_lock and interrupt current ec directly

<table>
<thead>
<tr>
<th>File</th>
<th>Insertions</th>
<th>Deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal.h</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>process.c</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>signal.c</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>test/ruby/test_process.rb</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>thread.c</td>
<td>327</td>
<td></td>
</tr>
<tr>
<td>thread_pthread.c</td>
<td>586</td>
<td></td>
</tr>
<tr>
<td>thread_pthread.h</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>thread_win32.c</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>vm_core.h</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

9 files changed, 526 insertions(+), 543 deletions(-)

Hi, Greg, thanks for testing. I'll let k0kubun deal with JIT failures. For threading, I found more code to delete (goes on top of above patch):

https://80x24.org/spew/20180729002256.22651-1-e@80x24.org/raw

Also pushed to my git repo:

The following changes since commit 443f4d583c8fe78198bee791f2ac3da0be2dfb5e:
are available in the Git repository at:
https://80x24.org/ruby.git tt-designated
for you to fetch changes up to 776e12582ba847e0d4b96fc1cf05e2cd7049ee7:
drop thread_destruct_lock and interrupt current ec directly (2018-07-29 00:16:27 +0000)

Eric Wong (2):
thread_pthread: remove timer-thread by restructuring GVL
drop thread_destruct_lock and interrupt current ec directly

<table>
<thead>
<tr>
<th>File</th>
<th>Insertions</th>
<th>Deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal.h</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>process.c</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>signal.c</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>test/ruby/test_process.rb</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>thread.c</td>
<td>327</td>
<td></td>
</tr>
<tr>
<td>thread_pthread.c</td>
<td>586</td>
<td></td>
</tr>
<tr>
<td>thread_pthread.h</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>thread_win32.c</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>vm_core.h</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

9 files changed, 526 insertions(+), 543 deletions(-)
Thanks again! I'll probably commit soon since this looks very straightforward (it took a long time to get there, though; probably the most challenging thing I've done for Ruby).

#16 - 07/30/2018 01:34 PM - k0kubun (Takashi Kokubun)
Congrats to achieve this. I think this should be a notable change written in NEWS :)

#17 - 07/30/2018 06:22 PM - normalperson (Eric Wong)
takashikkbn@gmail.com wrote:

Congrats to achieve this. I think this should be a notable change written in NEWS :)

Thanks, I plan to :-). I don't think we're out of the woods, yet, sky3 had a failure at http://ci.rvm.jp/results/trunk-test@ruby-sky3/1173398 and it might be related. Investigating...

#18 - 07/30/2018 10:52 PM - normalperson (Eric Wong)

yet, sky3 had a failure at http://ci.rvm.jp/results/trunk-test@ruby-sky3/1173398 and it might be related. Investigating...

I thought r64133 fixed it, but I still saw this failure:

http://ci.rvm.jp/results/trunk@P895/1173951

I can't reproduce that regardless of multi or single-core and I'm not sure what is wrong... (Maybe I need food :x)

Also, vm_thread_condvar1 performance regressed (otherwise, benchmark performance seems improved overall). But I think I need to redo sleep/wakeups again for auto-fiber [Feature #13618] anyways...

#19 - 08/06/2018 04:52 AM - normalperson (Eric Wong)

http://ci.rvm.jp/results/trunk@P895/1173951

Damnit, this is because IO#gets on blocking pipes doesn't hit rb_wait_for_single_fd and thus never gets to check sigwait_fd (self-pipe)

Anyways, I hope it is acceptable to make all pipes (and eventually sockets) non-blocking by default; or I will have to revert and reintroduce timer-thread :<

https://bugs.ruby-lang.org/issues/14968

/me resumes beating his head against the wall...

#20 - 08/06/2018 05:32 AM - normalperson (Eric Wong)

http://ci.rvm.jp/results/trunk@P895/1173951

or I will have to revert and reintroduce timer-thread :<

reverted for now :< r64203

/me resumes beating his head against the wall...
Btw, using POSIX timers is a solution for this; because
\texttt{timer\_settime(2)} is async-signal safe and I can call it
from signal handlers.

\textbullet{} (\texttt{pthreads} platforms w/o POSIX timers can use emulation via
extra thread)

Done:

The following changes since commit 52102f6ff50eebf8c16667c9b49cef579d2057c1:

test/lib/leakchecker.rb (find\_tempfiles): don't warn for missing files (2018-08-09 02:14:27 +0000)

are available in the Git repository at:

\url{https://80x24.org/ruby.git} tt-posix-v2

for you to fetch changes up to ec91b49271b280cee7b455a480b43867fb8e9fd:

\texttt{thread\_pthread}: use POSIX timer or thread to get rid of races (2018-08-09 03:45:42 +0000)

Eric Wong (2):
\texttt{thread\_pthread.c}: eliminate timer thread by restructuring GVL

\url{https://80x24.org/spew/20180809034618.20082-1-e@80x24.org/raw}

\texttt{thread\_pthread.c}: use POSIX timer or thread to get rid of races
\url{https://80x24.org/spew/20180809034618.20082-2-e@80x24.org/raw}

\begin{verbatim}
configure.ac | 8 +
internal.h   | 3 +
process.c    | 140 ++++++-
signal.c     | 7 +-
test/ruby/test_io.rb | 9 +-
test/ruby/test_process.rb | 2 +-
test/ruby/test_thread.rb | 5 +-
thread.c     | 397 ++++++++++++++++++-
thread\_pthread.c | 946 ++++++++++++++++++++++++++++++--------------------
thread\_pthread.h | 20 +- THREAD_WIN32.C | 29 +
VM core.h    | 8 +-
\end{verbatim}

12 files changed, 991 insertions(+), 583 deletions(-)

I will commit next week (may not be around over the weekend :<).

\#23 - 08/14/2018 08:09 AM - ko1 (Koichi Sasada)

Let me clear your commits.

Maybe "Description" of this ticket is obsolete.

They are my understandings (fix me if they are wrong):
● Your idea is using POSIX timer if possible.

#define UBF_TIMER_NONE 0
#define UBF_TIMER_POSIX 1
#define UBF_TIMER_PTHREAD 2

UBF_TIMER_POSIX uses POSIX timer. UBF_TIMER_PTHREAD is traditional timer thread approach.

BTW

* UBF_TIMER is to close TOCTTOU signal race on programs
  * without GVL contention blocking read/write to sockets.

I can't understand these lines.
What is related between GVL contention and read/write sockets?
Single thread application do not kick gvl_acquire_common and timer_thread_function() is not kicked, right?

● GVL acquire loop invoke timer_thread_function()

... sorry I'm giving up to understand them correctly.

Could you explain more for current trunk?

After your commit, there are no timer threads?

---

**Comments/Questions**

gvl_acquire_common

VM_ASSERT(th->unblock.func == 0 && "we reuse ubf_list for GVL waitq");

I strongly disagree that reusing. It is very confusing.

rb_timer_(|dis)arm()

English question (sorry). What does it mean with word "arm"?

native_cond_timeout()

This function is not good name because I misunderstand it is same as native_cond_timedwait().
Maybe I had named it :p

list_add_tail(&vm->gvl.waitq, &nd->ubf_list);

Why add it? Whatvm->gvl.waitq manages?
To specify designate_timer_thread()

There are two condvars gvlq and switch_cond. Is it intentional?
Who remove it from the list? list_top() removes? (impl doesn't seem such deletion).

ubf_wakeup_all_threads();

why wakeup all blocking threads?

rb_timer_create()

There is a guard #if UBF_TIMER == UBF_TIMER_POSIX but not for rb_timer_pthread_create().
I misunderstand that UBF_TIMER_POSIX needs timer pthread.

rb_timer_ prefix

making timer_ will help to understand they are local static functions.
If you want to expose them outside thread_pthread.c, they should be more verbose name like rb_vm_timer and so on to recognize they are no Timer class in Ruby world (it is my opinion).
Let me clear your commits.
Maybe "Description" of this ticket is obsolete.

I guess, I will add doc or comments to thread_pthread.c

They are my understandings (fix me if they are wrong):

- Your idea is using POSIX timer if possible.

Yes, but scope of "timer" here is reduced.

Old timer-thread did several things:

1) set timer-interrupt (100ms timeslice)
2) handle ubf list
3) check for signals

New GVL can handle all of these tasks.

However, application without GVL contention need UBF_TIMER_* to
deal with 2) and 3) reliably (unless [Bug #14968] to make all
pipes/sockets non-blocking is accepted).

#define UBF_TIMER_NONE 0
define UBF_TIMER_POSIX 1
define UBF_TIMER_PTHREAD 2

UBF_TIMER_POSIX uses POSIX timer. UBF_TIMER_PTHREAD is traditional timer thread approach.

Not exactly. UBF_TIMER_* does not handle 1) (timer-interrupt)

BTW

* UBF_TIMER is to close TOCTTOU signal race on programs
  * without GVL contention blocking read/write to sockets.

I can't understand these lines.
What is related between GVL contention and read/write sockets?
Single thread application do not kick gvl_acquire_common and timer_thread_function() is not kicked, right?

Right.

- GVL acquire loop invoke timer_thread_function()

... sorry I'm giving up to understand them correctly.

I think the confusing thing is UBF_TIMER is NOT for timer-interrupt.
timer_thread_function is only for timer-interrupt, now (not
signals or ubf_list).

Also, timer_thread_pipe needs to be renamed, since it has
nothing to do with timer, now.

Could you explain more for current trunk?

After your commit, there are no timer threads?

Correct (for platforms with POSIX timers, at least).

Comments/Questions

gvl_acquire_common
VM_ASSERT(th->unblock.func == 0 && "we reuse ubf_list for GVL waitq");

I strongly disagree that reusing. It is very confusing.

I don't want to make rb_thread_struct any bigger. I can make it a union to clarify use.

rb_timer_(|dis)arm()

**English question (sorry). What does it mean with word "arm"?**

"arm" as in enable the periodic timer, "disarm" stops the timer.

"arm"/"disarm" is used in Linux timer_settime(2) manpage. Ditto for setitimer(2) manpage (setitimer was used in Ruby 1.8; but POSIX obsoletes it in favor of timer_settime).

native_cond_timeout()

This function is not good name because I misunderstand it is same as native_cond_timedwait(). Maybe I had named it :p

Yes, I just used an existing function :P

list_add_tail(&vm->gvl.waitq, &nd->ubf_list);

Yes, basically we manage the GVL wait-queue ourselves, now, instead of relying on same linked-list in glibc. This is similar to my Mutex and autoload locking rewrite; as well as waitqueue implementation in Linux kernel.

This allows us more fine-grained control and ordering of GVL waiters.

There are two condvars gvlq and switch_cond. Is it intentional?

Yes, switch_cond is from old GVL by kosaki. It made Thread.pass much faster. I tried to get rid of switch_cond and switch_wait_cond, but could not match Thread.pass performance from old GVL.

Who remove it from the list? list_top() removes? (impl doesn't seem such deletion).

The same thread which calls list_add_tail calls list_del_init in gvl_acquire_common.

ubf_wakeup_all_threads();

why wakeup all blocking threads?

ubf_wakeup_all_threads always needs to be called periodically if there's threads in ubf_list. If GVL is contended, it is easy to do it from the thread which is already handling timer interrupt.

**rb_timer_create()**

There is a guard #if UBF_TIMER == UBF_TIMER_POSIX but not for rb_timer_pthread_create(). I misunderstand that UBF_TIMER_POSIX needs timer pthread.
Right, I tried to avoid extra ifdef there since
rb_timer_pthread_create is empty function for UBF_TIMER_POSIX
anyways.

I can add "if (UBF_TIMER == UBF_TIMER_PTHREAD)" conditional in
C (not CPP) if it clarifies things. I prefer to avoid CPP ifdef
guard when possible and efficient to do compiler checks in more
platform-specific code.

**rb_timer_ prefix**

making timer_ will help to understand they are local static functions.
If you want to expose them outside thread_pthread.c, they should be more verbose name like rb_vm_timer and so on to recognize they are no
Timer class in Ruby world (it is my opinion).

OK, I will change to "ubf_timer_" prefix, instead.
Bare "timer_" is confusing with POSIX functions, and
I don't think they need to be exposed outside of
thread_pthread.c

#25 - 08/15/2018 07:32 AM - normalperson (Eric Wong)

Eric Wong wrote:

ko1@atdot.net wrote:

Let me clear your commits.
Maybe "Description" of this ticket is obsolete.

I guess, I will add doc or comments to thread_pthread.c

I think I addressed this in r64377 and made clarifications
in naming/flow in r64375, r64373, r64372 and r64371.

Thank you for comments and please let us know if you have
any more questions.

**Files**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001-thread_pthread-lazy-spawn-timer-thread-only-on-conte.patch</td>
<td>42.3 KB</td>
<td>07/24/2018</td>
<td>normalperson (Eric Wong)</td>
</tr>
</tbody>
</table>