Instead of relying on a native condition variable and mutex for every Ruby Mutex object, use a doubly linked-list to implement a waiter queue in the Mutex. The immediate benefit of this is reducing the size of every Mutex object, as some projects have many objects requiring synchronization.

In the future, this technique using a linked-list and on-stack list node (struct mutex_waiter) should allow us to easily transition to M:N threading model, as we can avoid the native thread dependency to implement Mutex.

We already do something similar for autoload in variable.c, and this was inspired by the Linux kernel wait queue (as cc/list is inspired by the Linux kernel linked-list).

Finally, there are big performance improvements for Mutex benchmarks, especially in contended cases:

```
measure target: real

<table>
<thead>
<tr>
<th>name</th>
<th>trunk</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>0.149</td>
<td>0.148</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>0.893</td>
<td>0.651</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>0.809</td>
<td>0.624</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>2.608</td>
<td>0.628</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>28.227</td>
<td>0.881</td>
</tr>
</tbody>
</table>
```

Speedup ratio: compare with the result of `trunk` (greater is better)

```
measure target: built

<table>
<thead>
<tr>
<th>name</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>1.002</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>1.372</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>1.297</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>4.149</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>32.044</td>
</tr>
</tbody>
</table>
```

Tested on AMD FX-8320 8-core at 3.5GHz

Associated revisions

Revision 3586c9e0 - 05/08/2017 12:18 AM - normal
reduce rb_mutex_t size from 160 to 80 bytes on 64-bit

Instead of relying on a native condition variable and mutex for every Ruby Mutex object, use a doubly linked-list to implement a waiter queue in the Mutex. The immediate benefit of this is reducing the size of every Mutex object, as some projects have many objects requiring synchronization.

In the future, this technique using a linked-list and on-stack list node (struct mutex_waiter) should allow us to easily transition to M:N threading model, as we can avoid the native thread dependency to implement Mutex.
We already do something similar for autoload in variable.c, and this was inspired by the Linux kernel wait queue (as ccan/list is inspired by the Linux kernel linked-list).

Finally, there are big performance improvements for Mutex benchmarks, especially in contended cases:

measure target: real

<table>
<thead>
<tr>
<th>name</th>
<th>trunk</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_Whileloop2</td>
<td>0.149</td>
<td>0.148</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>0.893</td>
<td>0.651</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>0.809</td>
<td>0.624</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>2.608</td>
<td>0.628</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>28.227</td>
<td>0.881</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of 'trunk' (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_Whileloop2</td>
<td>1.002</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>1.372</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>1.297</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>4.149</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>32.044</td>
</tr>
</tbody>
</table>

Tested on AMD FX-8320 8-core at 3.5GHz

- thread.c (debug_deadlock_check): update for new struct (rb_check_deadlock): ditto [ruby-core:80913] [Feature #13517]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@58604 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 58604 - 05/08/2017 12:18 AM - normalperson (Eric Wong)

reduce rb_mutex_t size from 160 to 80 bytes on 64-bit

Instead of relying on a native condition variable and mutex for every Ruby Mutex object, use a doubly linked-list to implement a waiter queue in the Mutex. The immediate benefit of this is reducing the size of every Mutex object, as some projects have many objects requiring synchronization.

In the future, this technique using a linked-list and on-stack list node (struct mutex_waiter) should allow us to easily transition to M:N threading model, as we can avoid the native thread dependency to implement Mutex.

We already do something similar for autoload in variable.c, and this was inspired by the Linux kernel wait queue (as ccan/list is inspired by the Linux kernel linked-list).

Finally, there are big performance improvements for Mutex benchmarks, especially in contended cases:

measure target: real

<table>
<thead>
<tr>
<th>name</th>
<th>trunk</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_Whileloop2</td>
<td>0.149</td>
<td>0.148</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>0.893</td>
<td>0.651</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>0.809</td>
<td>0.624</td>
</tr>
<tr>
<td>name</td>
<td>trunk</td>
<td>built</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>2.608</td>
<td>0.628</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>28.227</td>
<td>0.881</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of 'trunk' (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>1.002</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>1.372</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>1.297</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>4.149</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>32.044</td>
</tr>
</tbody>
</table>

Tested on AMD FX-8320 8-core at 3.5GHz

- thread_sync.c (struct mutex_waiter): new on-stack struct (struct rb_mutex_struct): remove native lock/cond, use ccan/list
  (rb_mutex_num_waiting): new function for debug_deadlock_check (mutex_free): remove native _destroy (mutex_alloc): initialize waitq, remove
  native_initialize (rb_mutex_trylock): remove native_mutex(lock,unlock) (lock_func): remove (lock_interrupt): remove (rb_mutex_lock): rewrite
  waiting path to use native_sleep + ccan/list (rb_mutex_unlock_th): rewrite to wake up from native_sleep using rb_threadptr_interrupt
  (rb_mutex_abandon_all): empty waitq
- thread.c (debug_deadlock_check): update for new struct (rb_check_deadlock): ditto [ruby-core:80913] [Feature #13517]

Revision 58604 - 05/08/2017 12:18 AM - normal

reduce rb_mutex_t size from 160 to 80 bytes on 64-bit

Instead of relying on a native condition variable and mutex for every Ruby Mutex object, use a doubly linked-list to implement a
waiter queue in the Mutex. The immediate benefit of this is reducing the size of every Mutex object, as some projects have
many objects requiring synchronization.

In the future, this technique using a linked-list and on-stack list node (struct mutex_waiter) should allow us to easily
transition to M:N threading model, as we can avoid the native thread dependency to implement Mutex.

We already do something similar for autoload in variable.c, and this was inspired by the Linux kernel wait queue (as
ccan/list is inspired by the Linux kernel linked-list).

Finally, there are big performance improvements for Mutex benchmarks, especially in contended cases:

<table>
<thead>
<tr>
<th>name</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>0.149</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>0.893</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>0.809</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>2.608</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>28.227</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of 'trunk' (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>1.002</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>1.372</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>1.297</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>4.149</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>32.044</td>
</tr>
</tbody>
</table>

05/16/2020
Tested on AMD FX-8320 8-core at 3.5GHz

- thread_sync.c (struct mutex_waiter): new on-stack struct (struct rb_mutex_struct): remove native lock/cond, use ccan/list
  (rb_mutex_num_waiting): new function for debug_deadlock_check (mutex_free): remove native_destroy (mutex_alloc): initialize waitq, remove
  native_initialze (rb_mutex_trylock): remove native_mutex(lock,unlock) (lock_func): remove (lock_interrupt): remove (rb_mutex_lock): rewrite
  waiting path to use native_sleep + ccan/list (rb_mutex_unlock_th): rewrite to wake up from native_sleep using rb_threadptr_interrupt
  (rb_mutex_abandon_all): empty waitq
- thread.c (debug_deadlock_check): update for new struct (rb_check_deadlock): ditto [ruby-core:80913] [Feature #13517]

Revision 58604 - 05/08/2017 12:18 AM - normal
reduce rb_mutex_t size from 160 to 80 bytes on 64-bit

Instead of relying on a native condition variable and mutex for every Ruby Mutex object, use a doubly linked-list to implement a
waiter queue in the Mutex. The immediate benefit of this is reducing the size of every Mutex object, as some projects have
many objects requiring synchronization.

In the future, this technique using a linked-list and on-stack
list node (struct mutex_waiter) should allow us to easily
transition to M:N threading model, as we can avoid the native
thread dependency to implement Mutex.

We already do something similar for autoload in variable.c,
and this was inspired by the Linux kernel wait queue (as ccan/list is inspired by the Linux kernel linked-list).

Finally, there are big performance improvements for Mutex
benchmarks, especially in contended cases:

measure target: real

<table>
<thead>
<tr>
<th>name</th>
<th>trunk</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>0.149</td>
<td>0.148</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>0.893</td>
<td>0.651</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>0.809</td>
<td>0.624</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>2.608</td>
<td>0.628</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>28.227</td>
<td>0.881</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of ‘trunk’ (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>1.002</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>1.372</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>1.297</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>4.149</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>32.044</td>
</tr>
</tbody>
</table>

Tested on AMD FX-8320 8-core at 3.5GHz

- thread_sync.c (struct mutex_waiter): new on-stack struct (struct rb_mutex_struct): remove native lock/cond, use ccan/list
  (rb_mutex_num_waiting): new function for debug_deadlock_check (mutex_free): remove native_destroy (mutex_alloc): initialize waitq, remove
  native_initialze (rb_mutex_trylock): remove native_mutex(lock,unlock) (lock_func): remove (lock_interrupt): remove (rb_mutex_lock): rewrite
  waiting path to use native_sleep + ccan/list (rb_mutex_unlock_th): rewrite to wake up from native_sleep using rb_threadptr_interrupt
  (rb_mutex_abandon_all): empty waitq
- thread.c (debug_deadlock_check): update for new struct (rb_check_deadlock): ditto [ruby-core:80913] [Feature #13517]

Revision 129a0711 - 07/07/2017 05:59 PM - normal
NEWS: note [Feature #13517] is Linux-only (no side-effects on _nonblock)
git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@59284 b2dd03c8-39d4-4d6f-98ff-823fe69b080e
I'm slightly worried about some external code subclassing ConditionVariable, Queue, and SizedQueue and relying on them being Structs. However, they only started being Structs with Ruby 2.1, and were implemented in pure Ruby before that; so hopefully nobody notices that implementation detail.

Also, note the Mutex change as it may affect program design when space can be saved.

- NEWS: entries for [Feature #13552] and [Feature #13517]

History
For who care about 32-bit, single-core x86, here are my Pentium M (Centrino) @ 1.6GHz numbers:

Size reduction of Mutex on 32-bit is 112 => 40 bytes

minimum results in each 3 measurements.
Execution time (sec)
name             trunk   built
loop_whileloop2  0.554   0.554
vm2_mutex*       3.136   2.217
vm_thread_mutex1 2.783   2.186
vm_thread_mutex2 2.907   2.174
vm_thread_mutex3 9.740   2.586

Speedup ratio: compare with the result of `trunk' (greater is better)
name             built
loop_whileloop2  0.999
vm2_mutex*       1.414
vm_thread_mutex1 1.273
vm_thread_mutex2 1.337
vm_thread_mutex3 3.766

In the future, I think the cond_waiting flag can be moved into a FL_USER flag, too.

But I also want to try similar changes to avoid Array usage in Queue, SizedQueue, and ConditionVariable classes and rely on ccan/list + stack for waiters. I will convert from T_STRUCT to T_DATA.

Any comment? I would like to commit this, soon.

Thanks.

At a glance, it seems nice.
But I need time to check deeply.
I'll check with 'Misc #13514'.

Please wait these days. In Japan, now we have holiday week. I'll check on these days.

Thanks,
Koichi

I have no objection about this patch. thank you.

one question.

```c
list_for_each_safe(&mutex->waitq, cur, next, node) {
  list_del_init(&cur->node);
  switch (cur->th->state) {
    case THREAD_KILLED:
      continue;
    case THREAD_STOPPED:
    case THREAD_RUNNABLE:
    case THREAD_STOPPED_FOREVER:
      rb_threadptr_interrupt (cur->th);
```
rb_mutex_lock() set th->status as THREAD_STOPPED_FOREVER before native sleep, but the above code quoted from rb_mutex_unlock_th().

What kind of situation do you assume when the thread status is other than THREAD_STOPPED_FOREVER?

Thanks,
Koichi

--
// SASADA Koichi at atdot dot net

#5 - 05/07/2017 11:11 PM - normalperson (Eric Wong)

SASADA Koichi ko1@atdot.net wrote:

sorry for late response.
I have no objection about this patch. thank you.

one question.

list_for_each_safe(&mutex->waitq, cur, next, node) {
  list_del_init(&cur->node);
  switch (cur->th->state) {
  case THREAD_KILLED:
    continue;
  case THREAD_STOPPED:
  case THREAD_RUNNABLE:
  case THREAD_STOPPED_FOREVER:
    rb_threadptr_interrupt(cur->th);
    goto found;
  }
}

Oops, that should be status, not state:

switch (cur->th->status) {
  case THREAD_KILLED:
    continue;
  case THREAD_STOPPED:
  case THREAD_RUNNABLE:
  case THREAD_STOPPED_FOREVER:
    rb_threadptr_interrupt(cur->th);
    goto found;
}

`rb_mutex_lock()` set `th->status` as `THREAD_STOPPED_FOREVER` before native sleep, but the above code quoted from `rb_mutex_unlock_th()`.

What kind of situation do you assume when the thread status is other than `THREAD_STOPPED_FOREVER`?

Back to your original question. THREAD_RUNNABLE is possible if somebody uses Thread#run:

require 'thread'
m = Mutex.new
th = Thread.new do
  sleep 0.1 # wait for main thread to get lock
  m.synchronize do
    sleep
  end
end

m.synchronize do
  sleep 0.2 # wait for th to block on m.synchronize
  th.run
end

I am not sure about other statuses. Maybe exit/GC can trigger THREAD_KILLED, the mutex_free->rb_mutex_unlock_th call chain looks like it might due to GC ordering. Anyways, I will add comments here when I commit.

Thanks,
Thank you for the review!

#6 - 05/07/2017 11:41 PM - ko1 (Koichi Sasada)
On 2017/05/08 8:08, Eric Wong wrote:

Back to your original question. THREAD_RUNNABLE is possible if somebody uses Thread#run:

```ruby
require 'thread'
m = Mutex.new
th = Thread.new do
  sleep 0.1 # wait for main thread to get lock
  m.synchronize do
    sleep
  end
end

m.synchronize do
  sleep 0.2 # wait for th to block on m.synchronize
  th.run
end
```

I also confirm that this code set THREAD_RUNNABLE. However, th waits locking forever, current Thread#run should be bug. mmmm. But not so serious because it is only small period (maybe as you know). We should modify later.

I am not sure about other statuses. Maybe exit/GC can trigger THREAD_KILLED, the mutex_free->rb_mutex_unlock_th call chain looks like it might due to GC ordering. Anyways, I will add comments here when I commit.

I think adding rb_bug[] guard is good to know the flow of such situation.

--
// SASADA Koichi at atdot dot net

#7 - 05/08/2017 12:18 AM - Anonymous
- Status changed from Open to Closed

Applied in changeset trunk/r58604.

reduce rb_mutex_t size from 160 to 80 bytes on 64-bit

Instead of relying on a native condition variable and mutex for every Ruby Mutex object, use a doubly linked-list to implement a waiter queue in the Mutex. The immediate benefit of this is reducing the size of every Mutex object, as some projects have many objects requiring synchronization.

In the future, this technique using a linked-list and on-stack list node (struct mutex_waiter) should allow us to easily transition to M:N threading model, as we can avoid the native thread dependency to implement Mutex.

We already do something similar for autoload in variable.c, and this was inspired by the Linux kernel wait queue (as ccan/list is inspired by the Linux kernel linked-list).

Finally, there are big performance improvements for Mutex benchmarks, especially in contended cases:

```ruby
measure target: real
```
<table>
<thead>
<tr>
<th>name</th>
<th>trunk</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>0.149</td>
<td>0.148</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>0.893</td>
<td>0.651</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>0.809</td>
<td>0.624</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>2.608</td>
<td>0.628</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>28.227</td>
<td>0.881</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of `trunk` (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>1.002</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>1.372</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>1.297</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>4.149</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>32.044</td>
</tr>
</tbody>
</table>

Tested on AMD FX-8320 8-core at 3.5GHz

- thread_sync.c (struct mutex_waiter): new on-stack struct (struct rb_mutex_struct): remove native lock/cond, use ccan/list
  (rb_mutex_num_waiting): new function for debug_deadlock_check (mutex_free): remove native_destroy (mutex_alloc): initialize waitq, remove nativeinitialize (rb_mutex_trylock): remove native_mutex(lock,unlock) (lock_func): remove (lock_interrupt): remove (rb_mutex_lock): rewrite waiting path to use native_sleep + ccan/list (rb_mutex_unlock_th): rewrite to wake up from native_sleep using rb_threadptr_interrupt
  (rb_mutex_abandon_all): empty waitq
- thread.c (debug_deadlock_check): update for new struct (rb_check_deadlock): ditto [ruby-core:80913] [Feature #13517]

Files

0001-reduce-rb_mutex_t-size-from-160-to-80-bytes-on-64-bi.patch 9.17 KB 04/28/2017 normalperson (Eric Wong)