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
name       | trunk | built
-----------|-------|------:
loop_whileloop2 | 0.149| 0.148
vm2_mutex*  | 0.893| 0.651
vm_thread_mutex1| 0.809| 0.624
vm_thread_mutex2| 2.608| 0.628
vm_thread_mutex3| 28.227| 0.881
```

Speedup ratio: compare with the result of 'trunk' (greater is better)
```
name       | built
-----------|------:
loop_whileloop2 | 1.002
vm2_mutex*  | 1.372
vm_thread_mutex1| 1.297
vm_thread_mutex2| 4.149
vm_thread_mutex3| 32.044
```

Tested on AMD FX-8320 8-core at 3.5GHz
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>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>2.609</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

[Feature #13517]

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:
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

**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>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>

<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>

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

<table>
<thead>
<tr>
<th>name</th>
<th>trunk</th>
<th>built</th>
</tr>
</thead>
<tbody>
<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>

### 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]

**Revision 59604 - 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:

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

<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>0.624</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>28.227</td>
<td>0.881</td>
</tr>
</tbody>
</table>
name | built
--- | ---
loop_whileloop2 | 1.002
vm2_mutex* | 1.372
vm_thread_mutex1 | 1.297
vm_thread_mutex2 | 4.149
vm_thread_mutex3 | 32.044

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
  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 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 59284 - 07/07/2017 05:59 PM - normalperson (Eric Wong)
NEWS: note [Feature #13517] is Linux-only (no side-effects on _*nonblock)

Revision 59284 - 07/07/2017 05:59 PM - normal
NEWS: note [Feature #13517] is Linux-only (no side-effects on _*nonblock)

Revision 59284 - 07/07/2017 05:59 PM - normal
NEWS: note [Feature #13517] is Linux-only (no side-effects on _*nonblock)

Revision 12bf0c00 - 07/21/2017 07:06 PM - normal
NEWS: add entries for thread_sync.c changes
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]
git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@59385 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 59385 - 07/21/2017 07:06 PM - normalperson (Eric Wong)
NEWS: add entries for thread_sync.c changes
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

08/26/2022
when space can be saved.

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

**Revision 59385 - 07/21/2017 07:06 PM - normal**

NEWS: add entries for thread_sync.c changes

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]

**Revision 59385 - 07/21/2017 07:06 PM - normal**

NEWS: add entries for thread_sync.c changes

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**

#1 - 04/28/2017 08:08 AM - normalperson (Eric Wong)
normalperson@yhbt.net wrote:

https://bugs.ruby-lang.org/issues/13517

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)

<table>
<thead>
<tr>
<th>name</th>
<th>trunk</th>
<th>built</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_whileloop2</td>
<td>0.554</td>
<td>0.554</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>3.136</td>
<td>2.217</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>2.783</td>
<td>2.186</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>2.907</td>
<td>2.174</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>9.740</td>
<td>2.586</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>0.999</td>
</tr>
<tr>
<td>vm2_mutex*</td>
<td>1.414</td>
</tr>
<tr>
<td>vm_thread_mutex1</td>
<td>1.273</td>
</tr>
<tr>
<td>vm_thread_mutex2</td>
<td>1.337</td>
</tr>
<tr>
<td>vm_thread_mutex3</td>
<td>3.766</td>
</tr>
</tbody>
</table>

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.

#2 - 05/02/2017 09:12 AM - normalperson (Eric Wong)
normalperson@yhbt.net wrote:
Any comment? I would like to commit this, soon.

Thanks.

#3 - 05/02/2017 09:21 AM - ko1 (Koichi Sasada)
At a glance, it seems nice.
But I need to 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

#4 - 05/07/2017 07:08 PM - ko1 (Koichi Sasada)
sorry for late response.
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);
            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?

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.

```

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);
```

// 08/26/2022
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:

```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 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,
Koichi

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.
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>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]
<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001-reduce-rb_mutex_t-size-from-160-to-80-bytes-on-64-bi.patch</td>
<td>9.17 KB</td>
<td>04/28/2017</td>
<td>normalperson (Eric Wong)</td>
</tr>
</tbody>
</table>