

## Ruby trunk - Feature #9632

### [PATCH 0/2] speedup IO#close with linked-list from ccan

03/13/2014 03:26 AM - normalperson (Eric Wong)

<b>Status:</b>	Closed
<b>Priority:</b>	Normal
<b>Assignee:</b>	ko1 (Koichi Sasada)
<b>Target version:</b>	2.2.0
<b>Description</b>	
<p>This imports the ccan linked-list (BSD-MIT licensed version of the Linux kernel linked list). I cut out some of the unused str* code (only for debugging), but it's still a big import of new code. Modifications to existing code is minimal, and it makes the living_threads iteration functions simpler.</p> <p>The improvement is great, and there may be future places where we could use a doubly linked list.</p> <p>= vm-&gt;living_threads:</p> <ul style="list-style-type: none"><li>• before: st hash table had extra malloc overhead, and slow iteration due to bad cache locality</li><li>• after: guaranteed O(1) insert/remove performance (branchless!) iteration is still O(n), but performance is improved in IO#close due to less pointer chasing</li></ul> <p>= IO#close: further improvement with second linked list</p> <ul style="list-style-type: none"><li>• before: IO#close is linear based on number of living threads</li><li>• after: IO#close is linear based on number of waiting threads</li></ul> <p>No extra malloc is needed (only 2 new pointers in existing structs) for a secondary linked-list for waiting FDs.</p> <p>I chose the ccan linked list over BSD for two reasons:</p> <ol style="list-style-type: none"><li>1) insertion and removal are both branchless</li><li>2) locality is improved if a struct may be a member of multiple lists</li></ol> <p>git://80x24.org/ruby.git threads-list</p>	

#### Associated revisions

##### Revision 508091d9 - 05/20/2017 09:47 AM - normal

speed up IO#close with many threads

Today, it increases IO#close performance with many threads:

Execution time (sec)

name	trunk	after
vm_thread_close	4.276	3.018

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

name	after
vm_thread_close	1.417

This speedup comes because rb\_notify\_fd\_close only scans threads inside rb\_thread\_io\_blocking\_region, not all threads in the VM.

In the future, this type data structure may allow us to notify waiters of multiple FDs on a single thread (when using Fibers).

- thread.c (struct waiting\_fd): declare (rb\_thread\_io\_blocking\_region): use on-stack list waiter (rb\_notify\_fd\_close): walk vm->waiting\_fds instead (call\_without\_gvl): remove old field setting (th\_init): ditto

- vm\_core.h (typedef struct rb\_vm\_struct): add waiting\_fds list
- (typedef struct rb\_thread\_struct): remove waiting\_fd field (rb\_vm\_living\_threads\_init): initialize waiting\_fds list

I am now kicking myself for not thinking about this 3 years ago when I introduced ccan/list in [Feature #9632] to optimize this same function :<

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@58812 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

**Revision 58812 - 05/20/2017 09:47 AM - normalperson (Eric Wong)**

speed up IO#close with many threads

Today, it increases IO#close performance with many threads:

Execution time (sec)  
 name trunk after  
 vm\_thread\_close 4.276 3.018

Speedup ratio: compare with the result of `trunk` (greater is better)  
 name after  
 vm\_thread\_close 1.417

This speedup comes because rb\_notify\_fd\_close only scans threads inside rb\_thread\_io\_blocking\_region, not all threads in the VM.

In the future, this type data structure may allow us to notify waiters of multiple FDs on a single thread (when using Fibers).

- thread.c (struct waiting\_fd): declare (rb\_thread\_io\_blocking\_region): use on-stack list waiter (rb\_notify\_fd\_close): walk vm->waiting\_fds instead (call\_without\_gvl): remove old field setting (th\_init): ditto
- vm\_core.h (typedef struct rb\_vm\_struct): add waiting\_fds list
- (typedef struct rb\_thread\_struct): remove waiting\_fd field (rb\_vm\_living\_threads\_init): initialize waiting\_fds list

I am now kicking myself for not thinking about this 3 years ago when I introduced ccan/list in [Feature #9632] to optimize this same function :<

**Revision 58812 - 05/20/2017 09:47 AM - normal**

speed up IO#close with many threads

Today, it increases IO#close performance with many threads:

Execution time (sec)  
 name trunk after  
 vm\_thread\_close 4.276 3.018

Speedup ratio: compare with the result of `trunk` (greater is better)  
 name after  
 vm\_thread\_close 1.417

This speedup comes because rb\_notify\_fd\_close only scans threads inside rb\_thread\_io\_blocking\_region, not all threads in the VM.

In the future, this type data structure may allow us to notify waiters of multiple FDs on a single thread (when using Fibers).

- thread.c (struct waiting\_fd): declare (rb\_thread\_io\_blocking\_region): use on-stack list waiter (rb\_notify\_fd\_close): walk vm->waiting\_fds instead (call\_without\_gvl): remove old field setting (th\_init): ditto
- vm\_core.h (typedef struct rb\_vm\_struct): add waiting\_fds list
- (typedef struct rb\_thread\_struct): remove waiting\_fd field (rb\_vm\_living\_threads\_init): initialize waiting\_fds list

I am now kicking myself for not thinking about this 3 years ago when I introduced ccan/list in [Feature #9632] to optimize this same function :<

**Revision 58812 - 05/20/2017 09:47 AM - normal**

speed up IO#close with many threads

Today, it increases IO#close performance with many threads:

Execution time (sec)  
name trunk after  
vm\_thread\_close 4.276 3.018

Speedup ratio: compare with the result of `trunk` (greater is better)  
name after  
vm\_thread\_close 1.417

This speedup comes because `rb_notify_fd_close` only scans threads inside `rb_thread_io_blocking_region`, not all threads in the VM.

In the future, this type data structure may allow us to notify waiters of multiple FDs on a single thread (when using Fibers).

- `thread.c` (struct `waiting_fd`): declare (`rb_thread_io_blocking_region`): use on-stack list waiter (`rb_notify_fd_close`): walk `vm->waiting_fds` instead (`call_without_gvl`): remove old field setting (`th_init`): ditto
- `vm_core.h` (typedef struct `rb_vm_struct`): add `waiting_fds` list
- (typedef struct `rb_thread_struct`): remove `waiting_fd` field (`rb_vm_living_threads_init`): initialize `waiting_fds` list

I am now kicking myself for not thinking about this 3 years ago when I introduced `cscan/list` in [Feature #9632] to optimize this same function :<

### Revision 651990fa - 07/07/2017 02:03 AM - usa (Usaku NAKAMURA)

This backport of r58812 is necessary to ease backporting r59028, which fixes a real bug.

- `thread.c` (struct `waiting_fd`): declare (`rb_thread_io_blocking_region`): use on-stack list waiter (`rb_notify_fd_close`): walk `vm->waiting_fds` instead (`call_without_gvl`): remove old field setting (`th_init`): ditto [Feature #9632]
- `vm_core.h` (typedef struct `rb_vm_struct`): add `waiting_fds` list
- (typedef struct `rb_thread_struct`): remove `waiting_fd` field (`rb_vm_living_threads_init`): initialize `waiting_fds` list

This should fix bad interactions with `test_race_gets_and_close` in `test/ruby/test_io.rb` since we ensure `rb_notify_fd_close` continues returning the busy flag after enqueueing the interrupt.

- `thread.c` (`rb_notify_fd_close`): do not enqueue multiple interrupts [ruby-core:81581] [Bug #13632]
- `test/ruby/test_io.rb` (`test_single_exception_on_close`): new test based on script from Nikolay

git-svn-id: [svn+ssh://ci.ruby-lang.org/ruby/branches/ruby\\_2\\_3@59274](https://ci.ruby-lang.org/ruby/branches/ruby_2_3@59274) b2dd03c8-39d4-4d8f-98ff-823fe69b080e

### Revision 59274 - 07/07/2017 02:03 AM - usa (Usaku NAKAMURA)

This backport of r58812 is necessary to ease backporting r59028, which fixes a real bug.

- `thread.c` (struct `waiting_fd`): declare (`rb_thread_io_blocking_region`): use on-stack list waiter (`rb_notify_fd_close`): walk `vm->waiting_fds` instead (`call_without_gvl`): remove old field setting (`th_init`): ditto [Feature #9632]
- `vm_core.h` (typedef struct `rb_vm_struct`): add `waiting_fds` list
- (typedef struct `rb_thread_struct`): remove `waiting_fd` field (`rb_vm_living_threads_init`): initialize `waiting_fds` list

This should fix bad interactions with `test_race_gets_and_close` in `test/ruby/test_io.rb` since we ensure `rb_notify_fd_close`

continues returning the busy flag after enqueueing the interrupt.

- `thread.c (rb_notify_fd_close)`: do not enqueue multiple interrupts  
[ruby-core:81581] [Bug #13632]
- `test/ruby/test_io.rb (test_single_exception_on_close)`:  
new test based on script from Nikolay

#### Revision 27251312 - 07/08/2017 02:21 AM - nagachika (Tomoyuki Chikanaga)

merge revision(s) 58284,58812,59028: [Backport #13632]

`vm_core.h: ruby_error_stream_closed`

```
* vm_core.h (ruby_special_exceptions): renamed
  ruby_error_closed_stream as ruby_error_stream_closed, like the
  message.
speed up IO#close with many threads
```

Today, it increases IO#close performance with many threads:

Execution time (sec)		
name	trunk	after
vm_thread_close	4.276	3.018

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

name	after
vm_thread_close	1.417

This speedup comes because `rb_notify_fd_close` only scans threads inside `rb_thread_io_blocking_region`, not all threads in the VM.

In the future, this type data structure may allow us to notify waiters of multiple FDs on a single thread (when using Fibers).

```
* thread.c (struct waiting_fd): declare
  (rb_thread_io_blocking_region): use on-stack list waiter
  (rb_notify_fd_close): walk vm->waiting_fds instead
  (call_without_gvl): remove old field setting
  (th_init): ditto
* vm_core.h (typedef struct rb_vm_struct): add waiting_fds list
* (typedef struct rb_thread_struct): remove waiting_fd field
  (rb_vm_living_threads_init): initialize waiting_fds list
```

I am now kicking myself for not thinking about this 3 years ago when I introduced `ccan/list` in [Feature #9632] to optimize this same function :<

IO#close: do not enqueue redundant interrupts (take #2)

Enqueueing multiple errors for one event causes spurious errors down the line, as reported by Nikolay Vashchenko in <https://bugs.ruby-lang.org/issues/13632>

This should fix bad interactions with `test_race_gets_and_close` in `test/ruby/test_io.rb` since we ensure `rb_notify_fd_close` continues returning the busy flag after enqueueing the interrupt.

Backporting changes to 2.4 and earlier releases will be more challenging...

```
* thread.c (rb_notify_fd_close): do not enqueue multiple interrupts
  [ruby-core:81581] [Bug #13632]
* test/ruby/test_io.rb (test_single_exception_on_close):
  new test based on script from Nikolay
```

git-svn-id: [svn+ssh://ci.ruby-lang.org/ruby/branches/ruby\\_2\\_4@59286](https://ci.ruby-lang.org/ruby/branches/ruby_2_4@59286) b2dd03c8-39d4-4d8f-98ff-823fe69b080e

#### Revision 59286 - 07/08/2017 02:21 AM - nagachika (Tomoyuki Chikanaga)

merge revision(s) 58284,58812,59028: [Backport #13632]

`vm_core.h: ruby_error_stream_closed`

```
* vm_core.h (ruby_special_exceptions): renamed
  ruby_error_closed_stream as ruby_error_stream_closed, like the
  message.
speed up IO#close with many threads
```

Today, it increases IO#close performance with many threads:

```
Execution time (sec)
name            trunk    after
vm_thread_close 4.276    3.018
```

```
Speedup ratio: compare with the result of `trunk' (greater is better)
name            after
vm_thread_close 1.417
```

This speedup comes because rb\_notify\_fd\_close only scans threads inside rb\_thread\_io\_blocking\_region, not all threads in the VM.

In the future, this type data structure may allow us to notify waiters of multiple FDs on a single thread (when using Fibers).

```
* thread.c (struct waiting_fd): declare
  (rb_thread_io_blocking_region): use on-stack list waiter
  (rb_notify_fd_close): walk vm->waiting_fds instead
  (call_without_gvl): remove old field setting
  (th_init): ditto
* vm_core.h (typedef struct rb_vm_struct): add waiting_fds list
* (typedef struct rb_thread_struct): remove waiting_fd field
  (rb_vm_living_threads_init): initialize waiting_fds list
```

I am now kicking myself for not thinking about this 3 years ago when I introduced ccan/list in [Feature #9632] to optimize this same function :<

```
IO#close: do not enqueue redundant interrupts (take #2)
```

Enqueuing multiple errors for one event causes spurious errors down the line, as reported by Nikolay Vashchenko in <https://bugs.ruby-lang.org/issues/13632>

This should fix bad interactions with test\_race\_gets\_and\_close in test/ruby/test\_io.rb since we ensure rb\_notify\_fd\_close continues returning the busy flag after enqueueing the interrupt.

Backporting changes to 2.4 and earlier releases will be more challenging...

```
* thread.c (rb_notify_fd_close): do not enqueue multiple interrupts
  [ruby-core:81581] [Bug #13632]
* test/ruby/test_io.rb (test_single_exception_on_close):
  new test based on script from Nikolay
```

## History

---

### #1 - 03/19/2014 08:28 AM - normalperson (Eric Wong)

[normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

0001-doubly-linked-list-from-ccan-to-manage-vm-living\_thr.patch (68.1 KB)

I'll de-duplicate the CCO declaration files if allowed to commit this. The original had symlinks, but I assume symlinks are not allowed in this source tree for portability.

I really like the Linux-kernel-style of linked-list.

### #2 - 03/30/2014 03:28 AM - normalperson (Eric Wong)

Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

[normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

0001-doubly-linked-list-from-ccan-to-manage-vm-living\_thr.patch (68.1 KB)

I'll de-duplicate the CC0 declaration files if allowed to commit this.  
The original had symlinks, but I assume symlinks are not allowed in this source tree for portability.

Updated 0001 patch with deduplicated license files:  
<http://bogomips.org/ruby.git/patch?id=b5401cdc6f72>

I also renamed CCAN\_INCLUDES to CCAN\_LIST\_INCLUDES in common.mk; in case we import other modules from ccan[1].

[1] - <http://ccodearchive.net/>

**#3 - 04/05/2014 11:38 PM - normalperson (Eric Wong)**

[normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Updated 0001 patch with deduplicated license files:  
<http://bogomips.org/ruby.git/patch?id=b5401cdc6f72>

Any comment? My main concern is it's a large import of new code; but it is also highly reusable. I'll commit in 2-4 weeks if no response. The 0002 patch can wait longer.

**#4 - 05/10/2014 11:58 PM - normalperson (Eric Wong)**

Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Any comment? My main concern is it's a large import of new code; but it is also highly reusable. I'll commit in 2-4 weeks if no response. The 0002 patch can wait longer.

Committed as r45913. Hopefully nothing breaks, I tested extensively on my "production" server. Sorry for the delay, was busy.

**#5 - 05/11/2014 10:07 AM - ko1 (Koichi Sasada)**

Sorry for late response.

Just curious (I'm not against of this change).

1. How performance improved?
2. Should we modify ccan/\* files? Or should we sync with originals?
3. What mean the name "CCAN"?

**#6 - 05/11/2014 10:08 AM - ko1 (Koichi Sasada)**

1. Should we use it on compile.c?

**#7 - 05/11/2014 10:58 AM - normalperson (Eric Wong)**

[ko1@atdot.net](mailto:ko1@atdot.net) wrote:

1. How performance improved?

There is less pointer chasing for iteration:

Before: `st_table_entry->rb_thread_t->st_table_entry->rb_thread_t ...`  
After: `rb_thread->rb_thread ...`

This is made possible by the `container_of` macro.

I plan to use `container_of` in method/constant/symbol table, too (ihash in Feature [#9614](#)).

1. Should we modify ccan/\* files? Or should we sync with originals?

I probably best to sync with originals. I removed parts of `ccan/str/str.h` we are not using, but we can use more of `str.h` later.

I may also put ihash in CCAN so other projects may use it easily.  
But I am not sure about the name "ihash".

1. What mean the name "CCAN"?

Comprehensive C Archive Network - ccodearchive.net

1. Should we use it on compile.c?

Maybe. I do not know compile.c well enough...  
If we can reduce allocations and pointer chasing without regressions,  
we should use it.

**#8 - 05/11/2014 11:09 AM - normalperson (Eric Wong)**

Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Before: st\_table\_entry->rb\_thread\_t->st\_table\_entry->rb\_thread\_t ...

Sorry, bad picture for Before, this is more accurate:

```
st_table_entry -> st_table_entry -> st_table_entry
  |               |               |
  v               v               v
rb_thread_t     rb_thread_t     rb_thread_t
```

**#9 - 05/13/2014 07:08 AM - akr (Akira Tanaka)**

2014-05-11 8:50 GMT+09:00 Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net):

Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Any comment? My main concern is it's a large import of new code;  
but it is also highly reusable. I'll commit in 2-4 weeks if no response.  
The 0002 patch can wait longer.

Committed as r45913. Hopefully nothing breaks, I tested extensively  
on my "production" server. Sorry for the delay, was busy.

I found that doxygen produces many warnings in ccan/ directory.  
<http://www.rubyist.net/~akr/chkbuild/debian/ruby-trunk/log/20140510T235500Z.diff.html.gz>

It seems the comments in ccan/ directory is not doxygen-compatible.

Anyone use doxygen?  
If no one use it, we can drop doxygen support.  
(It makes the CI faster.)  
--  
Tanaka Akira

**#10 - 05/13/2014 07:38 AM - normalperson (Eric Wong)**

Tanaka Akira [akr@fsij.org](mailto:akr@fsij.org) wrote:

I found that doxygen produces many warnings in ccan/ directory.  
<http://www.rubyist.net/~akr/chkbuild/debian/ruby-trunk/log/20140510T235500Z.diff.html.gz>

It seems the comments in ccan/ directory is not doxygen-compatible.

Sorry about that.

Anyone use doxygen?  
If no one use it, we can drop doxygen support.  
(It makes the CI faster.)

I do not use it.

We may also fix the comments to be doxygen-compatible and send patches

upstream to ccan. But if nobody uses doxygen, we save time by dropping it.

**#11 - 05/13/2014 07:38 AM - nobu (Nobuyoshi Nakada)**

(2014/05/13 16:29), Eric Wong wrote:

Tanaka Akira [akr@fsij.org](mailto:akr@fsij.org) wrote:

Anyone use doxygen?  
If no one use it, we can drop doxygen support.  
(It makes the CI faster.)

I do not use it.

I don't use it too (it's too time consuming)

We may also fix the comments to be doxygen-compatible and send patches upstream to ccan. But if nobody uses doxygen, we save time by dropping it.

Or adding ccan to EXCLUDE in template/Doxyfile.tmpl.

**#12 - 09/13/2014 11:58 PM - normalperson (Eric Wong)**

[ko1@atdot.net](mailto:ko1@atdot.net) wrote:

1. Should we use it on compile.c?

Yes, and probably gc.c, too. I think it would help improve readability and remove some branches in our current code.

I have submitted patches for list\_add\_after, list\_add\_before and list\_swap functions:  
<https://lists.ozlabs.org/pipermail/ccan/2014-September/thread.html>

I think this will be next year for Ruby 2.3.

**#13 - 05/20/2017 09:47 AM - Anonymous**

- Status changed from Open to Closed

Applied in changeset [trunk|r58812](#).

---

speed up IO#close with many threads

Today, it increases IO#close performance with many threads:

Execution time (sec)  
name trunk after  
vm\_thread\_close 4.276 3.018

Speedup ratio: compare with the result of `trunk` (greater is better)  
name after  
vm\_thread\_close 1.417

This speedup comes because rb\_notify\_fd\_close only scans threads inside rb\_thread\_io\_blocking\_region, not all threads in the VM.

In the future, this type data structure may allow us to notify waiters of multiple FDs on a single thread (when using Fibers).

- thread.c (struct waiting\_fd): declare (rb\_thread\_io\_blocking\_region): use on-stack list waiter (rb\_notify\_fd\_close): walk vm->waiting\_fds instead (call\_without\_gvl): remove old field setting (th\_init): ditto
- vm\_core.h (typedef struct rb\_vm\_struct): add waiting\_fds list
- (typedef struct rb\_thread\_struct): remove waiting\_fd field (rb\_vm\_living\_threads\_init): initialize waiting\_fds list

I am now kicking myself for not thinking about this 3 years ago when I introduced ccan/list in [Feature [#9632](#)] to optimize this



same function :<

## Files

---

0002-speedup-IO-close-with-many-living-threads.patch	2.86 KB	03/13/2014	normalperson (Eric Wong)
0001-doubly-linked-list-from-ccan-to-manage-vm-living_thr.patch	68.1 KB	03/13/2014	normalperson (Eric Wong)