Ruby - Misc #15014

thread.c: use rb_hrtime_scalar for high-resolution time operations

08/20/2018 11:29 PM - normalperson (Eric Wong)

Status:	Closed	
Priority:	Normal	
Assignee:		

Description

thread.c: use rb_hrtime_t scalar for high-resolution time operations

Relying on "struct timespec" was too annoying API-wise and
used more stack space. "double" was a bit wacky w.r.t rounding
in the past, so now we'll switch to using a 64-bit type.

Unsigned 64-bit integer is able to give us over nearly 585
years of range with nanoseconds. This range is good enough
for the Linux kernel internal time representation, so it
ought to be good enough for us.

This reduces the stack usage of functions while GVL is held
(and thus subject to marking) on x86-64 Linux (with ppoll):

```
rb_wait_for_single_fd    120 => 104
do_select                120 => 88
```

I plan on using this for Timeout-in-VM [Feature #14859] and
auto-fiber [Feature #13618].

Associated revisions

Revision b0253a7569ffbf58fd3a61500aacd7369cce36dd - 08/25/2018 06:58 AM - Eric Wong

thread.c: use rb_hrtime_t scalar for high-resolution time operations

Relying on "struct timespec" was too annoying API-wise and
used more stack space. "double" was a bit wacky w.r.t rounding
in the past, so now we'll switch to using a 64-bit type.

Unsigned 64-bit integer is able to give us over nearly 585
years of range with nanoseconds. This range is good enough
for the Linux kernel internal time representation, so it
ought to be good enough for us.

This reduces the stack usage of functions while GVL is held
(and thus subject to marking) on x86-64 Linux (with ppoll):

```
    rb_wait_for_single_fd    120 => 104
    do_select                120 => 88
```

[ruby-core:88582] [Misc #15014]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64533 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision b0253a7569ffbf58fd3a61500aacd7369cce36dd - 08/25/2018 06:58 AM - Eric Wong

thread.c: use rb_hrtime_t scalar for high-resolution time operations

Relying on "struct timespec" was too annoying API-wise and
used more stack space. "double" was a bit wacky w.r.t rounding
in the past, so now we'll switch to using a 64-bit type.

Unsigned 64-bit integer is able to give us over nearly 585
years of range with nanoseconds. This range is good enough
for the Linux kernel internal time representation, so it
ought to be good enough for us.

This reduces the stack usage of functions while GVL is held
(and thus subject to marking) on x86-64 Linux (with ppoll):

```
    rb_wait_for_single_fd    120 => 104
    do_select                120 => 88
```

[ruby-core:88582] [Misc #15014]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64533 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision b0253a75 - 08/25/2018 06:58 AM - Eric Wong

thread.c: use rb_hrtime_t scalar for high-resolution time operations

Relying on "struct timespec" was too annoying API-wise and
used more stack space. "double" was a bit wacky w.r.t rounding
in the past, so now we'll switch to using a 64-bit type.

Unsigned 64-bit integer is able to give us over nearly 585
years of range with nanoseconds. This range is good enough
for the Linux kernel internal time representation, so it
ought to be good enough for us.

This reduces the stack usage of functions while GVL is held
(and thus subject to marking) on x86-64 Linux (with ppoll):

```
    rb_wait_for_single_fd    120 => 104
    do_select                120 => 88
```

[ruby-core:88582] [Misc #15014]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64533 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 95abea43fe6538c5e1d179f9b296c76c97652c31 - 08/25/2018 09:02 AM - Eric Wong

hrtime.h: add documentation

I updated the patch with documentation but forgot about it,
earlier :x

[ruby-core:88616] [Misc #15014]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64535 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 95abea43fe6538c5e1d179f9b296c76c97652c31 - 08/25/2018 09:02 AM - Eric Wong

hrtime.h: add documentation

I updated the patch with documentation but forgot about it,
earlier :x

[ruby-core:88616] [Misc #15014]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64535 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 95abea43 - 08/25/2018 09:02 AM - Eric Wong

hrtime.h: add documentation

I updated the patch with documentation but forgot about it,
earlier :x

[ruby-core:88616] [Misc #15014]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64535 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 08/21/2018 02:57 PM - MSP-Greg (Greg L)

@normalperson (Eric Wong) Eric,

At r64493, ruby-loco built successfully using the abiove patch, Thanks, Greg

#2 - 08/21/2018 07:52 PM - normalperson (Eric Wong)

Greg.mpls@gmail.com wrote:

> At r64493, ruby-loco built successfully using the abiove patch, Thanks, Greg

Thanks! I don't know when I'll commit this, yet.

#3 - 08/23/2018 07:16 AM - ko1 (Koichi Sasada)

Just a comment.
API references (a list of provided functions with a comment) at the beginning of the hrtime.h will help us.

#4 - 08/23/2018 10:42 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> Just a comment.
>
> API references (a list of provided functions with a comment) at the beginning of the hrtime.h will help us.

OK, updated:
https://80x24.org/spew/20180823103503.29516-1-e@80x24.org/raw

(Should have Thread::Coro (auto-fiber) update using that, soon,
so the benefit will become more apparent, but personal crap
keeps getting in the way of hacking :<)

#5 - 08/25/2018 06:58 AM - normalperson (Eric Wong)

- Status changed from Open to Closed

Applied in changeset trunk|r64533.

---

thread.c: use rb_hrtime_t scalar for high-resolution time operations

Relying on "struct timespec" was too annoying API-wise and
used more stack space. "double" was a bit wacky w.r.t rounding
in the past, so now we'll switch to using a 64-bit type.

Unsigned 64-bit integer is able to give us over nearly 585
years of range with nanoseconds. This range is good enough
for the Linux kernel internal time representation, so it
ought to be good enough for us.

This reduces the stack usage of functions while GVL is held
(and thus subject to marking) on x86-64 Linux (with ppoll):

```
rb_wait_for_single_fd    120 => 104
do_select                120 => 88
```

[ruby-core:88582] [Misc #15014]

#6 - 08/27/2018 06:55 AM - ko1 (Koichi Sasada)

Thanks!

> - Hi-res monotonic clock. It is currently nsec resolution, which has over 500 years of range (unsigned).

uint64?

> MY_RUBY_BUILD_MAY_TIME_TRAVEL

what is this?

#7 - 08/27/2018 07:22 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> Thanks!
>
> > - Hi-res monotonic clock. It is currently nsec resolution, which has over 500 years of range (unsigned).
>
> uint64?

You're welcome. Yep. Maybe some small systems can benefit with
usec/msec resolution and uint32_t, though.

However...

> > MY_RUBY_BUILD_MAY_TIME_TRAVEL
>
> what is this?

A joke; at least as far as we know it is a joke :>

#8 - 08/27/2018 07:42 AM - ko1 (Koichi Sasada)

On 2018/08/27 16:16, Eric Wong wrote:

> Yep. Maybe some small systems can benefit with
> usec/msec resolution and uint32_t, though.

My comment is, if it assumes uint64_t, it is worth to write it
explicitly (I verified this years with 2^64/ (1000 * 1000 * 1000 * 60 * 60 * 24 * 365) on google, but it will help for readers).

> > MY_RUBY_BUILD_MAY_TIME_TRAVEL
>
> what is this?

A joke; at least as far as we know it is a joke :>

I'm not sure why uint64_t doesn't solve time machine problem and
uint128_t solves it. Any RFC arguing it? (time machines should not leap
1.0790283e+22 years)?

Beside joking, I misunderstand that some systems can change the machine
time (time machine may means it) which affect monotonic time, and I
could not understand why uint128 can solve such system issue.

--
// SASADA Koichi at atdot dot net

#9 - 08/27/2018 09:22 AM - normalperson (Eric Wong)

Koichi Sasada ko1@atdot.net wrote:

> On 2018/08/27 16:16, Eric Wong wrote:
>
> > Yep. Maybe some small systems can benefit with
> > usec/msec resolution and uint32_t, though.
>
> My comment is, if it assumes uint64_t, it is worth to write it explicitly (I
> verified this years with 2^64/ (1000 * 1000 * 1000 * 60 * 60 * 24 * 365)
> on google, but it will help for readers).

OK, I made r64554

> > > MY_RUBY_BUILD_MAY_TIME_TRAVEL
>
> what is this?

A joke; at least as far as we know it is a joke :>

> I'm not sure why uint64_t doesn't solve time machine problem and uint128_t
> solves it. Any RFC arguing it? (time machines should not leap 1.0790283e+22
> years)?

It delays the onset of the problem. 585 years well within the
known range of human history; but 1.0790283e+22 years is much
farther away (probably beyond lifespan of the universe).

On the other hand, I don't know how a monotonic clock behaves
when a computer travels through time :>

> Beside joking, I misunderstand that some systems can change the machine time
> (time machine may means it) which affect monotonic time, and I could not
> understand why uint128 can solve such system issue.

Right, if monotonic time goes out of uint64_t range, things will
break. Maybe int128 can hide the problem. But I don't think
it's something we need to care about, though (OS would be
broken, I think).

#10 - 08/28/2018 02:04 AM - ko1 (Koichi Sasada)

On 2018/08/27 18:17, Eric Wong wrote:

> > A joke; at least as far as we know it is a joke :>
>
> I'm not sure why uint64_t doesn't solve time machine problem and uint128_t
> solves it. Any RFC arguing it? (time machines should not leap 1.0790283e+22
> years)?
>
> It delays the onset of the problem. 585 years well within the
> known range of human history; but 1.0790283e+22 years is much
> farther away (probably beyond lifespan of the universe).

We need to a time machine to confirm it :p
(but ruby doesn't support it...
any other language runtime support it? :p)

> > Beside joking, I misunderstand that some systems can change the machine time
> > (time machine may means it) which affect monotonic time, and I could not
> > understand why uint128 can solve such system issue.
> --
> // SASADA Koichi at atdot dot net

Files

0001-thread.c-use-rb_hrtime_t-scalar-for-high-resolution-.patch	30.2 KB	08/20/2018	normalperson (Eric Wong)