Ruby master - Misc #11904

Why was Thread.exclusive deprecated?

12/28/2015 06:31 AM - bascule (Tony Arcieri)

Status:	Rejected	
Priority:	Normal	
Assignee:		

Description

See: https://bugs.ruby-lang.org/projects/ruby-trunk/repository/revisions/52554

Why was Thread.exclusive deprecated? It is useful for when you're uncertain about whether the caller is multithreaded or not, and therefore cannot initialize a mutex because the mutex must be initialized in a thread-safe context where it's not possible for multiple caller threads to initialize the mutex concurrently.

One use case is here: this is an idempotent native function invoked via FFI. The contract is that it can be called repeatedly, but only by one thread at a time (concurrent calls from multiple threads can potentially corrupt its internal state):

https://github.com/cryptosphere/rbnacl/blob/master/lib/rbnacl.rb#L88

Thread.exclusive is useful because it provides an implicit mutex you can ensure is initialized correctly before any other threads start.

History

#1 - 12/28/2015 06:13 PM - Eregon (Benoit Daloze)

Why not create your own Mutex and store it in a constant at load time?
Thread.exclusive could cause accidental contention because it is obviously global for all uses.

#2 - 12/28/2015 06:17 PM - Eregon (Benoit Daloze)

Also in this particular case, it seems relying on require being thread-safe would be good enough for ensuring a single initialization.

#3 - 12/28/2015 09:43 PM - bascule (Tony Arcieri)

Is a single-threaded require actually a *guarantee* of every Ruby VM?

#4 - 12/28/2015 10:42 PM - headius (Charles Nutter)

I agree with Tony that Thread.exclusive still has uses. Was there some discussion on deprecating it?

#5 - 12/29/2015 08:01 PM - bascule (Tony Arcieri)

I received a PR to my project to "add Ruby 2.3 support" (via an inline Mutex) and I think it illustrates the problem:

https://github.com/cryptosphere/rbnacl/pull/127

This doesn't really help, and I guess to make the warning message go away I need to just remove Thread.exclusive

#6 - 12/29/2015 10:58 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

This use of Thread.exclusive seems nonsense.
https://github.com/cryptosphere/rbnacl/commit/25444979d161fa0f02a61a5003768c2173cfa2ee

While Thread.exclusive stops all other threads in 1.8 and earlier, it just blocks other threads which also calling it.
If no other threads calls it, it is useless.

#7 - 12/30/2015 12:29 PM - Eregon (Benoit Daloze)

Tony Arcieri wrote:

> Is a single-threaded require actually a *guarantee* of every Ruby VM?

Not single-threaded, but each file should be loaded at most once.
Single require per file/path is certainly highly desirable,
otherwise you may load the same file twice.

It's also a lot easier to make it thread-safe than autoload IMHO.

#8 - 01/05/2016 05:17 PM - headius (Charles Nutter)

Nobuyoshi Nakada wrote:

> This use of Thread.exclusive seems nonsense.
> https://github.com/cryptosphere/rbnacl/commit/25444979d161fa0f02a61a5003768c2173cfa2ee
>
> While Thread.exclusive stops all other threads in 1.8 and earlier, it just blocks other threads which also calling it.
> If no other threads calls it, it is useless.

That's the point; no two threads can call init via this code at the same time.

Of course, require's guarantee of "only once" would do the same thing here.

#9 - 11/22/2016 06:48 PM - jrochkind (jonathan rochkind)

I came here cause I was curious why Thread.exclusive was deprecated, and didn't find an answer!

But for anyone else curious about the answer to Tony's original question (Tony has probably long since figured it out), I think you can create the Mutex at 'load time', to make sure you only have one? I think you can safely replace all uses of Thread.exclusive with something like this:

```
class RbNaCl::NaCl
    @init_mutex = Mutex.new
    class << self
       def init_mutex
          @init_mutex
       end
     end

    # all the rest of the definition....

end

# no more: Thread.exclusive { RbNaCl::NaCl.sodium_init }

RbNaCl::NaCL.init_mutex.synchronize { RbNaCl::NaCl.sodium_init }

# Or of course you could include the mutex.synchronize inside the `sodium_init` implementation,
# which would probably make more sense.
```

#10 - 11/25/2016 03:41 AM - shyouhei (Shyouhei Urabe)

I feel that Thread.exclusive is too big a primitive to merely initialize a mutex at a process startup. We could perhaps introduce pthread_once_t analogous more fine-grained light-weight control structure that does the job.

P.S. we have /#{@mutex=Mutex.new}/o, so we already have such thing, to some extent at least.

#11 - 11/25/2016 09:21 AM - normalperson (Eric Wong)

shyouhei@ruby-lang.org wrote:

> P.S. we have /#{@mutex=Mutex.new}/o, so we already have such thing, to some extent at least.

Awesome, ROFL; you almost made me choke on my food!

Yes, //o is awesome, but this really takes the cake.

Carry on :)