Ruby trunk - Feature #5446

at_fork callback API

10/14/2011 11:10 AM - normalperson (Eric Wong)

Status:	Assigned	
Priority:	Normal	
Assignee:	kosaki (Motohiro KOSAKI)	
Target version:		

Description

It would be good if Ruby provides an API for registering fork() handlers.

This allows libraries to automatically and agnostically reinitialize resources
such as open IO objects in child processes whenever fork() is called by a user
application. Use of this API by library authors will reduce false/improper
sharing of objects across processes when interacting with other
libraries/applications that may fork.

This Ruby API should function similarly to pthread_atfork() which allows
(at least) three different callbacks to be registered:

1) prepare - called before fork() in the original process
2) parent - called after fork() in the original process
3) child - called after fork() in the child process

It should be possible to register multiple callbacks for each action
(like at_exit and pthread_atfork(3)).

These callbacks should be called whenever fork() is used:

- Kernel#fork
- IO.popen
- ``
- Kernel#system

... And any other APIs I've forgotten about

I also want to consider handlers that only need to be called for plain
fork() use (without immediate exec() afterwards, like with `` and system()).

Ruby already has the internal support for most of this this to manage mutexes,
Thread structures, and RNG seed. Currently, no external API is exposed. I can
prepare a patch if an API is decided upon.

Related issues:

Related to Ruby trunk - Bug #14009: macOS High Sierra and “fork” compatibility	**Closed**

History

#1 - 10/14/2011 05:15 PM - kosaki (Motohiro KOSAKI)

As you know, we can only call asynchronous-signal-safe function between fork and exec when the process is multi threaded.
but ruby code invocation definitely need to use malloc which not async-signal-safe. so, it's pretty hard to implement.

Am I missing something?

#2 - 10/23/2011 05:21 PM - naruse (Yui NARUSE)

- Project changed from CommonRuby to Ruby trunk

- Target version deleted (2.6)

#3 - 03/27/2012 10:45 PM - mame (Yusuke Endoh)

- Status changed from Open to Feedback

Eric Wong,

Do you still want this feature?
If so, could you answer kosaki's comment?

OT: We noticed and surprised at your ID (normalperson) at the recent developers' meeting in Akihabara. Clearly, you are greatperson :-)

--
Yusuke Endoh mame@tsg.ne.jp

#4 - 04/09/2012 01:53 PM - normalperson (Eric Wong)

"mame (Yusuke Endoh)" mame@tsg.ne.jp wrote:

> Do you still want this feature?

Yes, but lower priority.

I think default to IO#close_on_exec=true for 2.0.0 makes this less important.

> If so, could you answer kosaki's comment?

kosaki wrote:

> As you know, we can only call asynchronous-signal-safe function between fork and exec when the process is multi threaded. but ruby code invocation definitely need to use malloc which not async-signal-safe. so, it's pretty hard to implement. Am I missing something?

I can't edit the existing ticket, but I think only Kernel#fork should be touched. Methods that call exec after fork will already get things cleaned up from close_on_exec.

If there is a Ruby call to exec, then we already have a chance to use non-async-signal safe code.

It could be implemented in pure Ruby, even. This is a prototype (using xfork name) intead:

```
ATFORK = {
  :prepare => [ lambda { puts ":prepare in #$$" } ],
  :parent => [ lambda { puts ":parent in #$$" } ],
  :child => [ lambda { puts ":child in #$$" } ],
}

def xfork
  ATFORK[:prepare].each { |code| code.call }
  if block_given?
    pid = fork do
      ATFORK[:child].each { |code| code.call }
      yield
    end
    ATFORK[:parent].each { |code| code.call }
  else
    case pid = fork
    when nil
      ATFORK[:child].each { |code| code.call }
    when Integer
      ATFORK[:parent].each { |code| code.call }
    end
  end

  pid
end
```

I haven't thought of an API to manipulate the ATFORK arrays with. I don't want to emulate pthread_atfork() directly, it's too cumbersome for Ruby. Perhaps:

```
at_fork(:prepare) { ... }
at_fork(:child) { ... }
at_fork(:parent) { ... }
```

> OT: We noticed and surprised at your ID (normalperson) at the recent
> developers' meeting in Akihabara. Clearly, you are greatperson :-)

I don't think of myself as great. But if others think I'm great,
they should try to be like me, then we'll all be normal :)

#5 - 11/20/2012 10:32 PM - mame (Yusuke Endoh)

- Status changed from Feedback to Assigned

- Assignee set to kosaki (Motohiro KOSAKI)

- Target version set to 2.6

#6 - 11/06/2013 06:48 AM - sam.saffron (Sam Saffron)

This is a critical feature for Ruby imho, at the moment there are 100 mechanisms for at_fork, we need a clean, supported ordered one.

I think there should be strong parity with at_exit, so am not particularly fond of the symbol param. I would like.

at_fork{ } # returns proc, a stack of procs that are called in order just before fork
after_fork {} # returns a proc, a stack of procs that are called in order just after child process launches

The prepare thing for me seems like overkill. It would be like we are implementing 2 queues, one urgent, one less urgent. I don't really see the point.

--

There is a question of cancelling forks (eg: what happens when an exception is thrown during these callbacks?)

#7 - 11/06/2013 10:38 AM - jasonrclark (Jason Clark)

I'd love to see this added. Gems using threads (like newrelic_rpm) have a lot of potential for deadlocks when forking happens. This would gives a nice mechanism for dealing with those issues more generally, rather than having to hook things gem-by-gem like we do today.

New Relic + Resque has seen a lot of these types of problems, some of which are documented at https://github.com/resque/resque/issues/1101. While Resque has gem-specific hooks we lean on, having those hooks be at the Ruby level instead would be awesome.

Is there any possibility of this type of after_fork hook could also apply to daemonizing with Process.daemon? While we have fewer deadlocks, we often lose visibility after processes daemonize because we don't know to start our threads back up in the daemonized process. (See https://github.com/puma/puma/issues/335 for an example with the Puma web server).

#8 - 12/01/2013 06:52 AM - tmm1 (Aman Gupta)

Here is another example of a gem which implement its own fork hooks: https://github.com/zk-ruby/zk/blob/master/lib/zk/fork_hook.rb

#9 - 12/01/2013 06:54 AM - tmm1 (Aman Gupta)

Simple implementation of an after_fork{} hook: https://github.com/tmm1/ruby/commit/711a68b6599d176c5bcb4ef0c90aa195a290d1c0

Any objection?

#10 - 12/01/2013 07:12 AM - Eregon (Benoit Daloze)

tmm1 (Aman Gupta) wrote:

> Simple implementation of an after_fork{} hook: https://github.com/tmm1/ruby/commit/711a68b6599d176c5bcb4ef0c90aa195a290d1c0
>
> Any objection?

Sounds good and useful to me!

#11 - 12/01/2013 08:23 AM - normalperson (Eric Wong)

"tmm1 (Aman Gupta)" ruby@tmm1.net wrote:

> Simple implementation of an after_fork{} hook: https://github.com/tmm1/ruby/commit/711a68b6599d176c5bcb4ef0c90aa195a290d1c0
>
> Any objection?

I think there needs to be separate hooks for prepare/parent/child
(like pthread_atfork(3)). The parent may need to release resources
before forking (prepare hook), and perhaps reacquire/reinitialize
them after forking (parent hook).

The prepare hook is important for things like DB connections;
the parent hook might be less useful (especially for apps which
fork repeatedly).

#12 - 12/01/2013 09:25 AM - tmm1 (Aman Gupta)

I'd like to add a before_fork{} hook that fires in the parent before the fork. An after_fork hook in the parent seems unnecessary.

#13 - 12/06/2013 07:53 AM - kosaki (Motohiro KOSAKI)

2013/11/30 tmm1 (Aman Gupta) ruby@tmm1.net:

> Issue #5446 has been updated by tmm1 (Aman Gupta).
>
> Simple implementation of an after_fork{} hook: https://github.com/tmm1/ruby/commit/711a68b6599d176c5bcb4ef0c90aa195a290d1c0
>
> Any objection?

??!?!?

1. Why no before_fork?
2. zk has :after_child nad :after_parent hook and your patch don't. Why?
3. Why should the new method return proc?
4. When rb_daemon() is used, some platform call after_fork once and the other call twice. It seems useless.
5. Why do hook fire at rb_thread_atfork() make a lot of new array?
6. Your patch doesn't aim to remove the hooks and I'm sure it is required.

You said the new method should be created for killing the gem specific hooks.
But your patch seems not to be able to.

#14 - 12/06/2013 03:37 PM - tmm1 (Aman Gupta)

Thanks for your feedback.

> 1. Why no before_fork?

I planned to add this in the same way as after_fork, as long as there are no issues with my patch.
I wasn't sure if rb_vm_t is the best place for the new hooks.
If it seems sane, I can expand the patch.

> 1. zk has :after_child nad :after_parent hook and your patch don't. Why?

With two hooks, before_fork/after_fork methods are enough. If we want to implement three hooks, should we add three new methods?

> 1. Why should the new method return proc?

I guessed the return value can be used to de-register the hook later. But there is no way to do this currently.

> 1. When rb_daemon() is used, some platform call after_fork once and the other call twice. It seems useless.

Ah, good point. Maybe we need a flag to ensure only one invocation.

> 1. Why do hook fire at rb_thread_atfork() make a lot of new array?

No reason. Patch was a simple proof of concept for feedback.

> 1. Your patch doesn't aim to remove the hooks and I'm sure it is required.

Do you have any API ideas for this? I agree we need some way to remove the hooks.

One option is to use tracepoint api:

tp = TracePoint.new(:before_fork, :after_fork_child){ ... }

tp.enable
tp.disable

#15 - 12/07/2013 06:53 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> the parent hook might be less useful (especially for apps which
> fork repeatedly).

I take that back. It can be useful for for setting up pipes/sockets for IPC between parent and child.

Example without wrapper class and explicit IO#close:

```
r, w = IO.pipe
pid = fork do
  w.close # could be atfork_child
  ...
end
r.close # could be atfork_parent
...
```

However, I want to do this via callback, example with Worker class:

```
class Worker
  attr_writer :pid

  def initialize
    @r, @w = IO.pipe
    Process.atfork_parent { @r.close unless @r.closed? }
    Process.atfork_child { @w.close unless @w.closed? }
  end
end

worker = Worker.new # IO.pipe
worker.pid = fork { ... }
...

# No need to remember what to close in parent/child
```

#16 - 12/07/2013 07:29 AM - tmm1 (Aman Gupta)

Ok good point. I agree we should add all three if we're going to do this.

I like the Process.atfork_parent{} API in your example. If we add three methods to Process, the only thing missing is a way to un-register a hook.

#17 - 12/07/2013 01:02 PM - tmm1 (Aman Gupta)

Another idea:

Process.at_fork{ |loc| case loc; when :before_fork; ... end } -> proc
Process.remove_at_fork(proc)

#18 - 12/07/2013 01:53 PM - akr (Akira Tanaka)

2013/12/7 Eric Wong normalperson@yhbt.net:

> However, I want to do this via callback, example with Worker class:
>
> ```
> class Worker
>   attr_writer :pid
>
>   def initialize
>     @r, @w = IO.pipe
>     Process.atfork_parent { @r.close unless @r.closed? }
>     Process.atfork_child { @w.close unless @w.closed? }
>   end
> end
>
> worker = Worker.new # IO.pipe
> worker.pid = fork { ... }
> ...
> ```

```
# No need to remember what to close in parent/child
```

I think it doesn't work with multiple thread.

2.times {
Thread.new {
worker = Worker.new # IO.pipe
worker.pid = fork { ... }
...
}
}

If fork for worker 1 is called between IO.pipe and fork for worker 2,
pipes for worker 2 is leaked for the process for worker 1 and
not inherited to the process for worker 2.

I feel it is not a good example for this issue.
--
Tanaka Akira

#19 - 12/08/2013 03:53 PM - normalperson (Eric Wong)

Tanaka Akira akr@fsij.org wrote:

> 2013/12/7 Eric Wong normalperson@yhbt.net:
>
> > However, I want to do this via callback, example with Worker class:
> >
> > ```
> > class Worker
> >   attr_writer :pid
> >
> >   def initialize
> >   @r, @w = IO.pipe
> >   Process.atfork_parent { @r.close unless @r.closed? }
> >   Process.atfork_child { @w.close unless @w.closed? }
> >   end
> > end
> >
> > worker = Worker.new # IO.pipe
> > worker.pid = fork { ... }
> > ...
> >
> > # No need to remember what to close in parent/child
> > ```
>
> I think it doesn't work with multiple thread.

True, but I wasn't intending this example to be used for an MT
application, but a single-threaded, multi-process HTTP server.

Generally, I do not use fork after I've spawned threads (unless
followed immediately with exec).

#20 - 11/02/2016 02:46 AM - sam.saffron (Sam Saffron)

ping, any plans to add these apis, they will help clean up a lot in the Ruby ecosystem, just hit it again today.

I think what is really needed here is a conceptual Yes or No from Koichi or Matz

#21 - 11/05/2016 05:06 PM - pitr.ch (Petr Chalupa)

We had to get around missing at_fork in concurrent-ruby as well. Since we are avoiding monkey_patching we had to result in lazy-checking pid, when it changes we re-create a resource. The resource has to check on every usage though which is not good.

Having Process.remove_at_fork(proc) is critical, an user of at_fork has to be able to unregister callback when the resource is not used any further.

#22 - 12/21/2016 03:08 PM - shyouhei (Shyouhei Urabe)

We looked at this issue at today's developer meeting.

I was told that async-signal-safety is no longer the issue because in comment #4 the OP already restricted the request to fork method that runs ruby code. That's a good thing to know.

Then, given repeated request of this functionality, why not start this feature as a gem? Because we focus on fork that runs ruby code, the needed functionality is doable in pure ruby I think. Is there any reason this should be done by the language core (apart from monkey-paytching ugliness)?

#23 - 12/21/2016 05:42 PM - Eregon (Benoit Daloze)

> Is there any reason this should be done by the language core (apart from monkey-paytching ugliness)?

Yes, Ruby code cannot change e.g. rb_f_fork().
There is already atfork-related logic and it seems brittle to override something as essential as #fork.
Since libraries which need this feature currently monkey-patch #fork, it means they do not compose well.
Monkey-patching is considered very fragile for this, I guess that's why no gem for it exists or it would not be used.

If monkey-patching is not used (because of above problems),
the cost of checking on every access becomes a problem for maintenance and performance.
Sometimes, it even means a library decides to be thread-safe but not process-safe:
https://github.com/jeremyevans/sequel/issues/691

Another example is Passenger and other web servers which have their own hook for after_fork.
This means the application is now forced to keep the same web server, or change the hook (very easy to forget).

Let's add Process.at_fork and Process.remove_at_fork,
to encourage more responsible programming.
The evidence for the need of a common API for this is obvious.

#24 - 10/13/2017 12:52 AM - shyouhei (Shyouhei Urabe)

- Related to Bug #14009: macOS High Sierra and “fork” compatibility added

#25 - 07/19/2018 06:29 AM - kivikakk (Ashe Connor)

Are we still interested in pursuing something like this? I'd be happy to push it forward.

#26 - 07/19/2018 05:42 PM - normalperson (Eric Wong)

ashe@kivikakk.ee wrote:

> Are we still interested in pursuing something like this? I'd
> be happy to push it forward.

I'm not sure it's necessary, anymore. Most programs and libs
already have workarounds at this point. pthread_atfork can also
be the source of surprising behaviors (thread/async-signal
safety), so Austin group proposed _Fork function:
http://austingroupbugs.net/view.php?id=62

> https://bugs.ruby-lang.org/issues/5446#change-73019
>
> These callbacks should be called whenever fork() is used:
>
> - IO.popen
> - ``
> - Kernel#system

Furthermore, many platforms have vfork (it's possible to support
vfork without MMU) and we favor using that for process spawning,
so the following is unneccessary:

> I also want to consider handlers that only need to be called for plain
> fork() use (without immediate exec() afterwards, like with `` and system()).

#27 - 07/19/2018 08:16 PM - Eregon (Benoit Daloze)

IMHO, this is is still very much needed for cases using plain fork().
The default close-on-exec behavior is irrelevant for fork() without exec() and does not help.
Inheriting, e.g., a database connection in such a case is extremely harmful (data corruption or leaking sensitive data) and a common pitfall.
Having a nice at_fork hook would allow libraries to safely disconnect the database in each forked process to avoid accidental file descriptor sharing between forks.

Sequel:
https://github.com/jeremyevans/sequel/issues/691

Passenger has PhusionPassenger.on_event(:starting_worker_process).
They give the example of memcached and by default use it to reestablish ActiveRecord connections:
https://www.phusionpassenger.com/library/indepth/ruby/spawn_methods/#unintentional-file-descriptor-sharing

Puma has before_fork:
https://github.com/puma/puma/pull/754/files

Unicorn implements its own before_fork/after_fork:
https://www.rubydoc.info/gems/unicorn/5.0.1/Unicorn%2FConfigurator:after_fork
normalperson (Eric Wong) Don't you think this deserves to be in core?

I would guess all of these do not work if rb_fork() is called.
What more motivation do we need?
Wait that a major incident happen due to unintentional fd sharing because database libraries cannot protect themselves from fork()?

#28 - 07/20/2018 10:22 AM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> IMHO, this is is still very much needed for cases using plain fork().
> The default close-on-exec behavior is irrelevant for fork() without exec() and does not help.

Agreed.

> Inheriting, e.g., a database connection in such a case is
> extremely harmful (data corruption or leaking sensitive data)
> and a common pitfall.

It's been a known problem for decades, now (at least since the
days of mod_perl + DBI on Apache 1.x); and AFAIK there's no data
leaks from it. Anybody who makes that mistake would likely
raise/crash/burn long before seeing, much less leaking sensitive
data.

> Having a nice at_fork hook would allow to safely disconnect the database in each forked process to avoid accidental file descriptor sharing between forks.
>
> Sequel:
> https://github.com/jeremyevans/sequel/issues/691

I agree with Jeremy on this; it will likely cause new problems
and surprises if libraries use it.

> Passenger has PhusionPassenger.on_event(:starting_worker_process).
> They give the example of memcached and by default use it to reestablish ActiveRecord connections:
> https://www.phusionpassenger.com/library/indepth/ruby/spawn_methods/#unintentional-file-descriptor-sharing
>
> Puma has before_fork:
> https://github.com/puma/puma/pull/754/files

Exactly my point, everything relevant which forks has documented
and supported means to disconnect/restart connections.

> normalperson (Eric Wong) Don't you think this deserves to be in core?

Not anymore.

> I would guess all of these do not work if rb_fork() is called.
> What more motivation do we need?
> Wait that a major incident happen due to unintentional fd
> sharing because database libraries cannot protect themselves
> from fork()?

Again, this is a known problem with known workarounds for
decades, now. I expect nobody has been able to get close to
production or dealing with important data with such a broken

setup.

#29 - 07/20/2018 12:06 PM - Eregon (Benoit Daloze)

normalperson (Eric Wong) wrote:

> It's been a known problem for decades, now (at least since the
> days of mod_perl + DBI on Apache 1.x); and AFAIK there's no data
> leaks from it. Anybody who makes that mistake would likely
> raise/crash/burn long before seeing, much less leaking sensitive data.

Yes, it's not a new problem.
I disagree about no production leaks, because it happened to me on a website running for a national programming contest.
For most of the contest it was fine as one process was able to handle the load,
but at some point the webserver decided to spawn another process by forking,
people starting seeing each's other solution, the scores were corrupted and everyone was puzzled as to what happened.
We had to stop the contest due to this.

I want to help protect future programmers from such bugs, if at all possible.
And I believe it's possible.

> I agree with Jeremy on this; it will likely cause new problems
> and surprises if libraries use it.

Let's design it so it doesn't.
What's the harm/surprise to reconnect in at_fork(:after_child) for instance?

The current hooks are webserver-specific and so migrating between unicorn/puma/passenger/etc means
it's quite easy to forget to adapt to the new webserver hook, which would trigger this bug.
Especially if e.g., using ActiveRecord with Passenger, where the hook is automatic and migrating to another forking webserver.
Having standard hooks solves this, also works for rb_f_fork(),
and empowers libraries to solve this issue (not just DB libraries but also concurrent-ruby (see above), etc).

#30 - 07/20/2018 08:52 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> normalperson (Eric Wong) wrote:
>
> > It's been a known problem for decades, now (at least since the
> > days of mod_perl + DBI on Apache 1.x); and AFAIK there's no data
> > leaks from it. Anybody who makes that mistake would likely
> > raise/crash/burn long before seeing, much less leaking sensitive data.
>
> Yes, it's not a new problem.
> I disagree about no production leaks, because it happened to me on a website running for a national programming contest.
> For most of the contest it was fine as one process was able to handle the load,
> but at some point the webserver decided to spawn another process by forking,
> people starting seeing each's other solution, the scores were corrupted and everyone was puzzled as to what happened.
> We had to stop the contest due to this.

fork is full of caveats; using atfork hook to work around one
caveat out of many is not a solution. The solution is knowing
the caveats of the tools you use.

In your case, it seemed like you were not paying attention to
the server setup at all and would not have known to use atfork
hook regardless if it was in the webserver or core.

> I want to help protect future programmers from such bugs, if at all possible.
> And I believe it's possible.
>
> > I agree with Jeremy on this; it will likely cause new problems
> > and surprises if libraries use it.
>
> Let's design it so it doesn't.
> What's the harm/surprise to reconnect in at_fork(:after_child) for instance?

It's a waste of time and resources when the child has no need for

a connection at all. Simply put, library authors have no clue how an application will use/manage processes and would (rightfully) not bother using such hooks.

Also, consider that pthread_atfork has been around for many years, it's not adopted by library authors (of C/C++ libraries) because of problems surrounding it; and POSIX is even considering deprecating pthread_atfork[1].

How about an alternate proposal?

```
Introduce a new object_id-like identifier which changes
across fork: Thread.current.thread_id
```

It doesn't penalize platforms without fork, and can work well with existing thread-aware code.

IMHO, Thread.current.object_id being stable in forked child isn't good; but I expect compatibility problems if we change it at this point. At least some usages of monitor.rb would break.

> The current hooks are webserver-specific and so migrating between unicorn/puma/passenger/etc means it's quite easy to forget to adapt to the new webserver hook, which would trigger this bug.

I hate the amount of vendor lock-in each webserver has.
But making hooks which library authors can fire unpredictably on application authors is worse, especially if there's no "opt-out".

[1] http://austingroupbugs.net/view.php?id=858

#31 - 07/23/2018 10:51 PM - tenderlovemaking (Aaron Patterson)

normalperson (Eric Wong) wrote:

> eregontp@gmail.com wrote:
>
> > normalperson (Eric Wong) wrote:
> >
> > > It's been a known problem for decades, now (at least since the days of mod_perl + DBI on Apache 1.x); and AFAIK there's no data leaks from it. Anybody who makes that mistake would likely raise/crash/burn long before seeing, much less leaking sensitive data.
> >
> > Yes, it's not a new problem.
> > I disagree about no production leaks, because it happened to me on a website running for a national programming contest.
> > For most of the contest it was fine as one process was able to handle the load,
> > but at some point the webserver decided to spawn another process by forking,
> > people starting seeing each's other solution, the scores were corrupted and everyone was puzzled as to what happened.
> > We had to stop the contest due to this.

I've experienced a data leak similar to this. Saying the impact was "catastrophic" would be an understatement. :(

> fork is full of caveats; using atfork hook to work around one caveat out of many is not a solution. The solution is knowing the caveats of the tools you use.
>
> In your case, it seemed like you were not paying attention to the server setup at all and would not have known to use atfork hook regardless if it was in the webserver or core.
>
> > I want to help protect future programmers from such bugs, if at all possible.
> > And I believe it's possible.
> >
> > > I agree with Jeremy on this; it will likely cause new problems and surprises if libraries use it.

> Let's design it so it doesn't.
> What's the harm/surprise to reconnect in at_fork(:after_child) for instance?

> It's a waste of time and resources when the child has no need for
> a connection at all. Simply put, library authors have no clue
> how an application will use/manage processes and would
> (rightfully) not bother using such hooks.

I think library authors can make things easier though. Web frameworks, like Rails for example, are expected to handle this situation for the user. In addition, say a library author provided no such feature like Sequel, how would a user know they need to call DB.disconnect after a fork? Are they responsible for completely understanding the implementation of the library they are using? Even if an end user called DB.disconnect in an after fork hook, what if that wasn't enough? How would an end user know what needs to be called?

> Also, consider that pthread_atfork has been around for many
> years, it's not adopted by library authors (of C/C++ libraries)
> because of problems surrounding it; and POSIX is even
> considering deprecating pthread_atfork[1].
>
> How about an alternate proposal?
>
> Introduce a new object_id-like identifier which changes
> across fork: Thread.current.thread_id
>
> It doesn't penalize platforms without fork, and can work well
> with existing thread-aware code.

I think this is a good idea, but I'm not sure it addresses the communication issue I brought up. IMO it would be great to have some sort of hook so that library authors can dictate what "the right thing to do" is after a fork (maybe there are other resources or caches that need to be cleaned, and maybe that changes from version to version).

> IMHO, Thread.current.object_id being stable in forked child
> isn't good; but I expect compatibility problems if we change it
> at this point. At least some usages of monitor.rb would
> break.
>
> > The current hooks are webserver-specific and so migrating
> > between unicorn/puma/passenger/etc means it's quite easy to
> > forget to adapt to the new webserver hook, which would trigger
> > this bug.
>
> I hate the amount of vendor lock-in each webserver has.
> But making hooks which library authors can fire unpredictably
> on application authors is worse, especially if there's no
> "opt-out".

I think requiring users to specify a db disconnect after fork causes even more "vendor lock-in". Lets say I did add the after fork code to deal with Sequel, but now I want to switch to a threaded webserver. Now I have to do more work to figure out what's required (if anything) in a threaded environment. It puts the onus on the app dev to figure out what's right for a particular environment, and that means it's harder to change: locking you in by making more work.

Additionally, forking servers all have to provide this type of hook anyway (Unicorn, Resque, Puma, to name a few) but today they have to specify their own API. I think it would be great if we had a "Rack for fork hooks", if that makes sense. :)

#32 - 07/24/2018 04:41 AM - jeremyevans0 (Jeremy Evans)

tenderlovemaking (Aaron Patterson) wrote:

> I think library authors can make things easier though. Web frameworks, like Rails for example, are expected to handle this situation for the user. In addition, say a library author provided no such feature like Sequel, how would a user know they need to call DB.disconnect after a fork? Are they responsible for completely understanding the implementation of the library they are using? Even if an end user called DB.disconnect in an after fork hook, what if that wasn't enough? How would an end user know what needs to be called?

Preloading before forking is not the default in unicorn or puma, and the documentation for both unicorn and puma mention the related issues in the documentation that describes how to enable preloading before forking. Additionally, Sequel's documentation mentions the need to disconnect when using preloading before forking. Users should not be expected to understand the implementation of the libraries they are using, but they should be expected to read the documentation for non-default configuration options they explicitly enable.

Now, passenger enables preloading before forking by default. Some people may consider that an issue with passenger or others may be OK with them just optimizing for the case where only ActiveRecord is used without any other libraries that would allocate file descriptors. To be fair,

passenger's documentation does discuss the issue in detail.

FWIW, disconnecting (at least with Sequel) should be done before forking, not after forking. After forking you are already sharing the file descriptors. As long as the library can reestablish connections as needed (as Sequel can), a single line of code to disconnect before fork is globally much simpler than extensive code that tries (and fails) to handle all possible cases when fork can be used.

I will posit that any attempt to disconnect automatically during forking (either in the parent or child) can break a reasonable (if less common) setup where forking is used correctly without explicit disconnection.

> How about an alternate proposal?
>
> ```
> Introduce a new object_id-like identifier which changes
> across fork: Thread.current.thread_id
> ```
>
> It doesn't penalize platforms without fork, and can work well
> with existing thread-aware code.

I think this is a good idea, but I'm not sure it addresses the communication issue I brought up. IMO it would be great to have some sort of hook so that library authors can dictate what "the right thing to do" is after a fork (maybe there are other resources or caches that need to be cleaned, and maybe that changes from version to version).

At least for Sequel, I don't think this will matter. I'm neither for nor against this.

Additionally, forking servers all have to provide this type of hook anyway (Unicorn, Resque, Puma, to name a few) but today they have to specify their own API. I think it would be great if we had a "Rack for fork hooks", if that makes sense. :)

I think the main problem here is that libraries may offer before_fork or after_fork hooks that are called with arguments that a in-core at_fork hook couldn't use. I know that is the case with unicorn.

Additionally, there is the issue with in-core at_fork hooks in terms of inheritance:

```
  fork do # calls at_fork hooks
    fork do # calls at_fork hooks?
    end
  end
```

There are arguments for inheriting the hooks, and there are arguments for not doing so. I suppose you could make it configurable, but that increases complexity. When fork hooks are implemented in a library, they apply to only the case where the library forks, and not all cases where fork is called.

Consider that you are using a preforking webserver, and in your parent process, you do:

```
r, w = IO.pipe
if fork
  # ...
else
  # ...
  exit!
end
```

If an in-core at_fork handler is used, the webserver's forking hooks may be called for this fork, when that may not be desired. That is not the case for library-specific fork hooks that wrap the library's usage of fork.

#33 - 07/24/2018 04:42 AM - normalperson (Eric Wong)

tenderlove@ruby-lang.org wrote:

> normalperson (Eric Wong) wrote:
>
> > eregontp@gmail.com wrote:
> >
> > > I want to help protect future programmers from such bugs, if at all possible.
> > > And I believe it's possible.
> > >
> > > > I agree with Jeremy on this; it will likely cause new problems
> > > > and surprises if libraries use it.
> > >
> > > Let's design it so it doesn't.
> > > What's the harm/surprise to reconnect in at_fork(:after_child) for instance?
> >
> > It's a waste of time and resources when the child has no need for

> > a connection at all. Simply put, library authors have no clue how an application will use/manage processes and would (rightfully) not bother using such hooks.
>
> I think library authors can make things easier though. Web frameworks, like Rails for example, are expected to handle this situation for the user. In addition, say a library author provided no such feature like Sequel, how would a user know they need to call DB.disconnect after a fork? Are they responsible for completely understanding the implementation of the library they are using? Even if an end user called DB.disconnect in an after fork hook, what if that wasn't enough? How would an end user know what needs to be called?

I don't know how to deal with unknowledgeable users aside from making them knowledgeable about the tools they use.

However, the issue also relates to allowing things to become thread-safe and fiber-safe at the moment, and in the future: Guild-safe.

So maybe ko1 can give his thoughts on this topic since he is working on Guilds

> > Also, consider that pthread_atfork has been around for many years, it's not adopted by library authors (of C/C++ libraries) because of problems surrounding it; and POSIX is even considering deprecating pthread_atfork[1].
> >
> > How about an alternate proposal?
> >
> > Introduce a new object_id-like identifier which changes across fork: Thread.current.thread_id
> >
> > It doesn't penalize platforms without fork, and can work well with existing thread-aware code.
>
> I think this is a good idea, but I'm not sure it addresses the communication issue I brought up. IMO it would be great to have some sort of hook so that library authors can dictate what "the right thing to do" is after a fork (maybe there are other resources or caches that need to be cleaned, and maybe that changes from version to version).

Library authors also do not know if the child process will touch their objects/code; so making such assumptions can be costly when child processes with different goals are used.

I think the only thing low-level library authors (e.g. authors of libpq/sqlite3/libcurl/etc...) can do is document how to deal with such issues when interacting with threads and processes.

Maybe Sequel and AR can have add optional PID checks ($$!= expected_pid), which is totally generic and usable in bunch of cases.

One thing I have been annoyed by is END/at_exit hooks firing unexpectedly in child processes. I work around them with the PID check. So I want to avoid having the same annoyance from at_exit by avoiding atfork hooks.

> > IMHO, Thread.current.object_id being stable in forked child isn't good; but I expect compatibility problems if we change it at this point. At least some usages of monitor.rb would break.
> >
> > > The current hooks are webserver-specific and so migrating between unicorn/puma/passenger/etc means it's quite easy to forget to adapt to the new webserver hook, which would trigger this bug.

> I hate the amount of vendor lock-in each webserver has.
> But making hooks which library authors can fire unpredictably
> on application authors is worse, especially if there's no
> "opt-out".

> I think requiring users to specify a db disconnect after fork
> causes even more "vendor lock-in". Lets say I did add the
> after fork code to deal with Sequel, but now I want to switch
> to a threaded webserver. Now I have to do more work to figure
> out what's required (if anything) in a threaded environment.
> It puts the onus on the app dev to figure out what's right for
> a particular environment, and that means it's harder to
> change: locking you in by making more work.

AFAIK, Sequel and AR already provides out-of-the-box thread-safe
behavior. Perhaps Sequel and AR should also have out-of-the-box
thread-AND-fork-safe connection pool which checks the PID. I
don't expect PID check cost to be expensive, but they can make
that an option for users and it'll work on all forking servers
and maybe Guild servers in the future.

People who don't use forking can use old thread-safe-only or
single-threaded-only code.

The key thing I want to avoid from having atfork hooks is:

```
DB.disconnect
pid = fork do
  # something which never touches DB
  exit!(0) # avoid at_exit hook
end
DB.reconnect
```

It's needlessly expensive in the parent process to pay the
disconnect/reconnect cost when forking a child which never
touches the DB. And there's already a gotcha for avoiding
at_exit hooks.

> Additionally, forking servers all have to provide this type of
> hook anyway (Unicorn, Resque, Puma, to name a few) but today
> they have to specify their own API. I think it would be great
> if we had a "Rack for fork hooks", if that makes sense. :)

I don't think limiting this to fork is necessary; forking may
become obsolete with Guilds.

#34 - 07/24/2018 04:57 AM - jeremyevans0 (Jeremy Evans)

normalperson (Eric Wong) wrote:

> Maybe Sequel and AR can have add optional PID checks
> ($$!= expected_pid), which is totally generic and usable
> in bunch of cases.

My problem with doing this by default is that it penalizes knowledgeable users just to avoid problems for users that don't read the documentation. Some libraries may consider that an acceptable tradeoff, but I do not. I guess it depends at least partly on the expected audience for the library.

I'd be OK with this as an optional behavior, but in general anyone who would use it would be better off just disconnecting before forking, so I haven't implemented it yet.

> AFAIK, Sequel and AR already provides out-of-the-box thread-safe
> behavior. Perhaps Sequel and AR should also have out-of-the-box
> thread-AND-fork-safe connection pool which checks the PID. I
> don't expect PID check cost to be expensive, but they can make
> that an option for users and it'll work on all forking servers
> and maybe Guild servers in the future.

I'm guessing that Guild servers will operate differently than forking servers (not having different PIDs for different Guilds), but ko1 would know more about that.

> I don't think limiting this to fork is necessary; forking may become obsolete with Guilds.

I do not think that will be the case. For one thing, you are always going to want a forking approach if you want a parent process that forks children and reaps them when they crash/exit. This is especially true when the child process uses different permissions than the parent via setuid. It could definitely be the case that a single-process multi-Guild approach will become more popular than current forking multi-process servers.