# Ruby - Feature #17795

## Around `Process.fork` callbacks API

04/12/2021 08:09 AM - byroot (Jean Boussier)

| | |
|---|---|
| **Status:** | Closed |
| **Priority:** | Normal |
| **Assignee:** | |
| **Target version:** | |

**Description**

Replaces: https://bugs.ruby-lang.org/issues/5446

## Context

Ruby code in production is very often running in a forking setup (puma, unicorn, etc), and it is common some types of libraries to need to know when the Ruby process was forked. For instance:

- Most database clients, ORMs or other libraries keeping a connection pool might need to close connections before the fork happens.
- Libraries relying on some kind of dispatcher thread might need to restart the thread in the forked children, and clear any internal buffer (e.g. statsd clients, newrelic_rpm).

**This need is only for forking the whole ruby process, extensions doing a fork(2) + exec(2) combo etc are not a concern, this aim at only catching kernel.fork, Process.fork and maybe Process.daemon.**.
The use case is for forks that end up executing Ruby code.

## Current solutions

Right now this use case is handled in several ways.

**Rely on the integrating code to call a before_fork or after_fork callback.**

Some libraries simply rely on documentation and require the user to use the hooks provided by their forking server.

Examples:

- Sequel: http://sequel.jeremyevans.net/rdoc/files/doc/fork_safety_rdoc.html
- Rails's Active Record: https://devcenter.heroku.com/articles/concurrency-and-database-connections#multi-process-servers
- ScoutAPM (it tries to detect popular forking setup and register itself): https://github.com/scoutapp/scout_apm_ruby/blob/fa83793b9e8d2f9a32c920f59b57d7f198f466b8/lib/scout_apm/environment.rb#L142-L146
- NewRelic RPM (similarly tries to register to popular forking setups): https://www.rubydoc.info/github/newrelic/rpm/NewRelic%2FAgent:after_fork

**Continuously check Process.pid**

Some libraries chose to instead keep the process PID in a variable, and to regularly compare it to Process.pid to detect forked children.
Unfortunately Process.pid is relatively slow on Linux, and these checks tend to be in tight loops, so it's not uncommon when using these libraries
to spend 1 or 2% of runtime in Process.pid.

Examples:

- Rails's Active Record used to check Process.pid https://github.com/Shopify/rails/blob/411ccbdab2608c62aabdb320d52cb02d446bb39c/activerecord/lib/active_record/connection_adapters/abstract/connection_pool.rb#L946, it still does but a bit less: https://github.com/rails/rails/pull/41850
- the Typhoeus HTTP client: https://github.com/typhoeus/typhoeus/blob/a345545e5e4ac0522b883fe0cf19e5e2e807b4b0/lib/typhoeus/pool.rb#L34-L42
- Redis client: https://github.com/redis/redis-rb/blob/6542934f01b9c390ee450bd372209a04bc3a239b/lib/redis/client.rb#L384
- Some older versions of NewRelic RPM: https://github.com/opendna/scoreranking-api/blob/8fba96d23b4d3e6b64f625079c184f3a292bbc12/vendor/gems/ruby/1.9.1/gems/newrelic_rpm-3.7.3.204/lib/new_relic/agent/harvester.rb#L39-L41

**Continuously check Thread#alive?**

Similar to checking Process.pid, but for the background thread use case. Thread#alive? is regularly checked, and if the thread is dead, it is assumed that the process was forked.
It's much less costly than a Process.pid, but also a bit less reliable as the thread could have died for other reasons. It also delays re-creating the thread to the next check rather than immediately upon forking.

Examples:

- statsd-instrument: https://github.com/Shopify/statsd-instrument/blob/0445cca46e29aa48e9f1efec7c72352aff7ec931/lib/statsd/instrument/batched_udp_sink.rb#L63

**Decorate Kernel.fork and Process.fork**

Another solution is to prepend a module in Process and Kernel, to decorate the fork method and implement your own callback. It works well, but is made difficult by Kernel.fork.

Examples:

- Active Support: https://github.com/rails/rails/blob/9aed3dcdfea6b64c18035f8e2622c474ba499846/activesupport/lib/active_support/fork_tracker.rb
- dd-trace-rb: https://github.com/DataDog/dd-trace-rb/blob/793946146b4709289cfd459f3b68e8227a9f5fa7/lib/ddtrace/profiling/ext/forking.rb
- To some extent, nakayoshi_fork decorates the fork method: https://github.com/ko1/nakayoshi_fork/blob/19ef5efc51e0ae51d7f5f37a0b785309bf16e97f/lib/nakayoshi_fork.rb

## Proposals

I see two possible features to improve this situation:

**Fork callbacks**

One solution would be for Ruby to expose a callback API for these two events, similar to Kernel.at_exit.

Most implementations of this functionnality in other languages (C's pthread_atfork, Python's os.register_at_fork) expose 3 callbacks:

- prepare or before executed in the parent process before the fork(2)
- parent or after_in_parent executed in the parent process after the fork(2)
- child or after_in_child executed in the child process after the fork(2)

A direct translation of such API in Ruby could look like Process.at_fork(prepare: Proc, parent: Proc, child: Proc) if inspired by pthread_atfork.

Or alternatively each callback could be exposed idependently: Process.before_fork {}, Process.after_fork_parent {}, Process.after_fork_child {}.

Also note that similar APIs don't expose any way to unregister callbacks, and expect users to use weak references or to not hold onto objects that should be garbage collected.

Pseudo code:

```
module Process
  @prepare = []
  @parent = []
  @child = []

  def self.at_fork(prepare: nil, parent: nil, child: nil)
    @prepare.unshift(prepare) if prepare
# prepare callbacks are executed in reverse registration order
    @parent << parent if parent
    @child << child if child
  end

  def self.fork
    @prepare.each(&:call)
```

```
    if pid = Primitive.fork
      @parent.each(&:call) # We could consider passing the pid here.
    else
      @child.each(&:call)
    end
  end
end
```

**Make Kernel.fork a delegator**

A simpler change would be to just make Kernel.fork a delegator to Process.fork. This would make it much easier to prepend a module on Process for each library to implement its own callback.

Proposed patch: https://github.com/ruby/ruby/pull/4361

**Related issues:**

| | |
|---|---|
| Related to Ruby - Feature #5446: at_fork callback API | **Closed** |

### Associated revisions

**Revision 645616c273aa9a328ca4ed3fceac8705e2e036cd - 07/15/2021 07:46 AM - mame (Yusuke Endoh)**

process.c: Call rb_thread_atfork in rb_fork_ruby

All occurrences of rb_fork_ruby are followed by a call rb_thread_fork in
the created child process.

This is refactoring and a potential preparation for [Feature #17795].
(rb_fork_ruby may be wrapped by Process.*fork*.)

**Revision 645616c273aa9a328ca4ed3fceac8705e2e036cd - 07/15/2021 07:46 AM - mame (Yusuke Endoh)**

process.c: Call rb_thread_atfork in rb_fork_ruby

All occurrences of rb_fork_ruby are followed by a call rb_thread_fork in
the created child process.

This is refactoring and a potential preparation for [Feature #17795].
(rb_fork_ruby may be wrapped by Process.*fork*.)

**Revision 645616c2 - 07/15/2021 07:46 AM - mame (Yusuke Endoh)**

process.c: Call rb_thread_atfork in rb_fork_ruby

All occurrences of rb_fork_ruby are followed by a call rb_thread_fork in
the created child process.

This is refactoring and a potential preparation for [Feature #17795].
(rb_fork_ruby may be wrapped by Process.*fork*.)

**Revision 13068ebe32a7b8a1a9bd4fc2d5f157880b374e1d - 10/25/2021 11:47 AM - mame (Yusuke Endoh)**

process.c: Add Process._fork (#5017)

- process.c: Add Process._fork

This API is supposed for application monitoring libraries to hook fork
event.

[Feature #17795]

Co-authored-by: Nobuyoshi Nakada nobu@ruby-lang.org

**Revision 13068ebe32a7b8a1a9bd4fc2d5f157880b374e1d - 10/25/2021 11:47 AM - mame (Yusuke Endoh)**

process.c: Add Process._fork (#5017)

- process.c: Add Process._fork

This API is supposed for application monitoring libraries to hook fork
event.

[Feature #17795]

Co-authored-by: Nobuyoshi Nakada nobu@ruby-lang.org

**Revision 13068ebe - 10/25/2021 11:47 AM - mame (Yusuke Endoh)**

process.c: Add Process._fork (#5017)

- process.c: Add Process._fork

This API is supposed for application monitoring libraries to hook fork event.

[Feature #17795]

Co-authored-by: Nobuyoshi Nakada nobu@ruby-lang.org

**Revision aa5bccfc65cf47a10d72cefa4bc2ee097f135b4c - 11/08/2021 06:38 PM - mame (Yusuke Endoh)**

NEWS.md: Mention Process._fork [[Feature #17795]]

**Revision aa5bccfc65cf47a10d72cefa4bc2ee097f135b4c - 11/08/2021 06:38 PM - mame (Yusuke Endoh)**

NEWS.md: Mention Process._fork [[Feature #17795]]

**Revision aa5bccfc - 11/08/2021 06:38 PM - mame (Yusuke Endoh)**

NEWS.md: Mention Process._fork [[Feature #17795]]

## History

**#1 - 04/12/2021 01:24 PM - Dan0042 (Daniel DeLorme)**

> Most database clients, ORMs or other libraries keeping a connection pool might need to close connections before the fork happens.

Am I missing something? Afaik the proper way to do this is to close the connection *after* the fork. Of course if you do it before the fork it will work, but then the parent must reconnect, not to mention it aborts any transaction in progress. Maybe there's some confusion because a "full" disconnect in this child will send a disconnection message to the DB server, which will then close its connection and cause a lost connection in the parent process. But as long as you just disconnect the DB *socket* in the child without doing the full disconnection thing, the DB connection will keep working just fine in the parent.

> Unfortunately Process.pid is relatively slow on Linux

I did a benchmark on my Linux machine (Linux ubuntu 4.4.0-206-generic) and $$ is the same speed as 1==1. Process.pid is only marginally slower. I'd like to know where the idea that it's slow is coming from.

> Make Kernel.fork a delegator

This works great for before_fork, but after_fork needs to handle both the with-block and without-block forms. So everyone who justs want an after_fork hook will need to write some boilerplate code like below. So it seems to me this is not such a nice API.

```
def fork(*a, **kw, &b)
  if block_given?
    super(*a, **kw) do
      after_fork
      yield
    end
  else
    pid=super
    after_fork if !pid
    pid
  end
end
```

**#2 - 04/12/2021 02:12 PM - byroot (Jean Boussier)**

> Afaik the proper way to do this is to close the connection after the fork.

No before. Otherwise the connection is "shared" and closing it in the children cause issues for the connections in the parent.

> I'd like to know where the idea that it's slow is coming from.

Maybe your glibc is quite old? https://sourceware.org/glibc/wiki/Release/2.25#pid_cache_removal

```
require 'benchmark/ips'

module Foo
  class << self
    attr_accessor :bar
  end
  @bar = 42
end

puts "#{RUBY_VERSION} #{RUBY_PLATFORM}"
Benchmark.ips do |x|
  x.report('Process.pid') { Process.pid }
  x.report('Module.attr') { Foo.bar }
  x.compare!
end

3.0.1 x86_64-darwin20
Warming up --------------------------------------
         Process.pid     1.914M i/100ms
         Module.attr     1.775M i/100ms
Calculating -------------------------------------
         Process.pid     19.144M (± 0.7%) i/s -     97.626M in   5.099666s
         Module.attr     17.820M (± 0.4%) i/s -     90.530M in   5.080332s

Comparison:
         Process.pid: 19144498.7 i/s
         Module.attr: 17820085.3 i/s - 1.07x  (± 0.00) slower

3.0.1 x86_64-linux
Warming up --------------------------------------
         Process.pid   698.792k i/100ms
         Module.attr     1.886M i/100ms
Calculating -------------------------------------
         Process.pid      6.862M (± 1.5%) i/s -     34.940M in   5.092832s
         Module.attr     19.184M (± 1.0%) i/s -     96.197M in   5.014902s

Comparison:
         Module.attr: 19183904.4 i/s
         Process.pid:  6862219.4 i/s - 2.80x  (± 0.00) slower
```

So fast enough for things that are infrequently called, but slow enough that I see it sitting at 1-2% of CPU profiles in real production workloads.

> So it seems to me this is not such a nice API.

It's not really intended as an actual API, but as a smaller change that would be more easily accepted by the core team.

#### #3 - 04/12/2021 02:59 PM - Dan0042 (Daniel DeLorme)

byroot (Jean Boussier) wrote in #note-2:

> > Afaik the proper way to do this is to close the connection after the fork.
>
> No before. Otherwise the connection is "shared" and closing it in the children cause issues for the connections in the parent.

It seems quite easy to demonstrate otherwise:

```
# with Sequel and mysql2
DB["select rand()"].first #=> {:"rand()"=>0.1878050166999968}
m = DB.pool.available_connections.first
Process.wait fork{
  IO.for_fd(m.socket).close #without this the query below would fail
}
DB["select rand()"].first #=> {:"rand()"=>0.3590592307588637}
```

> Maybe your glibc is quite old? https://sourceware.org/glibc/wiki/Release/2.25#pid_cache_removal

That must be it. Still, I'm relieved we're only talking about 3x slower than the simplest/fastest ruby operations.

#### #4 - 04/13/2021 01:32 AM - mrkn (Kenta Murata)

Python provides os.register_at_fork API for registering callbacks which are called on both before and after fork.
Especially after-fork, it enables to register callbacks for parent and child processes, separately.
See https://docs.python.org/3/library/os.html#os.register_at_fork

#### #5 - 04/13/2021 02:32 AM - jeremyevans0 (Jeremy Evans)

The main issue I see with this is the potential for misuse. If I could be sure this would only be used by applications and not libraries, it seems reasonable. However, I suspect the main usage will be libraries, and it's not really possible to use this correctly in libraries. For example, this currently works:

```
Sequel.postgres.transaction do
  exit! if fork
end
```

So does this:

```
Sequel.postgres.transaction do
  exit! unless fork
end
```

If a library starts using a before_fork or after_fork hook to close connections automatically, it breaks at least one of the two cases. Automatically disconnecting in before_fork will break both cases, since the connection is disconnected before the fork. Automatically disconnecting in after_fork breaks at least the second case, and can break the first as well if it does a full cleanup instead of just closing the file descriptor of the connection socket.

While most cases where things break are not as clear cut as the above examples, they can and do exist.

Dan0042 (Daniel DeLorme) wrote in #note-3:

> byroot (Jean Boussier) wrote in #note-2:
>
> > Afaik the proper way to do this is to close the connection after the fork.
>
> No before. Otherwise the connection is "shared" and closing it in the children cause issues for the connections in the parent.
>
> It seems quite easy to demonstrate otherwise:
>
> ```
> # with Sequel and mysql2
> DB["select rand()"].first #=> {:"rand()"=>0.1878050166999968}
> m = DB.pool.available_connections.first
> Process.wait fork{
>   IO.for_fd(m.socket).close #without this the query below would fail
> }
> DB["select rand()"].first #=> {:"rand()"=>0.3590592307588637}
> ```

This is not a good example because it's not portable across adapters. You are relying on the database driver exposing the file descriptor (not all drivers do). Sequel doesn't have code that only closes sockets for connections, it calls a method that will actually clean up the connection state, such as rolling back active transactions, which when called on a connection in a child process will break later usage of the same connection in the parent process.

Sequel recommends calling Sequel::Database#disconnect before forking in the common case where you are dealing with a pool of worker processes. However, even if Ruby added a general before/after fork hook, Sequel would never use it automatically, since it is impossible to determine whether doing so is desired or not.

An additional issue is that not all forks are equal. Consider code such as:

```
DB = Sequel.postgres
DB.disconnect
pids = 4.times.map do
  fork do
    # worker process
    # ...
    if some_condition
      fork do
        do_something_in_worker_child
        exit!
      end
    end
```

```
  end
end
```

The idea with this proposal is you could simplify things by eliminating the explicit DB.disconnect call before the first fork.  However, doing so automatically could break things in a worker process if the second fork was called.

Another issue with this proposal is that in the above example, DB.disconnect is only called once when called explicitly, but would be called 4 times if called implicitly.

I understand why this is being requested, and I do think it could simplify common Ruby web application deployment scenarios.  However, even if requires additional explicit code, I think the current approach of per-server fork hooks is better.

#### #6 - 04/13/2021 10:52 AM - byroot (Jean Boussier)

@jeremyevans0 (Jeremy Evans) I understand your point, and I agree that in an ideal world this wouldn't be necessary.

However pragmatically speaking, getpid(2) checks are about as old as fork(2). Even the glibc changelog acknowledge it:

> Applications performing getpid() calls in a loop will see the worst case performance degradation as the library call will perform a system call at each invocation.

And suggest to use pthread_atfork() instead:

> Applications performing getpid() in a loop that need to do some level of fork()-based invalidation can instead use pthread_atfork() to register handlers to handle the invalidation.

I think it's a clear indication that this need is nothing new.

> The main issue I see with this is the potential for misuse.

I agree, but this argument could apply to a large part of Ruby, I'm not sure it's relevant.

#### #7 - 04/13/2021 11:16 AM - Eregon (Benoit Daloze)

@byroot (Jean Boussier) Thanks for the new issue, this seems much clearer than the previous one :)

jeremyevans0 (Jeremy Evans) wrote in #note-5:

> If a library starts using a before_fork or after_fork hook to close connections automatically, it breaks at least one of the two cases.

And if it doesn't it's data corruption if there is any code between the fork and exit! accessing the DB, which I would argue is far worse.

> Sequel recommends calling Sequel::Database#disconnect before forking in the common case where you are dealing with a pool of worker processes. However, even if Ruby added a general before/after fork hook, Sequel would never use it automatically, since it is impossible to determine whether doing so is desired or not.

Sequel could expose a method to enable/disable closing connections on fork.
Then it's only a matter of which is the default.

Do people actually fork in the middle of a transaction? That seems fairly unsafe to me and I don't see the point.
For the 0.1% people actually doing something like that, isn't it acceptable that they explicitly disable the hook via e.g. some Sequel API?
IMHO safety/security is much more important than supporting very rare edge cases without an extra line of code.
Anyway, Sequel is free to do what it wants, but clearly ActiveRecord already sides on the side of safety/security by default here. Is there any user complaining about that?

Closing before forking is suboptimal as the caller process will need to reconnect, and it might interrupt other threads  in that process connected to the DB needlessly.
I understand it might not be easy to close the connection in the child process, depending on the DB driver.
IMHO, it is worth closing in the child when the driver provides a way to do it.

#### #8 - 04/13/2021 11:21 AM - Eregon (Benoit Daloze)

*- Related to Feature #5446: at_fork callback API added*

#### #9 - 04/13/2021 02:36 PM - mame (Yusuke Endoh)

I'd like to make sure: what process calls the after_fork hook? Only parent, only child, or both?

In the previous ticket #5446, atfork_parent and atfork_child were discussed. Is the distinguishment not needed?

#### #10 - 04/13/2021 02:46 PM - Dan0042 (Daniel DeLorme)

> Do people actually fork in the middle of a transaction?

I think the most likely case is if the transaction is unrelated to the fork. Let's say your web backend handles every request inside a transaction. If somewhere in there you fork a child process in order to perform some asynchronous work, you don't want to close the DB connection.

Sequel.postgres.transaction{ exit! if fork } is if you want to start DB work in the parent and continue in the child. I find that vanishingly improbable.

> I understand it might not be easy to close the connection in the child process, depending on the DB driver.
> IMHO, it is worth closing in the child when the driver provides a way to do it.

I agree. Closing in the child is ideal; if the DB driver doesn't expose the socket then closing before the fork is usually a fine workaround.

#### #11 - 04/13/2021 03:17 PM - byroot (Jean Boussier)

> what process calls the after_fork hook? Only parent, only child, or both?

The child. But I agree that a better naming could be found to avoid that confusion. I don't have an idea just yet, but I'll think about it.

> Is the distinguishment not needed?

From the use cases I compiled I saw no use for a callback in the parent after a fork. But maybe that use case exist and I missed it. Or maybe it's justified to add it simply to look a bit more like pthread_atfork.

#### #12 - 04/13/2021 04:32 PM - mame (Yusuke Endoh)

@byroot (Jean Boussier) I see, thanks! I think that the intended behavior of the proposed APIs is valuable to be explained in the ticket description.

#### #13 - 04/13/2021 10:22 PM - byroot (Jean Boussier)

@mame (Yusuke Endoh) ~~I'm afraid I don't have the permission to edit my own tickets.~~ Nevermind.

#### #14 - 04/14/2021 06:06 AM - ko1 (Koichi Sasada)

For debugger, I also want to use.
I want to use 3 hooks like pthread_atfork.

#### #15 - 04/14/2021 07:31 AM - byroot (Jean Boussier)

*- Subject changed from `before_fork` and `after_fork` callback API to Around `Process.fork` callbacks API*

*- Description updated*

#### #16 - 04/14/2021 07:34 AM - byroot (Jean Boussier)

I updated the description with:

- Added the third hook (after fork in parent) on @ko1's demand.
- Added links to similar feature in C and Python.
- Added pseudo-code of the implementation.
- Added precision about not being able to unregister callback.
- Added precision on call order (first registered vs last registered).
- Added two proposed APIs.

#### #17 - 04/16/2021 08:03 AM - akr (Akira Tanaka)

Another idea is introducing Process.fork_level which can be used to detect fork instead of getpid.

```
Process.fork_level #=> 0
fork {
  Process.fork_level #=> 1
  fork {
    Process.fork_level #=> 2
  }
}
```

It doesn't need system call unlike getpid.

Also, it doesn't have the problem with reused pid (pid collision).

Process.fork_level can be used to detect process forking in a library API
and it can close persistent connections (DB connection, HTTP persistent connection, statistics server connection, etc.) at a forked child process.

I think that closing persistent connections at forked child processes is better than disconnect before fork and reconnect after fork at parent and child.
Because some fork uses, such as parallel computing using fork, doesn't need to disconnect.
Disconnecting at all fork invocation would cause needless disconnection, I think.

#### #18 - 04/16/2021 08:07 AM - byroot (Jean Boussier)

> It doesn't need system call unlike getpid.

It would be a win for sure, but in my opinion we might as well go all the way and have a callback triggered immediately upon fork rather than have the cleanup be delayed until the next access to whatever the protected resource is, as well as having to constantly check for that change.

#### #19 - 04/16/2021 01:28 PM - ivoanjo (Ivo Anjo)

At Datadog, we use our monkey patching (mentioned in the description) to restart background threads that are used to periodically gather metrics. We use this so that we can transparently gather (profiling) metrics on all processes in Ruby apps that employ forking.

Thus, "passive" solutions -- checking the pid / fork level -- don't work for our use case. Hence why we're really really interested in the Fork callbacks / Make Kernel.fork a delegator solution.

#### #20 - 04/17/2021 07:56 AM - mame (Yusuke Endoh)

This thicket was discussed in the dev meeting, but no conclusion was reached. The current situation is not so good, but some committers agreed with Jeremy's concern (easy to misuse). @akr (Akira Tanaka) proposed fork_level (as above). @matz (Yukihiro Matsumoto) said that he wanted to provide something helpful, but he was not sure what we should provide.

#### #21 - 04/17/2021 11:02 AM - byroot (Jean Boussier)

> some committers agreed with Jeremy's concern (easy to misuse).

I must admit I don't understand this argument. There are plenty of advanced Ruby feature that are easy to misuse but extremely useful. So much that I don't think it's even worth listing them.

> akr (Akira Tanaka) proposed fork_level (as above).

As answered above, in my opinion, any solution akin to checking PID before access is unsatisfactory because:

- It delays the cleanup to the next access, which in same case can be problematic.
- Even if it's fast it's still needless busywork. Many use cases for this feature are things such as tracing libraries that are called very often in hotspots.

> nobu: IO.popen("-") is also a fork

As said in the description, fork + exec is out of scope. This only aim at catching forks that will continue to execute the Ruby VM.

> mame: How about extension libraries? Maybe we don't have to care them?

Yes, we don't care about them, because as far as I know an extension calling fork(2) would leave the VM in an unusable state (let me know if I'm wrong). I looked at various servers etc, all of them use Process.fork or Kernel.fork, I couldn't find any example of a C-extension calling fork(2).

> mrkn: Python provides C-API to call registered hooks

It also provide a Python API. https://docs.python.org/3/library/os.html#os.register_at_fork

Could we at least settle on my delegation PR (https://github.com/ruby/ruby/pull/4361) if a proper API is not acceptable?

#### #22 - 04/17/2021 05:43 PM - mame (Yusuke Endoh)

byroot (Jean Boussier) wrote in #note-21:

> I must admit I don't understand this argument. There are plenty of advanced Ruby feature that are easy to misuse but extremely useful. So much

> that I don't think it's even worth listing them.

Sorry, I have to admit my ignorance in this area, so it is difficult for me to log the discussion of the meeting, but as far as I understand... Some committers said that an appropriate cleanup/restart process before/after fork depends on applications, not libraries (as Jeremy said). This API is apparently proposed just for libraries, so there seems to be nothing but misuse.

> > nobu: IO.popen("-") is also a fork
>
> As said in the description, fork + exec is out of scope.

We are sure that fork + exec is out of scope. But IO.popen("-") is not fork + exec but fork, surprisingly.

```
$ ruby -e 'IO.popen("-"); $stderr.puts "Hello"'
Hello
Hello
```

> > mame: How about extension libraries? Maybe we don't have to care them?
>
> Yes, we don't care about them, because as far as I know an extension calling fork(2) would leave the VM in an unusable state (let me know if I'm wrong). I looked at various servers etc, all of them use Process.fork or Kernel.fork, I couldn't find any example of a C-extension calling fork(2).

Thanks, I failed to log the answer to my question, but any committer (I forgot, maybe @ko1 (Koichi Sasada)?) said the same. I'm convinced.

> Could we at least settle on my delegation PR (https://github.com/ruby/ruby/pull/4361) if a proper API is not acceptable?

The proposal is not rejected yet. In the previous meeting, we talked about this ticket in about one hour, and @akr (Akira Tanaka), one of the most familiar committers with this area, said that we can provide fork_level to support "Continuously check Process.pid" approach if it works. He also said that it is not a perferct solution (fork_level does not satisfy ko1's need explained in Comment #14), but we tentatively agreed that he counter-propose fork_level to see if it is good enough or not. I guess matz will finally change something for this ticket, but we have not yet determined what is best. Sorry for your patience.

#### #23 - 04/19/2021 09:55 AM - byroot (Jean Boussier)

> Sorry, I have to admit my ignorance in this area, so it is difficult for me to log the discussion of the meeting

I wasn't complaining about your log, thanks for the work you do, it's extremely valuable for people like me.

I've read the original argument by Jeremy, and I get that some people consider libraries shouldn't be in control of such things.

But I (and apparently a bunch of other library writers, since clearly a bunch of libraries already do it, and other languages offer that capability) disagree with this stance, and don't understand how Ruby design should be influenced by this types of "proper way to do it" arguments.

> But IO.popen("-") is not fork + exec but fork, surprisingly.

My bad I overlooked the "-" and I wasn't aware of this feature. However looking at IO.popen implementation, it doesn't seem hard to support as well.

#### #24 - 04/21/2021 07:30 AM - sam.saffron (Sam Saffron)

There is some precedent @jeremyevans0 (Jeremy Evans) for library authors offering all the poisons to the users.

For example:

https://github.com/redis/redis-rb/blob/6542934f01b9c390ee450bd372209a04bc3a239b/lib/redis/client.rb#L384-L389

> inherit_socket: true: disable safety check that prevents a forked child from sharing a socket with its parent; this is potentially useful in order to mitigate connection churn when:
> many short-lived forked children of one process need to talk to redis, AND
> your own code prevents the parent process from using the redis connection while a child is alive
>
> Improper use of inherit_socket will result in corrupted and/or incorrect responses.

So for example if this feature were given to redis they could get rid of the pid check on connected and instead rely on the fork hooks provided by Ruby.

There is a certain appeal to a Redis option where you can fork as much as you want and all your redis calls continue to work. Even if library authors ask you to opt in to this behavior. (eg: Redis.auto_safe_fork!)

I agree this is a super sharp tool, but it does give lib authors the ability to ship an "easy mode".

#### #25 - 04/22/2021 01:21 PM - byroot (Jean Boussier)

Another instance I just stumbled upon, Dalli the dominant Memcached client checks Process.pid before every single command: https://github.com/petergoldstein/dalli/blob/c73e52bf2993877ad509938dd840f0544633e998/lib/dalli/server.rb#L210

#### #26 - 04/22/2021 02:34 PM - Dan0042 (Daniel DeLorme)

akr (Akira Tanaka) wrote in #note-17:

> Another idea is introducing Process.fork_level which can be used to detect fork instead of getpid.

If that's possible, then it should be equally possible to just [pre-]cache Process.pid and it would be just as performant, right? Whatever were the reasons for pid cache removal in glibc, they would also apply to fork_level. From what I understand above it's not relevant here because calling fork(2) would leave the VM in an unusable state. Since checking the pid is a pattern that already exists, imho it's better to make that pattern performant than to introduce a new pattern with almost no advantage. I don't think pid reuse/collision is a realistic concern either.

But checking either Process.pid or fork_level doesn't fix one of the biggest issues I've had with forking: *finalizers*. If the database object has a finalizer that closes the connection, you need to prevent that finalizer from ever executing in the child process. You can use fork{ work(); exit! } to prevent finalizers from running on exit, but that also prevents any finalizers that were defined in the child process. So you actually need to use fork{ at_exit{exit!}; work() }, but even then it doesn't cover the case where a finalizer is run when the object is garbage-collected. This is a real thorn, and to me it's the main reason why the socket must be closed in the child process right after fork.

byroot (Jean Boussier) wrote in #note-23:

> But I (and apparently a bunch of other library writers, since clearly a bunch of libraries already do it, and other languages offer that capability) disagree with this stance, and don't understand how Ruby design should be influenced by this types of "proper way to do it" arguments.

+1; for me I believe fork safety should be handled by libraries because of encapsulation. If you have independant libs, libA that needs to close sockets on fork, and libB that uses fork, these two should remain independant. You should not need the app to insert glue code in order for the two libs to play nice together. That really breaks encapsulation and separation of concerns. 2¢

> Make Kernel.fork a delegator

I have two concerns with this:

1. Some existing gems already override both Kernel#fork and Process.fork in order to setup callbacks; this change would cause the callbacks to be triggered twice when using Kernel#fork. So it's somewhat backward-incompatible.
2. You need to handle the with-block and without-block forms differently.

So I'd like to make a counter-proposal: what about introducing a separate method (let's say Process._fork_) which *doesn't accept a block* and is called by the various forking methods, including IO.popen("-"). That would solve the two concerns above, and it might even be simple enough to be backported.

#### #27 - 04/22/2021 03:23 PM - Dan0042 (Daniel DeLorme)

from dev meeting notes:

> akr: Process.daemon is very special, so it doesn't have to call the hooks (or update fork_level)

It should be noted that Process.daemon has the effect of stopping active threads, just like fork. Since one of the stated goals in this ticket is to "restart the thread in the forked children", I think callbacks are also relevant for Process.daemon. Or maybe don't stop active threads? At least it should be consistent.

#### #28 - 04/22/2021 06:58 PM - byroot (Jean Boussier)

> Or maybe don't stop active threads?

That's not possible. After fork(2) all but the thread that called fork(2) die in the children.

> So I'd like to make a counter-proposal: what about introducing a separate method (let's say Process.*fork*) which doesn't accept a block and is called by the various forking methods, including IO.popen("-"). That would solve the two concerns above

Very good idea. If the proper callback API is not accepted, but the idea of having a proper chokepoint is, then your proposal is better than mine.

> one of the biggest issues I've had with forking: finalizers.

That's a good point I didn't thought of. Undefining serializers is another use case.

#### #29 - 04/23/2021 04:26 PM - Eregon (Benoit Daloze)

I very much agree with @Dan0042's point about exit!, finalizers, at_exit and subtleties.
This can be easily be handled with fork hooks, and it's a total nightmare otherwise, to the point many apps/libraries probably get it wrong (e.g., exit a different way than exit! in the child, which e.g. an exception reaching the top-level is already an issue, ruby -e 'at_exit { p "hi" }; fork { raise "foo" }' prints twice)
or needlessly restrict what child processes can do with at_exit.

I also heavily agree with encapsulation, requiring the app to know about every library which might have some IO/Socket/resourced opened that's not naturally fork-safe and what to call on each of these libraries with a sever-specific hook is a terrible programming model.
It invites invalid sharing and data corruption, so I'd go as far as saying libraries not handling are basically guaranteeing troubles for their users every now and then if they use fork.
The redis-rb approach above makes a lot of sense: safe by default and can be tuned if desired for more dangerous cases where sharing is wanted.

#### #30 - 05/03/2021 11:34 AM - Eregon (Benoit Daloze)

Another use-case I found: https://github.com/rubyjs/mini_racer#fork-safety
That gem (and others of course) could provide some helpers methods based on the chosen strategy if there is a common hook.

Right now providing fork safety (if it has any resource that needs handling across fork) is basically impossible for a gem, which seems a severe limitation.
And we all know that if something is not fork-safe and fork is used it's often leading to data corruption.

#### #31 - 05/21/2021 05:44 PM - mame (Yusuke Endoh)

@matz (Yukihiro Matsumoto) and some committers discussed this ticket for about an hour in today's dev-meeting, but we didn't reach any conclusion again.

Mainly, @akr (Akira Tanaka) -san still agrees with @jeremyevans0 (Jeremy Evans) and feels a bad smell about the proposed hook API. It is difficult for me to summarize his opinion precisely, but my partial understanding is:

- The API looks difficult to use correctly. The example that Jeremy says in #note-5 (fork in a transaction) might look a bit artificial, but it may occur in parallel; one thread runs a transaction and the other one calls fork.
- We understood that fork_level was insufficient for datadog case. However, it is still a preferable solution in a case where PID checking or fork_level is sufficient. If we add a casual API to hook a fork event, people may choose and (mis)use it blindly.
- If something is introduced for this issue, the API should look clearly not casual, not easy to use. Process._fork_ that @Dan0042 proposed in #note-26 is a possible option. TracePoint.new(:before_fork) { ... }.enable { ... } or something may be another one.

Honestly, I have no idea how to proceed this issue. @jeremyevans0 (Jeremy Evans), do you still think that this proposal is not suitable? Do you have any idea about counterproposals?

#### #32 - 05/21/2021 09:23 PM - jeremyevans0 (Jeremy Evans)

I still believe neither of these proposals (fork callbacks or Kernel.fork delegation) should be accepted. In general, the proposals are for the same feature (wrapping behavior on fork), so in my opinion there is not much difference between them, and this response will treat them as being the same.

Here's what I consider the primary benefit of adding support for fork callbacks:

- They handle the common case easily with no code required by the library user.

Here's what I consider the primary cost of adding support for fork callback:

- They invoke the fork callbacks when they should not be invoked, because libraries cannot know the purpose of the fork in other libraries or in application code.

I don't want to understate the benefit, language-level fork callbacks make certain common cases easier, and that is a valuable goal. However, I consider the cost too high.

Without reiterating my previous points too much, not all forks are the same. A fork made by a webserver library is different than a fork made in application code. With language-level fork callbacks, the callback has no context about why the fork is happening, and therefore cannot make appropriate decisions.

With the current situation, each library that uses fork generally implements their own callbacks around the library's use of fork, and users use those library-specific callbacks as appropriate to handle resources (e.g. closing sockets). This results in no problems when library-specific callbacks are used correctly. Almost all of the problems in this area occur when the library-specific callbacks should be used and are not.

What will likely happen if this is implemented is that libraries will no longer implement fork callbacks, instead telling users to use language-level fork callbacks. Worse, libraries that create resources (e.g. opening sockets) will start to automatically use the language-level fork callbacks to close the

resources on fork, regardless of whether doing so makes sense for the application using the library. This will lead to a situation where the common case is easier but the more complex case is broken completely with no ability to be fixed. If we are lucky, the libraries that create resources will offer a way to opt out of using the callbacks automatically. All of this is speculation on my part, but that is the future I foresee if this proposal is accepted.

In terms of Sequel, since it has been mentioned, only disconnecting before fork is considered safe. Disconnecting after fork is not safe as forks could be operating on the same connection, and Sequel may not have direct access to the underlying socket to close (Sequel supports 10+ adapters built-in, and more in external gems). Disconnecting before fork can only be done safely if you are sure nothing else is using the connections, such as before a web server forks child processes. At any other point, automatic disconnection before fork would be terrible, as the connections could be in use by other threads.

Another, more minor issue, is that a language-level before fork callback will necessarily be called on every fork, whereas current library-specific before fork callbacks are generally only called once before forking multiple times.

I think the problem does not have a good solution. However, in my opinion, the current situation is the least worst. I don't think libraries should try to detect forks using Process.pid or Thread.alive? or fork_level. Libraries should document how they should be used with fork, and should rely on the user correctly disconnecting before or after fork. Users that do not do so will likely have problems. To paraphrase Farquaad, "Some processes may die, but it's a sacrifice I am willing to make". :)

#### #33 - 06/02/2021 07:46 AM - ivoanjo (Ivo Anjo)

Since there seems to be no agreement on what a better and high-level API would look like, would it be reasonable to go back to the "Make Kernel.fork a delegator" proposal from https://github.com/ruby/ruby/pull/4361?

That patch seems like low-hanging fruit that improves the status quo for almost no added complexity, and also doesn't stop or conflict with the future introduction of better APIs.

Better yet, by having a single point-of-entry for extension, it simplifies all of the hacks out there, and can even be used as a building block for a common community implementations of some of the more higher-level approaches proposed above.

#### #34 - 06/02/2021 08:40 AM - byroot (Jean Boussier)

Agreed, but I think @Dan0042's Process._fork_ proposal is better.

#### #35 - 07/15/2021 07:18 AM - matz (Yukihiro Matsumoto)

I considered the idea of overriding _fork_ method. The idea itself sounds reasonable but _fork_ is not a good name.
It should be a more descriptive name, forking a new child process, expecting overriding. Any idea?

Matz.

#### #36 - 07/15/2021 07:29 AM - mame (Yusuke Endoh)

Discussed at today's dev-meeting.

- @akr (Akira Tanaka) agreed with the API design of Process._fork_
- As above, @matz (Yukihiro Matsumoto) disliked the name _fork_.
- I proposed __fork__, but matz dislike it too. The message of the names __send__ and __id__ is that a user MUST NOT override them, but __fork__ is supposed to be overridden, so the message is completely opposite.
- matz proposed sysfork, but akr disagreed because the method is not a bare wrapper of fork(2)
- We failed to find any reasonable name for the method. Please propose other good name candidates.

#### #37 - 07/15/2021 07:47 AM - mame (Yusuke Endoh)

*- Status changed from Open to Closed*

Applied in changeset git|645616c273aa9a328ca4ed3fceac8705e2e036cd.

---

process.c: Call rb_thread_atfork in rb_fork_ruby

All occurrences of rb_fork_ruby are followed by a call rb_thread_fork in
the created child process.

This is refactoring and a potential preparation for [Feature #17795].
(rb_fork_ruby may be wrapped by Process.*fork*.)

#### #38 - 07/15/2021 08:15 PM - byroot (Jean Boussier)

@mame (Yusuke Endoh) did you mean to close the ticket?

#### #39 - 07/15/2021 08:40 PM - byroot (Jean Boussier)

> Please propose other good name candidates.

Maybe Process.forkpid?

#### #40 - 07/16/2021 06:07 AM - mame (Yusuke Endoh)

*- Status changed from Closed to Open*

@byroot (Jean Boussier) Oh no! Reopening. Sorry for the noise.

#### #41 - 10/17/2021 07:39 PM - Dan0042 (Daniel DeLorme)

Here's hoping that a name, any name, will get matz' approval so this can make it into ruby 3.1

Process._fork
A name begining with an underscore is often used to indicate an internal/private method.

Process.fork!
A bang to communicate "be careful" when overriding this method.

Process.wrap_fork
Process.around_fork
Focus on the purpose of the overridden method; when we do def wrap_fork we are defining a method that wraps the fork behavior. Similar to around_filter in Rails, in a sense.

RubyVM.fork
Process::Wrap.fork
"fork" is really the most appropriate name for this method, so instead we communicate "this is a special case for overriding" via the module name. This goes back to @znz's suggestion of Process::Fork.fork which I find quite good.

#### #42 - 10/25/2021 02:36 AM - matz (Yukihiro Matsumoto)

The new method should be override (or wrap) target, so wrap_ or around_ are not a part of proper names.
I propose Process._fork for the method name, since:

- It is not for typical users
- It is the target for overriding
- We already have internal methods names like _dump, etc.

Matz.

#### #43 - 10/25/2021 04:18 AM - mame (Yusuke Endoh)

@matz (Yukihiro Matsumoto) Thanks. I've created a prototype PR: https://github.com/ruby/ruby/pull/5017

#### #44 - 10/25/2021 10:03 AM - ioquatix (Samuel Williams)

I agree with Jeremy, but I'll go a step further. If we want Ruby to be safe, the only sane way to handle fork (in general) is to close all resources in the child process (as already happens with Thread instances, and similarly with close on exec style behaviour). Any other model requires cooperation between the parent and child process, as Jeremy outlines and simply has to involve a shared understanding of what happens before and after fork in a highly application specific way.

I feel like adding an at_fork or after_fork hook should be sufficient for 99% of use cases, however we should allow users to bypass it, as in:

```
after_fork do
  connection_pool.close_on_fork
end

connection = connection_pool.acquire
connection.close_on_fork = false

fork do
  connection.query(...) # connection is still valid
end
```

However this kind of usage really seems like the edge of the edge cases and it begs the question, do we want to encourage libraries and applications to do this?

The more valid example was found by digging into the Datadog profiler:

```
after_fork do
  Datadog.profiler.start if Datadog.profiler
end
```

This to me seems like a totally legitimate use case but I don't think this needs _fork and I think it makes more sense to be consistent with at_exit, e.g. at_fork could be a better name, only executed in the child process. Wrapping Process#_fork seems very specific to how the Process module is

implemented and I don't think we should be exposing that or expecting users to prepend to standard library modules to get functionality. It sets a bad precedent IMHO.

#### #45 - 10/25/2021 11:55 AM - mame (Yusuke Endoh)

*- Status changed from Open to Closed*

Applied in changeset git|13068ebe32a7b8a1a9bd4fc2d5f157880b374e1d.

---

process.c: Add Process._fork (#5017)

- process.c: Add Process._fork

This API is supposed for application monitoring libraries to hook fork event.

[Feature #17795]

Co-authored-by: Nobuyoshi Nakada nobu@ruby-lang.org