Ruby - Feature #2619

Proposed method: Process.fork_supported?

01/21/2010 12:30 AM - hongli (Hongli Lai)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

=begin
It used to be that Process.fork is available on all POSIX platforms. With Ruby 1.9 that is no longer the case thanks to the fact that pthreads and fork cannot be combined on some platforms (CANNOT_FORK_WITH_PTHREAD). This makes it difficult to determine whether the current Ruby interpreter supports forking without actually calling fork.

I propose a method Process.fork_supported? which returns whether fork is supported on the current platform. See attached patch.

I realize that Process.spawn is the recommended way on Ruby 1.9 to spawn subprocesses, but there are times when forking without exec() is more efficient. I want to be able to use fork on platforms where it is fully supported, falling back to Process.spawn otherwise.

Specific use case:
Phusion Passenger is a Ruby web application server. It tries to conserve memory and reduce startup time by preloading an application's source code into a process, then forking multiple child processes that act as worker processes. This way spawning N worker instances of an application only takes L + N * F time instead of N * (L + F) time, where:

- L = the time it takes to load the source code
- F = the time it takes to fork a process
  =end

History

#1 - 01/21/2010 12:35 AM - hongli (Hongli Lai)

- File 0001-Add-method-Process.fork_supported-for-checking-wheth.patch added

=begin
Oops, I attached the wrong patch: the first one contains some local changes which I did not intend to include. Here's the correct patch.
=end

#2 - 01/21/2010 08:51 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

=begin
use defined?(fork) instead.
=end

#3 - 01/21/2010 09:33 AM - luislavena (Luis Lavena)

=begin
Hmn, defined?(fork) is not very useful:

```
C:\Users\Luis>pik ruby -e "puts defined?(fork)"
IronRuby 0.9.3.0 on .NET 2.0.0.0

nil

jruby 1.4.0 (ruby 1.8.7 patchlevel 174) (2009-11-02 69fbfa3) (Java HotSpot(TM) Client VM 1.6.0_17) [x86-java]

method

ruby 1.8.6 (2009-06-08 patchlevel 369) [i386-mswin32]

method

ruby 1.8.6 (2009-08-04 patchlevel 383) [i386-mingw32]

```

```
method

ruby 1.8.7 (2009-12-24 patchlevel 248) [i386-mingw32]

method

ruby 1.8.7 (2009-12-24 patchlevel 248) [i386-mswin32]

method

ruby 1.9.1p376 (2009-12-07 revision 26041) [i386-mingw32]

method

ruby 1.9.1p376 (2009-12-07 revision 26041) [i386-mswin32]

method

ruby 1.9.2dev (2010-01-02 trunk 26229) [i386-mingw32]
```

Yes, 1.9.2 returns nothing, not even nil

That means that checking for fork definition on any version as condition will make that code execute.

=end

#4 - 01/21/2010 01:13 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Thu, 21 Jan 2010 09:36:33 +0900,
Luis Lavena wrote in [ruby-core:27643]:

> C:\Users\Luis>pik ruby -e "puts defined?(fork)"
> ruby 1.9.2dev (2010-01-02 trunk 26229) [i386-mingw32]
>
> Yes, 1.9.2 returns nothing, not even nil

In 1.9, #puts no longer deals with nil specially. Try p.

> That means that checking for fork definition on any version
> as condition will make that code execute.

Anyway, neither can help you in older versions.

--
Nobu Nakada

=end

#5 - 01/21/2010 01:17 PM - luislavena (Luis Lavena)

=begin
Yikes, not even the old puts can save or be consistent anymore :-P

Thank you Nobu for enlighten me.

--
Luis Lavena

=end

#6 - 01/21/2010 06:44 PM - hongli (Hongli Lai)

=begin
#fork is defined in all versions of MRI on all platforms, even on platforms where it's not supported. It's just that calling #fork will raise NotImplementedError. Consider this snippet from process.c:

#if defined(HAVE_FORK) && !defined(CANNOT_FORK_WITH_PTHREAD)
...

```
static VALUE
rb_f_fork(VALUE obj)
{
...fork code here...
}
#else
#define rb_f_fork rb_f_notimplement
#endif

...

void
Init_process(void)
{
rb_define_virtual_variable("$?", rb_last_status_get, 0);
rb_define_virtual_variable("$$", get_pid, 0);
rb_define_global_function("exec", rb_f_exec, -1);
rb_define_global_function("fork", rb_f_fork, 0);
```

As you can see there's no #ifdef around rb_define_global_function("fork", ...).

I want to have a way to check whether rb_f_fork is implemented, without calling it and catching NotImplementedError.
=end

#7 - 01/21/2010 06:56 PM - hongli (Hongli Lai)

=begin
Tanaka Akira said that respond_to? returns false for methods that are not implemented since 1.9.2. I did not know that, but this solves my issue. This can be closed now.
=end

#8 - 01/21/2010 08:06 PM - vvs (Vladimir Sizikov)

=begin
On Thu, Jan 21, 2010 at 11:03 AM, Hongli Lai hongli@plan99.net wrote:

> On 1/21/10 10:53 AM, Tanaka Akira wrote:
>
>> respond_to? returns false for methods which is not implemented since 1.9.2.
>>
>> See NEWS.
>
> Ah, that is good news. :) You can close my issue then, sorry for the confusion.

Whoa! This is rather unexpected. And seems to be only adding to the confusion.
Other Ruby impls might not behave like this, not to mention that this
change of respond_to?
behavior only affects built-in methods. For regular methods, it won't work.

So, we end up with situation that for some non-implemented methods
respond_to? would return true,
for some other it would return false.

Thanks,
--Vladimir

=end

#9 - 01/22/2010 03:49 AM - vvs (Vladimir Sizikov)

=begin
On Thu, Jan 21, 2010 at 10:53 AM, Tanaka Akira akr@fsij.org wrote:

> 2010/1/21 Hongli Lai hongli@plan99.net:
>
>> This always returns true even on platforms where fork is not actually supported. See
>> my Redmine reply.
>
> respond_to? returns false for methods which is not implemented since 1.9.2.

I should probably also add that this is close to impossible to

implement on JRuby (and I assume on IronRuby too, and maybe some
others). In case of JRuby, we build once and then the resulting
product can be used on various platforms, so those platform specific
things are queried at runtime. In some cases we just invoke a native
functions via FFI at runtime and only then would see
NotImplementedError out of those calls. So, basically, untill the
method is executed, we can't really know whether it is supported or
not on some platforms.

So, this new behavior is not "portable" in a sense that it won't work
on all 1.9.2-compatible implementations, which would only add to
confusion.

Thanks,
--Vladimir

=end

#10 - 01/22/2010 04:09 AM - headius (Charles Nutter)

=begin
On Thu, Jan 21, 2010 at 12:49 PM, Vladimir Sizikov vsizikov@gmail.com wrote:

> I should probably also add that this is close to impossible to
> implement on JRuby (and I assume on IronRuby too, and maybe some
> others). In case of JRuby, we build once and then the resulting
> product can be used on various platforms, so those platform specific
> things are queried at runtime. In some cases we just invoke a native
> functions via FFI at runtime and only then would see
> NotImplementedError out of those calls. So, basically, untill the
> method is executed, we can't really know whether it is supported or
> not on some platforms.
>
> So, this new behavior is not "portable" in a sense that it won't work
> on all 1.9.2-compatible implementations, which would only add to
> confusion.

I agree. Unless we dynamically go through all "potentially
unimplemented" methods in the system and attach ahead-of-time tests to
them, JRuby will not be able to support this. There is no
system-specific compile time for JRuby, and as Vladimir mentions, we
can't know what system we're going to run on until we start running.

It also seems very confusing to me that a method would show up in
method lists, but respond_to? would produce false. I think this should
always be true:

assert obj.respond_to? X if obj.methods.include X

For handling this fork case (and similar cases), there are other solutions:

- Hongli's suggested fork_supported?, though that pollutes some
  namespace for every such method we want to check
- Just don't implement the method if it's not implemented. This
  doesn't help cases like in JRuby, where we may not know until runtime
  if we can support a method or not.
- Provide another mechanism for querying specific features, like
  Platform.support? :fork

As Ruby moves forward, it would be helpful if everyone remembered that
MRI is not the only implementation anymore, and changes impact all the
implementations in different ways.

- Charlie

=end

#11 - 01/22/2010 12:39 PM - luislavena (Luis Lavena)

=begin
On Thu, Jan 21, 2010 at 9:57 PM, Hongli Lai hongli@plan99.net wrote:

> [...]

> - Provide another mechanism for querying specific features, like Platform.support? :fork
>
> How is this better than calling the method and catching NotImplementedError? Your check method, although returning a boolean, still calls the method under the hood. This makes #support? an expensive call which is something I wanted to avoid with #fork_supported?.

I might be able to answer that.

While experimenting with Rubinius and MinGW, I've created a C library
that collected at compile time the related information of the
platform, and exposed it as CAPI extension.

That can, using extconf, #define, #ifdef etc the functionality exposed
by the platform you're running on.

In the case of JRuby, that Platform module can evaluate the underlying
OS capabilities and return you true/false without the need of
triggering NotImplementedError

At the end, I believe both fork_supported? and Platform.support? :fork
aim at the same, but the later can serve as API for other
functionality, like symlink, hardlinks, etc.

Just a thought.

PS: I propose we paint the bike shed red.

Luis Lavena
AREA 17

Perfection in design is achieved not when there is nothing more to add,
but rather when there is nothing more to take away.
Antoine de Saint-Exupéry

=end

#12 - 01/22/2010 02:27 PM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:27643] [Feature #2619] Proposed method: Process.fork_supported?"
on Thu, 21 Jan 2010 09:36:33 +0900, Luis Lavena redmine@ruby-lang.org writes:

|Hmn, defined?(fork) is not very useful:

We added a new behavior of defined?/respond_to? to 1.9 for exact same
purpose in more generic way. We are not going to add redundant
feature. If you want to stick with 1.8, JRuby, IronRuby or whatever,
you have to persuade their maintainers.

```
        matz.
```

=end

#13 - 01/22/2010 05:27 PM - headius (Charles Nutter)

=begin
On Thu, Jan 21, 2010 at 9:43 PM, Tanaka Akira akr@fsij.org wrote:

> In general, I recommend programs call such methods simply and rescue
> NotImplementedError. This style should work with various Ruby including
> Ruby 1.8 and JRuby.
>
> However there are requests for testing the availability of the methods without
> calling them. It is reasonable if the methods have unavoidable side-effects.
> This respond_to? behavior is for such requests.
>
> If we take the reason "we can't really know whether it is supported or
> not on some platforms.", we cannot provide a solution for the above

> requests.
>
> However I know the reason is actually true even in C.
> For example, new Linux system call is provided by glibc but
> Ruby runs on older Linux, the system call causes ENOSYS.
> We can't really know the system call is supported or not on the running system.
>
> So I recommend "just call and rescue" if possible.

So then in general, the new functionality is useless and should not be
used, since even for the C implementation you can't know if a method
will fail to invoke successfully. Should an unreliable or useless
feature ever be added?

- Charlie

=end

#14 - 01/22/2010 05:28 PM - headius (Charles Nutter)

=begin
On Thu, Jan 21, 2010 at 11:27 PM, Yukihiro Matsumoto matz@ruby-lang.org wrote:

> We added a new behavior of defined?/respond_to? to 1.9 for exact same
> purpose in more generic way. We are not going to add redundant
> feature. If you want to stick with 1.8, JRuby, IronRuby or whatever,
> you have to persuade their maintainers.

Except that as Akira pointed out, even on the C impl you can't know
whether the system itself has stubbed out certain functionality and
won't actually provide it. So everyone that expects respond_to? in
1.9+ to produce a definitive result will still be mistaken. What have
we gained?

- Charlie

=end

#15 - 01/22/2010 05:36 PM - headius (Charles Nutter)

=begin
On Thu, Jan 21, 2010 at 9:39 PM, Luis Lavena luislavena@gmail.com wrote:

> On Thu, Jan 21, 2010 at 9:57 PM, Hongli Lai hongli@plan99.net wrote:
>
>> [...]
>>
>>> - Provide another mechanism for querying specific features, like
>>>   Platform.support? :fork
>>
>> How is this better than calling the method and catching NotImplementedError? Your
>> check method, although returning a boolean, still calls the method under the hood.
>> This makes #support? an expensive call which is something I wanted to avoid with
>> #fork_supported?.
>
> I might be able to answer that.
>
> While experimenting with Rubinius and MinGW, I've created a C library
> that collected at compile time the related information of the
> platform, and exposed it as CAPI extension.
>
> That can, using extconf, #define, #ifdef etc the functionality exposed
> by the platform you're running on.
>
> In the case of JRuby, that Platform module can evaluate the underlying
> OS capabilities and return you true/false without the need of
> triggering NotImplementedError
>
> At the end, I believe both fork_supported? and Platform.support? :fork
> aim at the same, but the later can serve as API for other
> functionality, like symlink, hardlinks, etc.

Yes, you understand...this is why I proposed it. The general idea is
similar to the RUBY_ENGINE debates: a way to provide a "feature
enumeration" for users.

Using this code:

Platform.supported? :fork

...gives us the opportunity to handle specific keys in specific ways
at runtime rather than having to attach special behavior to all such
methods *at compile time*. And it does not require that we call fork,
since we can at least know some subset of names people are expecting
to have.

Based on what Tanaka has described about the C implementation, we
could comfortably implement respond_to? to always return true if we
define the method, regardless of whether it would raise an error or
not. Users should expect to rescue errors even if MRI does respond_to?
various methods, since the system itself may have stubbed them out to
produce errors. I'm not sure how the new behavior is actually useful.

- Charlie

=end

#16 - 01/22/2010 09:56 PM - vvs (Vladimir Sizikov)

=begin
Hi,

On Fri, Jan 22, 2010 at 9:51 AM, Tanaka Akira akr@fsij.org wrote:

> 2010/1/22 Charles Oliver Nutter headius@headius.com:
>
>> So then in general, the new functionality is useless and should not be
>> used, since even for the C implementation you can't know if a method
>> will fail to invoke successfully. Should an unreliable or useless
>> feature ever be added?
>
> I (and Vladimir) don't say that it always impossible.
>
> There are cases which is possible.

Yes, it seems that there are cases when this is possible. But the
problem is that this is not reliable and "portable" (across platforms
and across different implementations) in general case. So this looks
more like a hint rather than a reliable way to check whether some
method is implemented/functional or not. At the end of the day any
robust code still needs to take care of NotImplementedError and
whatnot.

Personally, I think that changing a public API and introducing this
complexity (and special cases like this *are* an additional
complexity) just to provide some 'hint' is overkill.

I don't think we could do that for JRuby (it will require lots and
lots of changes, ugly special-cases handling for respond_to?, and
still not be 100% reliable), so I just hate to see an incompatibility
where it could have been avoided.

Thanks,
--Vladimir

=end

#17 - 01/23/2010 04:25 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:27684] Re: [Feature #2619] Proposed method: Process.fork_supported?"
on Fri, 22 Jan 2010 17:27:45 +0900, Charles Oliver Nutter headius@headius.com writes:

|Except that as Akira pointed out, even on the C impl you can't know
|whether the system itself has stubbed out certain functionality and
|won't actually provide it. So everyone that expects respond_to? in
|1.9+ to produce a definitive result will still be mistaken. What have
|we gained?

As far as I understand, he only said there are some cases whether the
availability of a feature can only be known at runtime, not impossible
to implement 1.9 behavior. So I want to make your opinion clear. You
don't like the implementation complexity to implement 1.9 respond_to?
behavior, or something else?

```
      matz.
```

=end

#18 - 01/23/2010 12:06 PM - headius (Charles Nutter)

=begin
On Fri, Jan 22, 2010 at 1:25 PM, Yukihiro Matsumoto matz@ruby-lang.org wrote:

> As far as I understand, he only said there are some cases whether the
> availability of a feature can only be known at runtime, not impossible
> to implement 1.9 behavior. So I want to make your opinion clear. You
> don't like the implementation complexity to implement 1.9 respond_to?
> behavior, or something else?

There would definitely be a lot of added complexity to implement this
behavior in JRuby, since we don't know until runtime for *most*
methods whether they'll be supported or not. We build once at release
time, and then the same binary is used on many different platforms.

I would not argue against a feature just because of complexity,
however. In this case, it seems that users will still need to expect
that different platforms might respond_to? a method but still raise
errors at runtime. I don't feel that the added complexity (present but
less in 1.9, more in JRuby/IronRuby) is worth it.

It also seems like JRuby could simply lease 1.8 behavior in place,
since users may have to rescue errors on all implementations anyway.
Have I misunderstood?

Perhaps you could describe what you think the new behavior is supposed
to be? I'm confused now:

Which of these are correct? (referring to the default respond_to?, not
custom ones)

A: respond_to? should return true iff a method is bound on an object's
class or superclasses (i.e. will not raise NoMethodError)
B: respond_to? should return true iff (A) and the method will never
raise NotImplementedError on any platform
C: respond_to? should return true iff (A) and the method will never
raise NotImplementedError on the current platform
D: respond_to? should return true iff (C) and the method will not
raise system-level "not implemented" errors (like ENOTIMPL or ENOSYS)

Case A is 1.8 behavior and easy to support. If it's there, true; if not, false.

Case B is also pretty easy; we'd have to flag methods we know we have
hardcoded to raise NotImplementedError, and respond_to? would check
that flag. It only works for methods that are never implemented on any
platform, however, which makes me wonder why they're there in the
first place.

Case C is hard; we would have to have special-cased code for each such
method to check at runtime if the current platform supports it, and
make a best guess from there. This logic would have to run for every
new process.

Case D is probably not possible without a very large amount of work
and special-casing at compile time or runtime for all platforms.

I'm also curious how this affects Windows behavior, since currently

MRI stubs out many methods to return nil on Windows (like all of 'etc'
library, for example). Should they raise NotImplementedError? Should
they return false for respond_to?.

- Charlie

=end

#19 - 01/24/2010 05:56 AM - headius (Charles Nutter)

=begin
On Sat, Jan 23, 2010 at 1:50 PM, Caleb Clausen vikkous@gmail.com wrote:

> I don't know much about jruby, but I can't see why you say this is so
> difficult. It seems to me that fork_supported? (or whatever it should
> be called) could be implemented like this:

Here's fork_supported? on JRuby:

def Process.fork_supported?; false; end

There's no way to fork a JVM properly; too many signal handlers and
threads that just don't propagate properly. In fact, I'd *prefer*
fork_supported? over the respond_to? behavior change.

> Does that not work right in jruby? In the general case, there are many
> such system methods that need to be checked; you wouldn't want to
> repeat the above for each one. But it can be generalized and dry'd up
> a little:
>
> ```
> module Platform
>  SYSCALLNAME2TEST={
>    :fork => "!!fork{}",
>    :ppid => "res=Process.ppid; res and res !=0",
>    #etc
>  }
>
>  eval SYSCALLNAME2TEST.map{|name,test|
>     "
>     def #{name}_supported?
>       return @@#{name}_supported unless @@#{name}_supported==nil
>      @@#{name}_supported= (#{test})
>      return @@#{name}_supported
>    rescue Exception
>      return @@#{name}_supported=false
>    end
>    "
>  }.join("\n")
> end
> ```

Except that calling many methods will have side effects we don't want
to happen, or may have overcomplicated tests, or may not be testable
at all and you don't know until you call them with valid inputs if
they'll work. The general case is basically the halting problem...you
can't determine from inspection whether calling a given
method/function will behave well enough to claim it is "implemented".

> Essentially, you're doing the same kind of tests that configure does,
> but at runtime instead of compile time. Maybe you have to do a
> different special test for each system call, but that's one line, or
> at most a few, for each one. So in all we're talking about, what,
> several hundred lines of (ruby) code? I don't see how any of this is
> difficult. Tedious, perhaps, but not that difficult. I wouldn't even
> characterize it as "a very large amount of work".

Great, write it for all the system calls that 1.9 reports respond_to?
as false for :) And make sure it works correctly on all the platforms
where people use JRuby. You do have AIX, zLinux, and OpenVMS systems
available, right?

- Charlie

=end

#20 - 01/24/2010 06:13 AM - headius (Charles Nutter)

=begin
On Fri, Jan 22, 2010 at 9:06 PM, Charles Oliver Nutter
headius@headius.com wrote:

> I would not argue against a feature just because of complexity,
> however. In this case, it seems that users will still need to expect
> that different platforms might respond_to? a method but still raise
> errors at runtime. I don't feel that the added complexity (present but
> less in 1.9, more in JRuby/IronRuby) is worth it.

I want to make it clear I'm not just trying to be difficult. Currently
I don't see a simple way for us to observe the new respond_to?
behavior. We know for a fact there are many methods which we bind
regardless of platform because we can't know without calling them if
they're available. If we can't do it simply in JRuby, we won't do
it...and that will mean that the new respond_to? behavior is
ultimately unreliable across implementations. If it's unreliable, it's
no longer useful.

We are certainly open to suggestions on how to implement it. We would
also like to see a specification for how the behavior is supposed to
behave and which methods one might expect to exhibit the behavior.

A quick inspection of 1.9 shows the feature is not even consistent
there. The 'etc' module, for example, still returns nil for any
methods that are not available, and does not appear to respond_to? =>
false for any of them.

For the fork case, I think it best for us to not bind the method at
all. There's no good reason for us to bind it, knowing the JVM can't
fork, and so it will respond_to? => false for both 1.8 and 1.9 modes.
If there are other trivial cases, we will probably do the same. But I
don't see how we can guarantee that respond_to? => true will
accurately reflect that a method/function is available on the current
system, and for now we will not support that behavior. If the method
is bound, it will respond_to? => true on JRuby.

- Charlie

=end

#21 - 01/24/2010 06:45 AM - headius (Charles Nutter)

=begin
On Sat, Jan 23, 2010 at 3:27 PM, Tanaka Akira akr@fsij.org wrote:

> 2010/1/24 Charles Oliver Nutter headius@headius.com:
>
>> For the fork case, I think it best for us to not bind the method at
>> all. There's no good reason for us to bind it, knowing the JVM can't
>> fork, and so it will respond_to? => false for both 1.8 and 1.9 modes.
>
> It breaks code which rescue NotImplementedError.
>
> If no fork method, calling fork raise NoMethodError.
> rescue NotImplementedError don't rescue NoMethodError.

Yes, that's certainly true. But etc methods don't raise anything and
don't work either. UNIX socket doesn't even get defined or bound on
Windows, resulting in a NameError when you try to access it. There's
no consistency to any of it.

I did try removing fork, and tests in MRI 1.8 and RubySpec both expect
it to be there, to be private, and to raise NotImplementedError when
called if it's not implemented. So you're right, we can't just remove
it.

- Charlie

- Charlie

=end

#22 - 01/24/2010 07:35 PM - vvs (Vladimir Sizikov)

=begin
Hi,

On Sun, Jan 24, 2010 at 5:00 AM, Tanaka Akira akr@fsij.org wrote:

> 2010/1/22 Tanaka Akira akr@fsij.org:
>
>> FFI cannot test function availability?
>
> It seems attach_function in FFI raises FFI::NotFoundError if the specified
> function is not found.
>
> I don't understand why the detection is not reliable enough.

One of the reasons you've mentioned earlier: ENOSYS on POSIX systems
and E_NOTIMPL/ERROR_CALL_NOT_IMPLEMENTED on Windows. So the mere
presence of the method is not a guarantee that it will actually work
on particular system.

Second, MRI behavior treats in such special way only C-level methods,
while all the Ruby-level methods that raise NotImplementedError are
treated differently. So the users of the API need to know this
low-level/implementation detail in order to understand which method
could be checked with respond_to? and which couldn't.

Third, libffi itself is not supported on all platforms/architectures
where JRuby runs. For those cases we can't use it.
And, in some cases/environments the use of native support in JRuby is
prohibited and we can't use it.

Less important, but still worth mentioning:

Also, not all not-implemented features depend on underlying platform,
some are just can't be implemented sanely on JRuby, that's yet another
special case to handle.

JRuby uses jnr-posix 3rd party library (Java Native Runtime - POSIX),
it doesn't provide the capability to check which method is implemented
and which is not. Not to mention that there is a feature to map posix
method names to local platform method names at *runtime*, which
further complicates the matters, since until you call the method, you
don't know which native method will be actually called. And in some
cases one POSIX-like method is implemented via multiple calls on
particular platfrom (specifically, Windows).

Not to mention that loading FFI (esp. at startup) is costly and we try
to delay it if we can, but with this approach to respond_to? the
method call of respond_to? will cause the loading of FFI. Plus we have
to specialize respond_to? implementations for those classes that could
have "not implemented" methods, this is also not ideal for JVM
performance. And we'd pay some performance penalty for runtime system
capabilities introspection, keeping the lists of "not-implemented"
methods, doing special cases in respond_to?.

So, as you see, this whole thing is just a special case on top of
special case on top of another one. Pretty ugly, and still can't be
100% reliable, and is not really fully documented. While the original
behavior was simple and straightforward: iff there is a method
defined, than respond_to? returns true. This is natural approach in
object-oriented environments, no platform or implementation
capabilities should play a role here.

In those really important cases where is important/critical to
introspect platfrom/API/impl capabilities, there should be a
stand-alone API, and hijacking the base OO instruments for that is not
appropriate, in my opinion.

Thanks,
--Vladimir

=end

#23 - 01/25/2010 02:30 AM - headius (Charles Nutter)

=begin
On Sun, Jan 24, 2010 at 10:11 AM, Tanaka Akira akr@fsij.org wrote:

> 2010/1/23 Charles Oliver Nutter headius@headius.com:
>
>> I'm also curious how this affects Windows behavior, since currently
>> MRI stubs out many methods to return nil on Windows (like all of 'etc'
>> library, for example). Should they raise NotImplementedError? Should
>> they return false for respond_to?.
>
> It may be good chance to change etc library.

We originally had etc functions raise NotImplementedError in JRuby,
but quickly discovered that libraries in the wild expect nil results.
Specifically, RubyGems used etc to lookup home dir information, with
nil being a fallback to other methods of doing so.

So there are definitely libraries depending on it, but it would be
nice to make it consistent.

- Charlie

=end

#24 - 01/26/2010 03:34 AM - jonforums (Jon Forums)

=begin

> Thanks but I'm fine. I currently use the following code:
>
> RUBY_ENGINE = defined?(::RUBY_ENGINE) ? ::RUBY_ENGINE : "ruby"
> def self.fork_supported?
>
> # MRI >= 1.9.2's respond_to? returns false for methods
>
> # that are not implemented.
>
> return Process.respond_to?(:fork) &&
> RUBY_ENGINE != "jruby" &&
> RUBY_ENGINE != "macruby" &&
> Config::CONFIG['target_os'] !~ /mswin|windows/
> end

FYIW, 1.9.1p243 of the RC1 from http://rubyinstaller.org/ does...

C:\Users\Jon\Documents>ruby -e "require 'rbconfig'; puts Config::CONFIG['target_os']"
mingw32

=end

#25 - 01/26/2010 06:38 AM - luislavena (Luis Lavena)

=begin
On Mon, Jan 25, 2010 at 2:57 PM, Hongli Lai hongli@plan99.net wrote:

> On 1/25/10 3:46 PM, Roger Pack wrote:
>
>> I could add the method "OS.fork_supported?" to the OS gem if it would
>> help the original poster (and/or if it could help make unnecessary the
>> modification of respond_to? in 1.9.2)
>>
>> Just thinking out loud.

```
    Thanks but I'm fine. I currently use the following code:

    RUBY_ENGINE = defined?(::RUBY_ENGINE) ? ::RUBY_ENGINE : "ruby"
    def self.fork_supported?
        # MRI >= 1.9.2's respond_to? returns false for methods
        # that are not implemented.
        return Process.respond_to?(:fork) &&
            RUBY_ENGINE != "jruby" &&
            RUBY_ENGINE != "macruby" &&
            Config::CONFIG['target_os'] !~ /mswin|windows/
    end
```

JRuby used to respond to target_os as "windows", but none of the actual Windows implementations respond like that.

VC6 build respond with "mswin32" and GCC build respond with "mingw32"

Please consider these targets can be 64bits too, so /mswin|mingw/ will be appropriate.

Cheers,

Luis Lavena
AREA 17

Perfection in design is achieved not when there is nothing more to add,
but rather when there is nothing more to take away.
Antoine de Saint-Exupéry

=end

#26 - 01/26/2010 07:30 AM - hgs (Hugh Sasse)

=begin
On Tue, 26 Jan 2010, Roger Pack wrote:

```
        Thanks but I'm fine. I currently use the following code:

        RUBY_ENGINE = defined?(::RUBY_ENGINE) ? ::RUBY_ENGINE : "ruby"
        def self.fork_supported?
            # MRI >= 1.9.2's respond_to? returns false for methods
            # that are not implemented.
            return Process.respond_to?(:fork) &&
                RUBY_ENGINE != "jruby" &&
                RUBY_ENGINE != "macruby" &&
                Config::CONFIG['target_os'] !~ /mswin|windows/
        end
```

    The kicker is that there may still be edge cases, like ruby on
    interix, cygwin, or IronRuby run on mono. Testing it out seems the
    only stable way, to me.

    MacRuby doesn't support fork?
    -r

For what it is worth, on cygwin I get:

```
Hugh@Home-PC ~
$ cat fork_supported.rb
require 'rbconfig'

RUBY_ENGINE = defined?(::RUBY_ENGINE) ? ::RUBY_ENGINE : "ruby"
def fork_supported?
# MRI >= 1.9.2's respond_to? returns false for methods
# that are not implemented.
return Process.respond_to?(:fork) &&
RUBY_ENGINE != "jruby" &&
RUBY_ENGINE != "macruby" &&
Config::CONFIG['target_os'] !~ /mswin|windows/
end
```

```
puts fork_supported?.inspect

Hugh@Home-PC ~
$ ruby !$
ruby fork_supported.rb
true
```

This is on the older -- Pre Christmas -- cygwin. Last time I updated a machine
to the new major release it wiped /cygwin/home, so I've erred on the side of
caution for now.

```
        Hugh
```

=end

Files

0001-Add-method-Process.fork_supported-for-checking-wheth.patch	2.11 KB	01/21/2010	hongli (Hongli Lai)
0001-Add-method-Process.fork_supported-for-checking-wheth.patch	1.54 KB	01/21/2010	hongli (Hongli Lai)