Ruby trunk - Feature #12573

Introduce a straightforward way to discover whether a process is running

07/07/2016 10:38 PM - will_in_wi (William Johnston)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Background

The best present solution is to kill the process with a signal of 0, and then evaluate the exit code and two possible exceptions to determine whether or not this process is alive. This is surprising when Ruby has so many beautiful ways of handling related things.

In Ruby, this looks like (from the process_exists gem):

```
def self.exists?(pid)
  Process.kill(0, pid.to_i)
  true
rescue Errno::ESRCH # No such process
  false
rescue Errno::EPERM # The process exists, but you don't have permission to send the signal to it.
  true
end
```

Usecase

I have a background process which needs to be able to determine whether or not another copy of it is running. This is accomplished via a PID file, but I need to be able to confirm whether the given PID actually exists.

See also

Someone has already wrapped up a Ruby version of this into a gem (from which the Ruby implementation comes): https://github.com/wilsonsilva/process_exists

History

#1 - 07/07/2016 10:39 PM - will_in_wi (William Johnston)

- Description updated

Fix formatting.

#2 - 07/07/2016 10:40 PM - will_in_wi (William Johnston)

Pull request: https://github.com/ruby/ruby/pull/1399

#3 - 07/08/2016 12:47 AM - shyouhei (Shyouhei Urabe)

What about Windows? Does this work as-is?

#4 - 07/08/2016 02:23 PM - will_in_wi (William Johnston)

I don't have a Windows machine to set this up and test. I'll try and do so. It was commented on the PR that there are some issues with the test suite in terms of Windows support, so I'll need to take a look at that.

#5 - 07/08/2016 03:08 PM - will_in_wi (William Johnston)

I've updated the PR to fix this round of comments. If there are more comments, I'll address them, otherwise, what else should I do to get this decided upon?

I'm also working on a Windows VM to confirm that this all works there.

Thanks!

#6 - 07/08/2016 06:45 PM - will_in_wi (William Johnston)

I tried to get Ruby to compile on a Windows 2012R2 VM, and I think I was partially successful. It appears that test_process_exists_when_not_exists fails on Windows, but a ton of other tests on trunk also fail, so I don't think I've gotten it to work correctly. If someone has a Windows Ruby setup that they can switch to my branch and run the test suite, I'd appreciate it.

#7 - 07/29/2016 08:57 PM - will_in_wi (William Johnston)

What's the process for a decision being made on this feature request?

https://bugs.ruby-lang.org/projects/ruby/wiki/HowToContribute seems to encourage a ping if a feature request gets missed. In this case, ping!

Thanks!

#8 - 07/30/2016 04:12 AM - duerst (Martin Dürst)

On 2016/07/30 05:57, will@johnstonclan.net wrote:

> What's the process for a decision being made on this feature request?
>
> https://bugs.ruby-lang.org/projects/ruby/wiki/HowToContribute seems to encourage a ping if a feature request gets missed.

I have sligthly updated that page. Please check again.

I wonder whether you have or can make a patch, and whether it would work
on e.g. Windows.

#9 - 07/30/2016 10:28 AM - nobu (Nobuyoshi Nakada)

Martin Dürst wrote:

> I wonder whether you have or can make a patch, and whether it would work
> on e.g. Windows.

It works on Windows too.
I'd added this to the agenda of the last meeting, but we didn't have time enough.

#10 - 08/09/2016 02:45 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

It should be named exist? to be consistent with other methods in the standard libraries.
Besides that, I don't see a proper use-case for this method. Even when you have pid, the process you are trying to watch may have already terminated and the process id may be recycled.

Do you have any concrete use-case of Process#exist? in mind? If you have, please reopen the issue.

Matz.

#11 - 03/06/2017 09:44 PM - will_in_wi (William Johnston)

I'm unable to reopen this ticket, so I'd appreciate it if someone would do so for me. My apologies for not responding sooner, I must not have seen the previous message somehow. I came back to see what the status was after reimplementing this again in another project.

I'm happy to rename the method if desired.

My use case is in implementing pid files. When building long-running processes, I will write out a file containing the pid of the presently running process. If I run this process again, it will see the pidfile and check to see whether the process is presently running. If it is running, the new process will exit with an appropriate message. This is a fairly common pattern for background processes.

You are correct that due to pid recycling, this does not prove that the process which is running is the same as the one which originally created the pid file. Thankfully, in most situations this is fairly rare and a false positive would simply result in a process incorrectly failing to start, which could be manually corrected by removing the pid file. If this is an unacceptable risk for a particular situation, additional means would be taken to verify that the process which is running is as expected.

I've already monkey-patched this into several applications and am expanding its use.

For additional evidence, the bulk of the damonizing gems listed here (https://www.ruby-toolbox.com/categories/daemonizing) reimplement this code:
Daemons: https://github.com/thuehlinger/daemons/blob/master/lib/daemons/pid.rb#L5
Daemon-Kit: https://github.com/kennethkalmer/daemon-kit/blob/master/lib/daemon_kit/pid_file.rb#L15
Dante: https://github.com/nesquena/dante/blob/master/lib/dante/runner.rb#L172
Fallen: https://github.com/inkel/fallen/blob/master/lib/fallen.rb#L115
Daemonizer: https://github.com/glebpom/daemonizer/blob/develop/lib/daemonizer/worker.rb#L67

My pitch for this is that when we see this exact pattern repeatedly reimplemented, it might be worth pulling into Ruby core.

Thanks for considering!

#12 - 03/07/2017 01:49 AM - akr (Akira Tanaka)

How about locking a file?

If processes locks a file exclusively,
Only one process can success.
It can be used to detect already running daemon process.

lockfile is reliable than signal 0 because the file is unlocked when the process exits.

FreeBSD provides utilities to do it:
https://www.freebsd.org/cgi/man.cgi?query=pidfile&apropos=0&sektion=0&manpath=FreeBSD+11.0-RELEASE+and+Ports&arch=default&format=html

#13 - 03/07/2017 02:02 AM - duerst (Martin Dürst)

- Status changed from Rejected to Open

William Johnston wrote:

> I'm unable to reopen this ticket, so I'd appreciate it if someone would do so for me.

Done.

#14 - 03/07/2017 01:57 PM - will_in_wi (William Johnston)

Thanks Martin!

Akira, that's another good way of handling the issue in a lot of circumstances. However, lockfiles have some issues on certain filesystems like NFS. Also, with a locked file, it isn't always straightforward across platforms to identify the application which has locked the file (which is necessary for init scripts and such). A number of implementations of this will combine the two: a locked pidfile. This way, it is fairly easy to tell if the application is still running by using the lockfile, and other applications can still read the file to find out the pid of the application which is running.

#15 - 03/07/2017 01:59 PM - will_in_wi (William Johnston)

A number of applications, such as Monit and God, use the pidfiles of the applications they are monitoring in order to identify if they are still running.

Pid files are referenced in the Linux Standards Base init script spec: http://refspecs.linuxbase.org/LSB_3.0.0/LSB-PDA/LSB-PDA/iniscrptfunc.html
Also, in the Filesystem Hierarchy Standard: http://www.pathname.com/fhs/2.2/fhs-5.13.html

#16 - 04/18/2017 05:24 AM - shyouhei (Shyouhei Urabe)

We looked at this issue in yesterday's developer meeting.

Agreed that a locked pidfile is the best to achieve your request. One question is, do we still need to kill(0) in that case? Because if we assume locking, that should be more reliable than kill.

#17 - 04/18/2017 11:53 AM - will_in_wi (William Johnston)

Thanks for looking at this!

Locking is less reliable than kill(0) in situations such as NFS filesystems. Not all filesystems have a robust locking mechanism. So, yes, we'd need kill(0) for a robust implementation of a locking pidfile. You can see in note 14 a number of examples of established gems which use the kill(0) pattern.

#18 - 04/19/2017 12:39 AM - shyouhei (Shyouhei Urabe)

will_in_wi (William Johnston) wrote:

> Thanks for looking at this!
>
> Locking is less reliable than kill(0) in situations such as NFS filesystems. Not all filesystems have a robust locking mechanism. So, yes, we'd need kill(0) for a robust implementation of a locking pidfile. You can see in note 14 a number of examples of established gems which use the kill(0) pattern.

Do you really want to place a pidfile on a NFS volume? I'd rather mount a ramdisk on /var for diskless situation.

#19 - 04/19/2017 12:42 AM - will_in_wi (William Johnston)

Yup, sometimes you have access to an NFS volume, but not enough system access to mount things.

#20 - 04/19/2017 03:54 AM - shyouhei (Shyouhei Urabe)

will_in_wi (William Johnston) wrote:

> Yup, sometimes you have access to an NFS volume, but not enough system access to mount things.

No, I mean, /var is expected to be suitable for locking. You have already mentioned FHS, and it's clearly stated there that /var/run shall not be shared among different systems.

#21 - 04/19/2017 12:05 PM - will_in_wi (William Johnston)

Ah, I see what you are saying.

/var isn't always available to an application, especially in a shared server situation. That's one of the reasons pretty much every one of the daemonizing gems implements the kill(0) mechanism.