Ruby - Feature #4015

File::DIRECT Constant for O_DIRECT

11/03/2010 06:50 AM - runpaint (Run Paint Run Run)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

=begin
A couple of the open(2) flags on Linux don't have corresponding File:: constants:

- O_DIRECT

  <<Try to minimize cache effects of the I/O to and from this file. In general this will degrade performance, but it is useful in special situations, such as
  when applications do their own caching. File I/O is done directly to/from user space buffers. The O_DIRECT flag on its own makes at an effort to trans-
  fer data synchronously, but does not give the guarantees of the O_SYNC that data and necessary metadata are transferred. To guarantee synchronous I/O the
  O_SYNC must be used in addition to O_DIRECT.>>

This is added with the attached patch. With the patch applied:

run@paint:~$ strace ruby -e 'open("/tmp/foo", File::DIRECT, 0644)' 2>&1|grep O_DIRECT
open("/tmp/foo", O_RDONLY|O_DIRECT)   = 3

- O_CLOEXEC

This has a patch pending in #1291.

- O_DIRECTORY

man 2 open notes "This flag is Linux-specific, and was added in kernel version 2.1.126, to avoid denial-of-service problems if opendir(3) is called on a FIFO or tape device, but should not be used outside of the implementation of opendir(3)." We can probably ignore this one.

- O_LARGEFILE

We already handle large-file support automatically, so this can be ignored, too.
=end

Related issues:

Related to Ruby - Feature #4038: IO#advise	**Closed**	**11/09/2010**

Associated revisions

Revision a3ba982c - 12/19/2010 04:18 PM - kosaki (Motohiro KOSAKI)

- io.c (Init_IO): Added O_DIRECT. This feature was propsed by Run Paint Run Run.
  [Feature #4015] [ruby-core:33018]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@30247 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 11/03/2010 07:42 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:33018] [Ruby 1.9-Feature#4015][Open] File::DIRECT Constant for O_DIRECT"
on Wed, 3 Nov 2010 06:51:04 +0900, Run Paint Run Run redmine@ruby-lang.org writes:

|A couple of the open(2) flags on Linux don't have corresponding File:: constants:
|
|* O_DIRECT
|* O_DIRECTORY

We haven't made up a policy for platform specific constants. Opinions
are welcome.

|* O_CLOEXEC

Is this really needed?

|* O_LARGEFILE
|
|We already handle large-file support automatically, so this can be ignored, too.

Agreed.

```
matz.
```

=end

#2 - 11/03/2010 07:56 AM - runpaint (Run Paint Run Run)

=begin

> A couple of the open(2) flags on Linux don't have corresponding File:: constants:
>
> - O_DIRECT
> - O_DIRECTORY
>
> We haven't made up a policy for platform specific constants. Opinions
> are welcome.

I find this surprising given that platform-specific constants are defined elsewhere. Consider the Linux-specific Process::RLIMIT_MSGQUEUE, Process::RLIMIT_SIGPENDING, and Process::RLIMIT_NICE, for just a few examples. Defining them for all platforms would be fine, also, especially because they often enhance the operation rather than changing its semantics.

> - O_CLOEXEC
>
> Is this really needed?

IMO, yes. As explained in #1291, it was introduced to avoid a race condition.
=end

#3 - 11/03/2010 08:19 AM - runpaint (Run Paint Run Run)

=begin
Two other approaches to platform-specific constants:

- Prefix the constant name with the platform name, e.g. File::LINUX_DIRECT, so as to clearly indicate that the code is not portable.
- Define them as private constants (assuming that feature is approved) so that #const_get needs to be used in order to access them. Again, this would indicate that special care was needed in using the constant.
  =end

#4 - 11/03/2010 05:13 PM - now (Nikolai Weibull)

=begin
On Wed, Nov 3, 2010 at 00:19, Run Paint Run Run redmine@ruby-lang.org wrote:

> - Prefix the constant name with the platform name, e.g. File::LINUX_DIRECT, so as to clearly indicate that the code is not portable.

What happens when, for example, FreeBSD implements it?

=end

#5 - 11/03/2010 07:06 PM - runpaint (Run Paint Run Run)

=begin
On Wed, Nov 3, 2010 at 8:13 AM, Nikolai Weibull now@bitwi.se wrote:

> On Wed, Nov 3, 2010 at 00:19, Run Paint Run Run redmine@ruby-lang.org wrote:
>
> - Prefix the constant name with the platform name, e.g. File::LINUX_DIRECT, so as to clearly indicate that the code is not portable.
>
> What happens when, for example, FreeBSD implements it?

According to http://goo.gl/Y6wF9 it already does, actually; OpenBSD (http://goo.gl/5pXnR), Mac (http://goo.gl/Ru57H), and Solaris (http://goo.gl/6QNsl) don't.

=end

#6 - 11/04/2010 02:09 AM - runpaint (Run Paint Run Run)

=begin
For O_DIRECT, would an approach along the following lines work?

1. Let open's options Hash recognise a :direct key, whose value defaults to false.
2. If O_DIRECT is defined and :direct is true, OR the existing open flags with O_DIRECT.
3. If the platform is Windows and :direct is true, OR the existing flags for CreateFile() with FILE_FLAG_NO_BUFFERING and FILE_FLAG_WRITE_THROUGH. http://goo.gl/kaHTs
4. Open the file as normal.
5. If the platform is MacOS and :direct is true, use fcntl() to set F_NOCACHE to 1 for the new file descriptor. http://goo.gl/cl9py

O_CLOEXEC is specified by POSIX.1-2008 http://goo.gl/Y7tS6, so isn't exactly platform-specific. We could let open's option Hash accept a :close_on_exec key which defaults to false. If O_CLOEXEC is defined, we OR it with the open flags, otherwise, after having opened the file, we invoke #close_on_exec. The semantics, therefore, are to provide atomicity on platforms that support it, or otherwise do the best we can. This is not portable, of course, but then neither is calling #close_on_exec= true directly after open, so we haven't lost anything. (The portability of O_CLOEXEC under mingw and other platforms is discussed in http://goo.gl/eliAf)
=end

#7 - 11/06/2010 03:59 AM - runpaint (Run Paint Run Run)

- File io-advise.patch added

=begin
Last suggestion from me, I promise. ;-)

O_DIRECT is a controversial (http://lkml.org/lkml/2007/1/11/121) flag, and Linus and others recommend (http://lkml.org/lkml/2007/1/10/233) using madvise(2)/posix_fadvise(2) instead. The latter is, as the name suggests, part of the POSIX standard, avoiding the current problem of defining platform-specific, non-standard flags. Its advantages are summarised in O'Reilly's *Linux System Programming*:

<<A handful of common application workloads can readily benefit from a little well-intentioned advice to the kernel. Such advice can go a long way toward mitigating the burden of I/O. With hard disks being so slow, and modern processors being so fast, every little bit helps, and good advice can go a long way.

Before reading a chunk of a file, a process can provide the POSIX_FADV_WILLNEED hint to instruct the kernel to read the file into the page cache. The I/O will occur asynchronously, in the background. When the application ultimately accesses the file, the operation can complete without generating blocking I/O.

Conversely, after reading or writing a lot of data—say, while continuously streaming video to disk—a process can provide the POSIX_FADV_DONTNEED hint to instruct the kernel to evict the given chunk of the file from the page cache. A large streaming operation can continually fill the page cache. If the application never intends to access the data again, this means the page cache will be filled with superfluous data, at the expense of potentially more useful data. Thus, it makes sense for a streaming video application to periodically request that streamed data be evicted from the cache.

A process that intends to read in an entire file can provide the POSIX_FADV_SEQUENTIAL hint, instructing the kernel to perform aggressive readahead. Conversely, a process that knows it is going to access a file randomly, seeking to and fro, can provide the POSIX_FADV_RANDOM hint, instructing the kernel that readahead will be nothing but worthless overhead.>>

Implementing posix_fadvise(2) would avoid complicating the logic of open any more, and at the same time provide a more general solution than O_DIRECT. The attached patch defines IO#advise as a wrapper around posix_fadvise(2). As this advice is never binding, and #respond_to? returns false for :advise on platforms that don't support it, it is trivial to write portable code that only invokes #advise where supported. Granted, this solution

still requires defining constants. However, there is no danger of defining them on all platforms because platforms that don't support this syscall will raise a NotImplementedError for #advise.
=end

#8 - 11/07/2010 11:23 PM - normalperson (Eric Wong)

=begin
Run Paint Run Run redmine@ruby-lang.org wrote:

> Last suggestion from me, I promise. ;-)

It's a great one and I second it. Promoting fadvise use would help to push kernel hackers to implement/improve support for it.

> around posix_fadvise(2). As this advice is never binding, and #respond_to? returns false for :advise on platforms that don't support it, it is trivial to write portable code that only invokes #advise where supported. Granted, this solution still requires defining constants. However, there is no danger of defining them on all platforms because platforms that don't support this syscall will raise a NotImplementedError for #advise.

I suggest making #advise just noop on platforms where it's not currently supported to avoid #respond_to? checks. Since "advice" is exactly that, the underlying implementation in the kernel never guarantees any effect (and neither can Ruby). For example, current versions of Linux just ignore POSIX_FADV_NOREUSE, but that doesn't stop me from putting it in my code anyways because it could one day be run on a machine where it is implemented.

Since Ruby isn't strictly tied to POSIX, it could eventually take some liberties and affect/influence the userspace buffering behavior, too.

- *advice* is one of the following constants:
  - File::POSIX_FADV_NORMAL - No advice to give; the default assumption for
- an open file.
  - File::POSIX_FADV_SEQUENTIAL - The data will be accessed sequentially:
- with lower offsets read before higher ones.
  - File::POSIX_FADV_RANDOM - The data will be accessed in random order.
  - File::POSIX_FADV_WILLNEED - The data will be accessed in the near future.
  - FILE::POSIX_FADV_DONTNEED - The data will not be accessed in the near
- future.
  - File::POSIX_FADV_NOREUSE - The data will only be accessed once.

One thing I like about the "fadvise" gem is that it takes symbolic arguments (much like the 1.9.2 socket API extensions) so I can use it like this:

```
io.fadvise(0, io.stat.size, :dont_need)
```

Though I'd probably use ":dontneed" or ":DONTNEED" instead.

--
Eric Wong

=end

#9 - 11/08/2010 09:48 PM - runpaint (Run Paint Run Run)

- File io-advise.patch added

=begin
Thanks, Eric. I've updated the patch with your suggestions. The *advice* argument is now a Symbol. On platforms that don't support posix_fadvise(2), the arguments are still sanity checked--hence the ugly ifdefs--but then we just return nil.
=end

#10 - 11/08/2010 11:43 PM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:33107] [Ruby 1.9-Feature#4015] File::DIRECT Constant for O_DIRECT"
on Mon, 8 Nov 2010 21:49:05 +0900, Run Paint Run Run redmine@ruby-lang.org writes:

|Issue #4015 has been updated by Run Paint Run Run.
|
|File io-advise.patch added
|
|Thanks, Eric. I've updated the patch with your suggestions. The *advice* argument is now a Symbol. On platforms that don't support posix_fadvise(2), the arguments are still sanity checked--hence the ugly ifdefs--but then we just return nil.

I am not against IO#advice, but you should at least open independent
ticket for it. May I consider File::DIRECT request is withdrawn?

```
      matz.
```

=end

#11 - 11/09/2010 10:29 AM - runpaint (Run Paint Run Run)

=begin
On Mon, Nov 8, 2010 at 2:42 PM, Yukihiro Matsumoto matz@ruby-lang.org wrote:

> Hi,
>
> In message "Re: [ruby-core:33107] [Ruby 1.9-Feature#4015] File::DIRECT Constant for O_DIRECT"
> on Mon, 8 Nov 2010 21:49:05 +0900, Run Paint Run Run redmine@ruby-lang.org writes:
>
> |Issue #4015 has been updated by Run Paint Run Run.
> |
> |File io-advise.patch added
> |
> |Thanks, Eric. I've updated the patch with your suggestions. The *advice* argument is now a Symbol. On platforms that don't support posix_fadvise(2), the arguments are still sanity checked--hence the ugly ifdefs--but then we just return nil.
>
> I am not against IO#advice, but you should at least open independent
> ticket for it. May I consider File::DIRECT request is withdrawn?

It feels like I'm gambling between which proposal has more chance of
being accepted... I've opened a new ticket for IO#advise (#4038), so I
guess this one can be closed.

=end

#12 - 11/09/2010 10:48 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:33111] Re: [Ruby 1.9-Feature#4015] File::DIRECT Constant for O_DIRECT"
on Tue, 9 Nov 2010 10:29:32 +0900, Run Paint Run Run runrun@runpaint.org writes:

|> I am not against IO#advice, but you should at least open independent
|> ticket for it. May I consider File::DIRECT request is withdrawn?
|
|It feels like I'm gambling between which proposal has more chance of
|being accepted... I've opened a new ticket for IO#advise (#4038), so I
|guess this one can be closed.

And other platform (linux?) specific constants, open other tickets, if
you need them.

```
      matz.
```

=end

#13 - 12/16/2010 02:47 AM - kosaki (Motohiro KOSAKI)

=begin
Hi

May I ask current status of this proposal? (Why no assignment) I think io-advise.patch works enough and we can't make O_DIRECT emulation and good fallback logic. Also, O_DIRECT is de-fact standard and a lot of platform support it. So, *if* nobody complain, I'd like to commit it.

Thanks.
=end

#14 - 12/16/2010 03:18 AM - runpaint (Run Paint Run Run)

=begin

> May I ask current status of this proposal? (Why no assignment)

I was wondering the same thing.

> I think io-advise.patch works enough and we can't make O_DIRECT emulation and good fallback logic. Also, O_DIRECT is de-fact standard and a lot of platform support it. So, *if* nobody complain, I'd like to it.

Do you want to commit IO#advise or File::DIRECT? Either's fine with
me, obviously, but I just wanted to check.

=end

#15 - 12/16/2010 03:32 AM - kosaki (Motohiro KOSAKI)

=begin
2010/12/16 Run Paint Run Run runrun@runpaint.org:

> > May I ask current status of this proposal? (Why no assignment)
>
> I was wondering the same thing.
>
> > I think io-advise.patch works enough and we can't make O_DIRECT emulation and good fallback logic. Also, O_DIRECT is de-fact standard and a lot of platform support it. So, *if* nobody complain, I'd like to it.
>
> Do you want to commit IO#advise or File::DIRECT? Either's fine with
> me, obviously, but I just wanted to check.

File::DIRECT. But Now I'm also reviewing IO#advise too. (sorry for the delay.)

=end

#16 - 12/16/2010 07:43 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

If you consider O_DIRECT is common across platforms, I'd agree with
merging it.

```
      matz.
```

In message "Re: [ruby-core:33729] [Ruby 1.9-Feature#4015] File::DIRECT Constant for O_DIRECT"
on Thu, 16 Dec 2010 02:47:50 +0900, Motohiro KOSAKI redmine@ruby-lang.org writes:

|Hi
|
|May I ask current status of this proposal? (Why no assignment) I think io-advise.patch works
|enough and we can't make O_DIRECT emulation and good fallback logic. Also, O_DIRECT is
|de-fact standard and a lot of platform support it. So, *if* nobody complain, I'd like to
|commit it.
|

Thanks.
http://redmine.ruby-lang.org/issues/show/4015

=end

#17 - 12/16/2010 10:04 PM - kosaki (Motohiro KOSAKI)

=begin
2010/12/16 Yukihiro Matsumoto matz@ruby-lang.org:

> Hi,
>
> If you consider O_DIRECT is common across platforms, I'd agree with
> merging it.

Yeah, at least following platforms support it.

o Linux
o FreeBSD
o NetBSD
o AIX

Unfortunatelly, Solaris and MacOS have similar but different function.
They support to turn direct-io on/off after open. However I haven't
seen such dynamically turning-on requirement.

o Solaris -> directio()
o Mac -> fcntl(F_NOCACHE)

So, we have two option, I think.

o Implement O_DIRECT emulation.
Easy. Only problem is how do we define O_DIRECT number. (iow
How do we know unused open flag bit on their platform)
o Enhance ruby open method.
Adding options hash argument at last of open. :directio=>true turn
directio feature on. It seems slightly overengineering. but options
hash can be used other ways. (Namely platform dependent open
option support e.g. windows have some platform specific open
flag for performance)

Which do you prefer?

Thanks.

=end

#18 - 12/20/2010 01:27 AM - kosaki (Motohiro KOSAKI)

- Status changed from Open to Closed

- % Done changed from 0 to 100

=begin
This issue was solved with changeset r30247.
Run Paint, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

=end

Files

io.c-o_direct.patch	468 Bytes	11/03/2010	runpaint (Run Paint Run Run)

io-advise.patch	4.92 KB	11/06/2010	runpaint (Run Paint Run Run)
io-advise.patch	5.23 KB	11/08/2010	runpaint (Run Paint Run Run)