The Old New Thing

We're currently using FILE_FLAG_NO_BUFFERING and FILE_FLAG_WRITE_THROUGH, but we would like our WriteFile to go even faster

6 Mar 2014 7:00 AM | 29

A customer said that their program's I/O pattern is to open a file and then every so often write about 100KB of data into the file. They are currently using the `FILE_FLAG_NO_BUFFERING` and `FILE_FLAG_WRITE_THROUGH` flags to open a file, and they wanted to know what else they could do to make their writes go even faster.

Um, for one thing, you stop passing those two flags!

Those two flags in combination basically mean "Give me the slowest possible I/O performance!" because they force all I/O to go through to the physical media right away.

Removing the `FILE_FLAG_WRITE_THROUGH` flag will be a big help. This allows the hardware disk cache to do its normal job of completing the I/O immediately and performing the physical I/O lazily (perhaps in an optimized order based on subsequent writes). A 100KB write is a small enough write that your I/O time on rotational media will be dominated by the seek time. It'll take five to ten milliseconds to move the head into position and only one millisecond to write out the data. You're wasting 80% or more of your time just preparing for the write.

Much better would be to issue the I/O without the `FILE_FLAG_WRITE_THROUGH` flag so that the entire 100KB I/O request goes into the hard drive on-board cache. (It will fit quite easily, since the on-board cache for today's hard drives will be 8 megabytes or larger.) Your `WriteFile` will complete immediately, and the commit to physical storage will occur while your program is busy doing computation.

If the writes truly are sporadic (as the customer claims), the I/O buffer will be flushed out by the time the next round of application I/O begins.

Removing the `FILE_FLAG_NO_BUFFERING` flag will also help, because that allows the operating system disk cache to get involved. If the application reads back from the file, the read can be satisfied from the disk cache, avoiding the physical I/O entirely.

As a side note, the `FILE_FLAG_WRITE_THROUGH` flag is largely ineffective nowadays, because SATA drivers ignore the flush request. The file system doesn't know that the driver is lying to it, so it will still do all the work on the assumption that the write-through request worked, even though we know that the extra work is ultimately pointless.

For example, NTFS will issue metadata writes with a flush to ensure that the data on the physical media is consistent. But if the driver is ignoring flush requests, all this extra work accomplishes nothing aside from wasting I/O bandwidth. Even worse, NTFS *thinks* that the data on the drive is physically consistent, *but it isn't*. The result is that a poorly-timed power outage (or device removal) can result in metadata corruption that takes a `chkdsk` to repair.

Now, it may be that the customer's program is using the `FILE_FLAG_NO_BUFFERING` and `FILE_FLAG_WRITE_THROUGH` flags for a specific purpose unrelated to performance, so you can't just go walking in and ripping them out without understanding why they were there. But if they added the flags thinking that it would make the program run faster, then they were operating under a false assumption.

Blog - Comment List MSDN TechNet

Comments

alegr1
6 Mar 2014 7:13 AM
#

In command packets for SATA Native Command Queing, there is FUA (Force Unit Access) bit. Some drives may ignore that bit, though.

Joshua
6 Mar 2014 7:14 AM
#

I wonder if it's possible to sue the HD manufacturer for advertising SATA when it's not really SATA because the flush command returns success when it doesn't work.

You do not use FILE_FLAG_WRITE_THROUGH because it's faster. You use it because it's correct.

Yuri Khan
6 Mar 2014 7:19 AM
#

There is a theory that if your application writes a large file and never reads back from it, then passing FILE_FLAG_NO_BUFFERING will help other applications and files retain their caches, which is ultimately beneficial to overall performance.

alegr1
6 Mar 2014 7:19 AM
#

>I wonder if it's possible to sue the HD manufacturer for advertising SATA when it's not really SATA because the flush command returns success when it doesn't work.

Remember when MS had to increase shutdown delay in Windows 95 because of IBM ATA drives not completing flush?

alegr1
6 Mar 2014 7:21 AM
#

>There is a theory that if your application writes a large file and never reads back from

it, then passing FILE_FLAG_NO_BUFFERING will help other applications and files retain their caches, which is ultimately beneficial to overall performance.

Large files opened in caching mode is a bane of Windows users. Even not just large files, but lots of files. For example, when an AV scan or system backup runs in background.

acq
6 Mar 2014 7:54 AM
#

I am aware that a lot of programming forums/blogs of suspicious quality are spreading myths for years that FILE_FLAG_NO_BUFFERING and FILE_FLAG_WRITE_THROUGH are the "magic" that makes your writes and reads faster. Based only on false beliefs and the lack of understanding and tests.

Raymond is of course right: unless you're making video conversion software (or antivirus scanner) that should push a lot of gigabytes through the CPU just because, you probably don't want to use FILE_FLAG_NO_BUFFERING. And using FILE_FLAG_WRITE_THROUGH has even less sense unless you write your own file system.

Olivier
6 Mar 2014 8:23 AM
#

> And using FILE_FLAG_WRITE_THROUGH has even less sense unless you write your own file system.

Or your own database system, where you want to make sure data is safely committed to disk.

Mark VY
6 Mar 2014 8:29 AM
#

Why ignore the request? Isn't that a recipe for disaster?

Brian_EE
6 Mar 2014 8:59 AM
#

I thought some of this sounded familiar. Raymond has covered this in one form or another several times (the HD ignorning flush requests).

blogs.msdn.com/.../10059575.aspx

benjamin

6 Mar 2014 9:59 AM
#

In the early days of CD Burning that NO_BUFFERING flag was a gift from heaven since burning a CD often meant getting control back and witnessing everything slowly. swap. in. as the disc image was evicted from memory.

Gerard
6 Mar 2014 10:13 AM
#

Legacy IDE drives provide no support for the equivalent of SCSI FUA in drives based on any version of the Advanced Technology Attachment (ATA) specification prior to version 7. This issue was addressed in ATA-7, on which most Serial ATA (SATA) drives are based. Of course that doesn't mean it is honored or correctly implemented in all cases..

arghhhhhhhhhhh
6 Mar 2014 11:24 AM
#

"The file system doesn't know that the driver is lying to it"

All SATA and AHCI drivers I've ever used has been provided and signed by Microsoft, so who is lying?

[*The hardware is lying to the driver. -Raymond*]

alegr1
6 Mar 2014 11:26 AM
#

[In the early days of CD Burning that NO_BUFFERING flag was a gift from heaven since burning a CD often meant getting control back and witnessing everything slowly. swap. in. as the disc image was evicted from memory.]

Ughhh. A very popular program I used then didn't do NO_BUFFERING. Good times. Well, at least you don't have to run PATA in PIO mode (or, worse, 8 bit SCSI in PIO mode) anymore.

Torkell
6 Mar 2014 12:07 PM
#

The issue of the hardware lying to the software gets especially fun when you stick a battery-backed RAID controller in the middle. Now, the controller being battery-backed can lie with impunity because it won't lose data across a power outage. But if the disks continue to lie to the RAID card (or worse, the RAID card doesn't even try to turn the disk's write cache off), then that cache becomes worthless and tends to result in a scrambled RAID array. I recall LiveJournal getting hit by exactly this issue many years ago. Their datacenter did have UPS, but that's not much use when someone hits the "switch everything off now damnit" button thinking it's the "unlock door" button...

Nico
6 Mar 2014 12:08 PM
#

> Large files opened in caching mode is a bane of Windows users.

This is a frustrating reality. For example I love using Robocopy for backup purposes, but it doesn't have an option to bypass the filesystem cache. This means that my entire hard drive filled with movies, games, disk images, etc all are getting written to memory. It's also annoying because Robocopy shows the progress of the copy and then just sits and hangs at 100% while it waits for the file to actually get written someplace. I much prefer the behavior of tools such as TeraCopy which have an option to bypass the cache. The copy speed is much more realistic and when a file is complete it really is complete. It also doesn't cause me thrashing/paging grief.

What's even worse is that this is a known issue in "write only" usage scenarios and various Microsoft utilities (originally ESEUTIL and later XCOPY) have options to bypass the cache. I just looked online to see if Robocopy got some love in Windows 8, but a first glance seems to indicate that it's actually pretty broken in Win8. Fantastic.

alegr1
6 Mar 2014 4:21 PM
#

@Nico:

One issue with writing files in NO_BUFFERING mode is that it doesn't allow to set the file length other than multiple of block size. SetFilePointer will explicitly fail if the offset is not a multiple of block size, even though no I/O happens at that time, and the actual offset validation also happens inside ReadFile/WriteFile. So if you want to write a file with non-round size, you have to reopen it in buffered mode, and set its length. This is totally stupid.

Richard Laager
6 Mar 2014 7:23 PM
#

> The hardware is lying to the driver. -Raymond

You have a typo in the blog post. It reads "SATA drivers" when it should be "SATA drives".

Lawrence
6 Mar 2014 11:20 PM

#

@Richard: Raymond's post is fine, the driver is lying too. It's telling the OS/Application "no worries; you tell me you want to flush the cache, I'll make sure it happens". The driver knows it can't make that guarantee with the drive lying to it.

TheCodeArtist
7 Mar 2014 1:18 AM
#

The same thing happens on Linux with how differently the filesystem and block device layers implement O_SYNC. This results in improving I/O throughput on block-dev when O_DIRECT is used. http://goo.gl/YNUvUp

Neil
7 Mar 2014 2:44 AM
#

I'm unclear as to what "write" means in this context.

If you preallocate the file, you could use overlapped I/O, although I don't know whether this makes the call to WriteFile faster or not*. Alternatively you could map the file into memory, eliminating the call to WriteFile altogether, thus reducing the time taking to call WriteFile by 100%!

If they're referring to the physical write of the blocks to disk, then bypassing the cache would presumably reduce the time it takes to get the bits on the disk. In the pathological worst case, the OS doesn't have any standby memory available and has to swap something anyway.

*Separately for each combination of the flags discussed.

John Doe
7 Mar 2014 3:42 AM
#

@Lawrence, the driver is telling what it thinks is the truth, which is what the drive says. If the driver, or the driver's developer, could tell the difference when the drive is lying, then you could say it's lying as well. Timing isn't safe: a fast response might be suspicious from a plate/disk drive, but not from a flash/SSD drive. And having an enumeration of all drive IDs doesn't work either: manufacturers often keep the exact same ID between what they regard as minor revisions or fixed glitch that no one outside is (yet, with luck, ever) aware of.

The liar is the drive, the one getting hurt is the file system module, the driver is just the messenger. Don't shoot the messenger.

Kirchner

7 Mar 2014 3:47 AM
#

robocopy /J ?

From usage:

/J :: copy using unbuffered I/O (recommended for large files).

Fleet Command
7 Mar 2014 9:00 AM
#

Ah, great. Another instance of storage hardware manufacturers atrocious behavior: Not honoring specs, using a different size for kilobyte/megabyte/gigabyte/etc. and lying to the OS. I wish there was a happy ending to all this.

smf
7 Mar 2014 10:12 AM
#

I think I've used at least one of those flags before when I needed to know that the write had completed. Windows mobile writing to an sd card, writes go awol if the unit suspends before the write hit the sd card because the os briefly forgets the card was there when it wakes up. Most of the I/O was on a background thread, so fast wasn't a huge deal. Not helped by compact framework not being able to give you any idea that the previous write failed.

I'd probably suggest they leave well alone. You would be better off telling them you can make their car go faster by removing the brakes.

Nico
7 Mar 2014 12:03 PM
#

@Kirchner: Ah, that's good to know. Thanks for sharing.

Ken Hagan
8 Mar 2014 2:37 AM
#

Your "lying" drive firmware may of course be provided by your VMM host, which may have UI options to let the end-user decide how much dishonesty is required.

Windows *may* be running on the bare metal and talking to honest hardware/firmware, or it may not. These days, it quite frequently isn't.

640k
10 Mar 2014 2:12 AM
#

How does "Enable write caching on the disk" and "enable advanced performance" integrate with FILE_FLAG_NO_BUFFERING and FILE_FLAG_WRITE_THROUGH?

www.windowsreference.com/.../disk-write-cache2.png

(using an OS which supports the options of course)

Can we have a table to explain all combinations?

acq
10 Mar 2014 4:28 AM
#

For us who still depend on Windows 7, robocopy got the /J switch starting with Windows 8, but good old xcopy has it even in Windows 7:

```
C:\>xcopy /?

...

 /J           Copies using unbuffered I/O. Recommended for very large files.
```

Michael Grier [MSFT]
10 Mar 2014 3:45 PM
#

Copying files without /J will evict possibly very interesting pages from the cache (aka the standby list).

Copying small files with /J is slower than normal copies.

This is am ongoing hard problem. CPUs have the same cache dilution/poisoning issue with memcpy() implementations leading to cached and uncached memory move instructions but that doesn't necessarily help the middleware coder. You probably know at the top of the stack if you're copying 1kb or 1tb but it's hard to have things in the middle guess the superior policy. (Copies are particularly bad since you can end up with both the source and destination in the cache if you're not careful. You *may* use one of them soon after so it *may* be useful but it's extremely unlikely you need both.)