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

devblogs.microsoft.com/oldnewthing/20140306-00

March 6, 2014

Raymond Chen

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.

Raymond Chen

Follow