Some questions about unflushed data and calling FlushFileBuffers on a new handle to a file

devblogs.microsoft.com/oldnewthing/20170524-00

May 24, 2017

Raymond Chen

Consider the following sequence of events:

1. Process A opens a file with `CreateFile` with attributes that include *neither* `FILE_FLAG_NO_BUFFERING` nor `FILE_FLAG_WRITE_THROUGH`.
2. Process A writes to the file with `WriteFile`. These writes are internally buffered since we didn't disable buffering.
3. Process A crashes without calling `CloseHandle`, and before the operating system's internal buffers are flushed to disk.

First question: Under these conditions, will the data written in step 2 be lazy-written to disk? Or is it at risk of being lost forever because the handle wasn't closed?

Let's look at the last part first. Whether the process closed the handle before crashing doesn't affect the story, because the kernel will close all the handles as part of process cleanup. The handle does get closed eventually. Whether the handle closure was done explicitly by the app or implicitly by the kernel doesn't affect the answer.

Okay, now let's look at the first part: Yes, the data written in step 2 will eventually be lazy-written to disk, assuming your system doesn't crash before then.

And that's the middle part of the question: The data is at risk of being lost forever, but not because the handle wasn't closed. It's at risk of being lost forever because the system might crash before it gets flushed out.

Okay, let's extend the scenario:

4. Before the operating system flushes its internal buffers naturally, Process B opens the same file, with the same attributes.
5. Process B calls `FlushFileBuffers`.

Does this call to `FlushFileBuffers` cause the data written by Process A to be flushed to disk?

Yes. A call to `FlushFileBuffers` will flush data for that file, even if the data was written by a different handle.

If `FlushFileBuffers` is never called, then the operating system will flush the buffer at its convenience.

Note: In step 2, the relevant call is `WriteFile`. If you write the data to the file using a runtime-provided function like `fwrite`, then that data might be sitting in the runtime's buffer without ever triggering a `WriteFile`. Only when the data is written with `WriteFile` does the data actually reach the operating system's buffers.

Raymond Chen

Follow