When would CopyFile succeed but produce a file filled with zeroes?

devblogs.microsoft.com/oldnewthing/20190517-00

May 17, 2019

Raymond Chen

A customer reported that they very rarely encounter cases where they used the `CopyFile` function to copy a file from a network location to the local system, the function reports success, but when they go to look at the file, it's filled with zeroes instead of actual file data. They were wondering what could cause this situation. They suspected that the computer may have rebooted and wanted to know whether file contents are flushed to disk under those conditions.

When the `CopyFile` function returns success, it does not mean that the data has reached physical media. It means that the data was written to the cache. The customer didn't specify whether the reboot was a crash or an organized restart. If the system crashed, then any unwritten data in the cache is lost. On the other hand, if the reboots through the normal shutdown-and-reboot process, then the cache will be written out as part of the shutdown.

The customer wondered whether passing the `COPY_ FILE_ NO_ BUFFERING` would cover the crash scenario.

A member of the file system team explained that the `COPY_ FILE_ NO_ BUFFERING` flag tells the system not to keep data in memory, but rather send it straight to the device. This flag was recommended for large files (which don't fit in RAM anyway), but it's a bad idea for small files, since every file will have to wait for the device to respond before the system can move on to the next file. You'll have to experiment to find the breakeven point for your specific data set and device.

Note, however, that the `COPY_ FILE_ NO_ BUFFERING` doesn't solve the problem.

For example, the system might crash while the file copy is still in progress. The `CopyFile` function creates a 4GB file (say), but manages to copy only 1GB of data into it before crashing. The other 3GB was never copied even though the file claims to be 4GB in size.

Another possibility is that all the file data makes it to the device, but the metadata does not get written to the device before the crash. The `CopyFile` returned success, but all of the bookkeeping didn't make it to the device.

Even if you call `FlushFileBuffers`, the system could crash before `FlushFileBuffers` returns.

One possible way to address these problems is to copy the file to a temporary name, flush the file buffers, and then rename the file to its final name. The downside of this is that it forces synchronous writes to the device, which slows down your overall workflow, so it's not a cheap algorithm to use.

But let's step back. Is there a way to avoid slowing down the common case just because there's a rare problem case?[1]

A higher-level solution may be in order: The next time your program runs, you can detect that it did not shut down properly. When that happens, hash the file contents and compare it to the expected value. This moves the expensive operation to the rare case, allowing the file copy to proceed at normal speed.

[1] We're assuming that system crashes are rare. If they're not rare, then I think you have bigger problems.

Raymond Chen

Follow