Does the CopyFile function verify that the data reached its final destination successfully?

devblogs.microsoft.com/oldnewthing/20120919-00

September 19, 2012

Raymond Chen

A customer had a question about data integrity via file copying.

> I am using the `File.Copy` to copy files from one server to another. If the call succeeds, am I guaranteed that the data was copied successfully? Does the `File.Copy` method internally perform a file checksum or something like that to ensure that the data was written correctly?

The `File.Copy` method uses the Win32 `CopyFile` function internally, so let's look at `CopyFile`.

`CopyFile` just issues `ReadFile` calls from the source file and `WriteFile` calls to the destination file. (Note: Simplification for purposes of discussion.) It's not clear what you are hoping to checksum. If you want `CopyFile` to checksum the bytes when the return from `ReadFile`, and checksum the bytes as they are passed to `WriteFile`, and then compare them at the end of the operation, then that tells you nothing, since they are the same bytes in the same memory.

```
while (...) {
 ReadFile(sourceFile, buffer, bufferSize);
 readChecksum.checksum(buffer, bufferSize);
 writeChecksum.checksum(buffer, bufferSize);
 WriteFile(destinationFile, buffer, buffer,Size);
}
```

The `readChecksum` and `writeChecksum` are identical because they operate on the same bytes. (In fact, the compiler might even optimize the code by merging the calculations together.) The only way something could go awry is if you have flaky memory chips that change memory values spontaneously.

Maybe the question was whether `CopyFile` goes back and reads the file it just wrote out to calculate the checksum. But that's not possible in general, because you might not have read access on the destination file. I guess you could have it do a checksum if the destination were readable, and skip it if not, but then that results in a bunch of weird behavior:

- It generates spurious security audits when it tries to read from the destination and gets `ERROR_ACCESS_DENIED`.
- It means that `CopyFile` sometimes does a checksum and sometimes doesn't, which removes the value of any checksum work since you're never sure if it actually happened.
- It doubles the network traffic for a file copy operation, leading to weird workarounds from network administrators like "Deny read access on files in order to speed up file copies."

Even if you get past those issues, you have an even bigger problem: How do you know that reading the file back will really tell you whether the file was physically copied successfully? If you just read the data back, it may end up being read out of the disk cache, in which case you're not actually verifying physical media. You're just comparing cached data to cached data.

But if you open the file with caching disabled, this has the side effect of purging the cache for that file, which means that the system has thrown away a bunch of data that could have been useful. (For example, if another process starts reading the file at the same time.) And, of course, you're forcing access to the physical media, which is slowing down I/O for everybody else.

But wait, there's also the problem of caching controllers. Even when you tell the hard drive, "Now read this data from the physical media," it may decide to return the data from an onboard cache instead. You would have to issue a "No really, flush the data and read it back" command to the controller to ensure that it's really reading from physical media.

And even if you verify that, there's no guarantee that the moment you declare "The file was copied successfully!" the drive platter won't spontaneously develop a bad sector and corrupt the data you just declared victory over.

This is one of those "How far do you really want to go?" type of questions. You can re-read and re-validate as much as you want at copy time, and you *still* won't know that the file data is valid when you finally get around to using it.

Sometimes, you're better off just trusting the system to have done what it says it did.

If you really want to do some sort of copy verification, you'd be better off saving the checksum somewhere and having the ultimate consumer of the data validate the checksum and raise an integrity error if it discovers corruption.

Raymond Chen

Follow