All I/O on a synchronous file handle is serialized; that's why it's called a synchronous file handle

devblogs.microsoft.com/oldnewthing/20130822-00

August 22, 2013

Raymond Chen

File handles are synchronous by default. If you want asynchronous file handles, you need to pass the `FILE_FLAG_OVERLAPPED` flag when you create the handle. And all operations on a synchronous file handle are serialized. You'd think this was a simple and obvious rule, but "Someone" finds it "very surprising that operations can block which only handle file metadata." Imagine if synchronous file handles were not serialized for metadata operations. First of all, it means that the documentation for synchronous file handles suddenly got a lot more complicated. "Some operations on synchronous file handles are serialized, but not others. Specifically, operations on file contents are synchronized, but operations on file metadata are not synchronized." Now things get weird. For example, the size of a file is metadata. Allowing file size operations to bypass serialization means that if you issue a `WriteFile` operation, and then on another thread call `GetFileSize`, you can get a size that is neither the old size nor the new size. Maybe that doesn't bother you. Okay, how about the `DeviceIoControl` function? Does that operate on file contents or file metadata? Your initial reaction might be that `DeviceIoControl` is obviously a metadata operation. For example, the object ID you get from `FSCTL_GET_OBJECT_ID` has nothing to do with the file contents. On the other hand, some I/O control codes do affect file contents. `FSCTL_SET_ZERO_DATA` zeroes out chunks of a sparse file, `FSCTL_FILE_LEVEL_TRIM` tells the underlying storage that it may (but is not required to) throw away the current contents of a section of a file. Since some I/O control codes affect file contents and some don't, you would have to say that the `DeviceIoControl` function on a synchronous file handle is sometimes serialized and sometimes not. It's not very reassuring to read documentation that goes something like "If the file handle is a synchronous file handle, the I/O control operation might be serialized, or it might not." Recall that all I/O in the kernel internally follows the asynchronous programming model. Synchronous file handles are a convenience provided by the kernel which converts operations on synchronous handles into the equivalent asynchronous operation, followed by a wait for the operation to complete; and all of these operations are serialized. Since drivers are allowed to make up custom I/O control codes, the I/O subsystem cannot know for certain whether a particular control code affects file contents or not. It wouldn't know whether any particular control code issued on a synchronous file handle should be converted to a synchronous operation or allowed to proceed

asynchronously. So now you're in an even more confused situation, where the kernel *doesn't even know* whether it should serialize a control operation or not. What would the documentation say now? "Actually, the kernel doesn't know whether the operation should be serialized, so it just flips a coin. Heads, the operation is serialized. Tails, it isn't. Do you feel lucky, punk?" So the kernel chooses a very simple algorithm: All operations are serialized. It doesn't care if an obvious file contents operation, a subtle file contents operation, a hidden file contents operation, or not a file contents operation at all. Sometimes the best way to solve a problem is to stop trying to be clever and just focus on predictability and reducing complexity. If you want to perform an operation that is not serialized, you can just create a second handle to the same file and perform the operation on the second handle. (The `ReOpenFile` function may come in handy.) Whether a file handle is synchronous or asynchronous is a property of the handle, not of the underlying file. There can be five handles to the same file, some synchronous and some asynchronous. Operations on the synchronous handles are serialized with respect to the file handle; operations on the asynchronous handles are not.

And with the decision to take the simple approach and serialize all accesses to a synchronous file handle, the kernel can wait on the `hFile` itself to determine when the I/O has completed, thereby saving it from having to create a temporary event for every single I/O operation.

Raymond Chen

Follow