How can I perform a CopyFile, but also flush the file buffers before the destination handle is closed?

devblogs.microsoft.com/oldnewthing/20221007-00

October 7, 2022

Raymond Chen

A customer wanted to copy a file, and the `CopyFile` function seemed to do the trick. However, they also wanted to perform a `FlushFileBuffers` after the copy completed, but there's a problem: The `FlushFileBuffers` function requires a handle to the file, and the `CopyFile` function doesn't give you a handle to the file.

One option is to replace the `CopyFile` with a manual `ReadFile` / `WriteFile` loop, and perform the `FlushFileBuffers` on the destination handle before closing it. However, this means that you give up all the features of `CopyFile`, like managing alternate data streams, extended attributes, security attributes, and all that other stuff.

Fortunately, there is a way to hook into the file copy operation.

Windows XP added the function `CopyFileEx` which lets you pass a callback function that is called to report on the progress of the file copy, and that callback function is our foot in the door. Among other things, the callback function is given a reason for the callback, as well as the file handles being used for both the source and destination of the copy.

What we can do is detect that the file copy has started copying a new stream. When that happens, we flush the file buffers of the old stream, and then duplicate the new destination handle so we can flush that guy when it finishes.

```
// WARNING! All error checking has been deleted
// for expository purposes!

BOOL CopyFileWithFlush(
    LPCWSTR sourceFileName,
    LPCWSTR destinationFileName,
    BOOL failIfExists)
{
    HANDLE mostRecentStream = nullptr;

    auto callback = [](
        auto totalSize,
        auto totalBytes,
        auto streamSize,
        auto streamBytes,
        auto streamId,
        auto reason,
        auto sourceHandle,
        auto destHandle,
        auto refdata
    ) -> DWORD
    {
        auto& mostRecentStream = *reinterpret_cast<HANDLE*>(refdata);
        if (reason == CALLBACK_STREAM_SWITCH) {
            if (mostRecentStream) {
                FlushFileBuffers(mostRecentStream);
                CloseHandle(mostRecentStream);
            }
            DuplicateHandle(
                GetCurrentProcess(), destHandle,
                GetCurrentProcess(), &mostRecentStream,
                0, false, DUPLICATE_SAME_ACCESS);
        }
        return PROGRESS_CONTINUE;
    };

    auto result = CopyFileEx(
        sourceFileName,
        destinationFileName,
        callback,
        &mostRecentStream,
        nullptr,
        failIfExists ? COPY_FILE_FAIL_IF_EXISTS : 0);

    if (mostRecentStream) {
        if (result) {
            FlushFileBuffers(mostRecentStream);
        }
        CloseHandle(mostRecentStream);
    }
    return result;
}
```

This is a lot of code, but it's actually not doing much.

We take action when we are told that the copy has started a new stream. First, we flush out the old stream, if any. We then close the old stream handle and duplicate the new destination handle, so that we can flush this new stream when it is finished.

If the copy completes successfully, we perform one last flush to flush out the last stream.

The nonsense with having to remember the most recent stream is a workaround for the fact that `CopyFileEx` does not notify you when it finishes the old stream. It only notifies you when it starts a new stream, so you have to infer that the old stream ended either when a new stream starts or when the entire operation completes.

If you can assume a minimum supported operating system of Windows 8, then you can use the newer `CopyFile2` function, which calls you back when a stream finishes:

```
// WARNING! All error checking has been deleted
// for expository purposes!

HRESULT CopyFileWithFlush(
    LPCWSTR sourceFileName,
    LPCWSTR destinationFileName,
    BOOL failIfExists)
{
    COPYFILE2_EXTENDED_PARAMETERS params{ sizeof(params) };
    params.dwCopyFlags = failIfExists ? COPY_FILE_FAIL_IF_EXISTS : 0;
    params.pProgressRoutine = [](auto message, auto) -> COPYFILE2_MESSAGE_ACTION
    {
        if (message->Type == COPYFILE2_CALLBACK_STREAM_FINISHED)
        {
            FlushFileBuffers(message->Info.StreamFinished.hDestinationFile);
        }
        return COPYFILE2_PROGRESS_CONTINUE;
    };

    return CopyFile2(sourceFileName, destinationFileName, &params);
}
```

If the callback is telling us that we just finished copying a stream, we sneak in and flush the buffers associated with that handle.

This technique can be extended to cover other operations you want to perform during the file copy. You can use the file handles of the source and destination to perform additional operations before everything gets closed.

Raymond Chen

Follow