Why does my single-byte write take forever?

devblogs.microsoft.com/oldnewthing/20110922-00

September 22, 2011

Raymond Chen

A customer found that a single-byte write was taking several seconds, even though the write was to a file on the local hard drive that was fully spun-up. Here's the pseudocode:

```
// Create a new file - returns quickly
hFile = CreateFile(..., CREATE_NEW, ...);
// make the file 1GB
SetFilePointer(hFile, 1024*1024*1024, NULL, FILE_BEGIN);
SetEndOfFile(hFile);
// Write 1 byte into the middle of the file
SetFilePointer(hFile, 512*1024*1024, NULL, FILE_BEGIN);
BYTE b = 42;
/ this write call takes several seconds!
WriteFile(hFile, &b, &nBytesWritten, NULL);
```

The customer experimented with using asynchronous I/O, but it didn't help. The write still took a long time. Even using `FILE_FLAG_NO_BUFFERING` (and writing full sectors, naturally) didn't help.

The reason is that on NTFS, extending a file reserves disk space but does not zero out the data. Instead, NTFS keeps track of the "last byte written", technically known as the *valid data length*, and only zeroes out up to that point. The data past the valid data length are logically zero but are not physically zero on disk. When you write to a point past the current valid data length, all the bytes between the valid data length and the start of your write need to be zeroed out before the new valid data length can be set to the end of your write operation. (You can manipulate the valid data length directly with the `SetFileValidData` function, but be very careful since it comes with serious security implications.)

Two solutions were proposed to the customer.

Option 1 is to force the file to be zeroed out immediately after setting the end of file by writing a zero byte to the end. This front-loads the cost so that it doesn't get imposed on subsequent writes at seemingly random points.

Option 2 is to make the file sparse. Mark the file as sparse with the `FSCTL_SET_SPARSE` control code, and immediately after setting the end of file, use the `FSCTL_SET_ZERO_DATA` control code to make the entire file sparse. This logically fills the file with zeroes without committing physical disk space. Anywhere you actually write gets converted from "sparse" to "real". This does open the possibility that a later write into the middle of the file will encounter a disk-full error, so it's not a "just do this and you won't have to worry about anything" solution, and depending on how randomly you convert the file from "sparse" to "real", the file may end up more fragmented than it would have been if you had "kept it real" the whole time.

Raymond Chen

Follow