On the various ways of creating large files in NTFS

devblogs.microsoft.com/oldnewthing/20150710-00

July 10, 2015

Raymond Chen

For whatever reason, you may want to create a large file.

The most basic way of doing this is to use `SetFilePointer` to move the pointer to a large position into the file (that doesn't exist yet), then use `SetEndOfFile` to extend the file to that size. This file has disk space assigned to it, but NTFS doesn't actually fill the bytes with zero yet. It will do that lazily on demand. If you intend to write to the file sequentially, then that lazy extension will not typically be noticeable because it can be combined with the normal writing process (and possibly even optimized out). On the other hand, if you jump ahead and write to a point far past the previous high water mark, you may find that your single-byte write lasts forever.

Another option is to make the file sparse. I refer you to the remarks I made some time ago on the pros and cons of this technique. One thing to note is that when a file is sparse, the virtual-zero parts do not have physical disk space assigned to them. Consequently, it's possible for a `WriteFile` into a previously virtual-zero section of the file may fail with an `ERROR_DISK_QUOTA_EXCEEDED` error.

Yet another option is to use the `SetFileValidData` function. This tells NTFS to go grab some physical disk space, assign it to the file, and to set the "I already zero-initialized all the bytes up to this point" value to the file size. This means that the bytes in the file will contain uninitialized garbage, and it also poses a security risk, because somebody can stumble across data that used to belong to another user. That's why `SetFileValidData` requires administrator privileges.

From the command line, you can use the `fsutil file setvaliddata` command to accomplish the same thing.

Bonus chatter: The documentation for `SetEndOfFile` says, "If the file is extended, the contents of the file between the old end of the file and the new end of the file are not defined." But I just said that it will be filled with zero on demand. Who is right?

The formal definition of the `SetEndOfFile` function is that the extended content is undefined. However, NTFS will ensure that you never see anybody else's leftover data, for security reasons. (Assuming you're not intentionally bypassing the security by using `SetFileValidData`.)

Other file systems, however, may choose to behave differently.

For example, in Windows 95, the extended content is not zeroed out. You will get random uninitialized junk that happens to be whatever was lying around on the disk at the time.

If you know that the file system you are using is being hosted on a system running some version of Windows NT (and that the authors of the file system passed their Common Criteria security review), then you can assume that the extra bytes are zero. But if there's a chance that the file is on a computer running Windows for Workgroups or Windows 95, then you need to worry about those extra bytes. (And if the file system is hosted on a computer running a non-Windows operating system, then you'll have to check the documentation for that operating system to see whether it guarantees zeroes when files are extended.)

Raymond Chen

Follow