How can I recognize file systems that don’t support 64-bit unique file identifiers?

devblogs.microsoft.com/oldnewthing/20220127-00

January 27, 2022

Raymond Chen

A customer observed that their attempt to find out whether two handles refer to the same file was not working because some remote file systems report zero for the volume serial number and unique file identifiers, regardless of what file you asked for. This caused them to mistakenly believe that all files were identical! What’s the correct way of identifying whether a file system supports unique file identifiers?

The meaning of the volume serial number is documented as

> **VolumeSerialNumber** (4 bytes): A 32-bit unsigned integer that contains the serial number of the volume. The serial number is an opaque value generated by the file system at format time, and is not necessarily related to any hardware serial number for the device on which the file system is located. No specific format or content of this field is required for protocol interoperation. This value is not required to be unique.

In other words, the volume serial number can mean anything it wants. There is no special meaning for the value of zero.

The documentation for the 64-bit file identifier is a little more helpful:

> **FileId** (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. For file systems which do not explicitly store directory entries named “..” (synonymous with the parent directory), an implementation MAY set this field to 0 for the entry named “..”, and this value MUST be ignored.[110]

Okay, so a value of zero for the FileId means that the file system doesn’t support 64-bit file IDs, or at least it doesn’t support 64-bit file IDs for the file you asked about. Therefore, if you get zero as the file ID, you should not use it as an indicator of uniqueness.

Chasing through to section 2.1.9 reveals some more information:

> For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored.
>
> For files for which a unique 64-bit file ID cannot be established, this field MUST be set to 0xffffffffffffffff, and MUST be ignored.

That second paragraph is talking specifically about ReFS, which uses 128-bit file identifiers. This is noted in the documentation for `BY_HANDLE_FILE_INFORMATION`:

> The ReFS file system, introduced with Windows Server 2012, includes 128-bit file identifiers. To retrieve the 128-bit file identifier use the GetFileInformationByHandleEx function with **FileIdInfo** to retrieve the FILE_ID_INFO structure. The 64-bit identifier in this structure is not guaranteed to be unique on ReFS.

What is going on is that if you ask for the 64-bit file identifier from a ReFS system, it will take the 128-bit identifier and see if the upper 64 bits are all zero. If so, then it gives you the lower 64 bits. But if not, then it gives you `` 0xFFFFFFFF`FFFFFFFF ``, to say “Yeah, um, I have a unique identifier, but I wasn’t able to squish it down into a 64-bit value”.

Therefore, you shouldn’t use `` 0xFFFFFFFF`FFFFFFFF `` as an indicator of uniqueness either.

Okay, so armed with this background information, let’s see if we can solve the original problem. We’ll do that next time.

Raymond Chen

Follow