Why doesn't the file system have a function that tells you the number of files in a directory?

devblogs.microsoft.com/oldnewthing/20090217-00

February 17, 2009

Raymond Chen

There are any number of bits of information you might want to query from the file system, such as the number of files in a directory or the total size of the files in a directory. Why doesn't the file system keep track of these things?

Well, of course, one answer is that it certainly couldn't keep track of every possible fragment of information anybody could possibly want, because that would be an infinite amount of information. But another reason is simply a restatement of the principle we learned last time: Because the file system doesn't keep track of information it doesn't need.

The file system doesn't care how many files there are in the directory. It also doesn't care how many bytes of disk space are consumed by the files in the directory (and its subdirectories). Since it doesn't care, it doesn't bother maintaining that information, and consequently it avoids all the annoying problems that come with attempting to maintain the information.

For example, one thing I noticed about many of the proposals for maintaining the size of a directory in the file system is that very few of them addressed the issue of hard links. Suppose a directory contains two hard links to the same underlying file. Should that file be double-counted? If a file has 200 hard links, then a change to the size of the file would require updating the size field in 200 directories, not just one as one commenter postulated. (Besides, the file size isn't kept in the directory entry anyway.)

Another issue most people ignored was security. If you're going to keep track of the recursive directory size, you have to make sure to return values consistent with *each user's* permissions. If a user does not have permission to see the files in a particular directory, you'd better not include the sizes of those files in the "recursive directory size" value when that user goes asking for it. That would be an information disclosure security vulnerability. Now all of a sudden that single 64-bit value is now a complicated set of values, each with a different ACL that controls which users each value applies to. And if you change the ACL on a file, the file system would have to update the file sizes for each of the directories that contains the file, because the change in ACL may result in a file becoming visible to one user and invisible to another.

Yet another cost many people failed to take into account is just the amount of disk I/O, particular writes, that would be required. Generating additional write I/O is a bad idea in general, particularly on media with a limited number of write cycles like USB thumb drives. One commenter did note that this metadata could not be lazy-written because a poorly-timed power outage would result in the cached value being out of sync with the actual value.

Indeed the added cost of all the metadata writes is one of the reasons why Windows Vista no longer updates the Last Access time by default.

Bonus chatter: My colleague Aaron Margosis points out a related topic over on the ntdebugging blog: *NTFS Misreports Free Space?* on the difficulties of accurate accounting, especially in the face of permissions which don't grant you total access to the drive.

Raymond Chen

Follow