Why is breadth-first searching better for file system tree walking?

devblogs.microsoft.com/oldnewthing/20050203-00

February 3, 2005

Raymond Chen

Earlier, Eric Lippert discussed one scenario where breadth-first searching is better than depth-first searching. Today, I'll tell you about another. If you go back to the old MS-DOS file enumeration functions, you'll find that there is a "Find first file" function and a "Find next file" function, but no "Find close" function. That's because the MS-DOS file enumeration functions maintained all their state in the find structure. The FAT file system was simple enough that the necessary search state fit in the reserved bytes and no external locking was necessary to accomplish the enumeration. (If you did something strange like delete a directory while there was an active enumeration on it, then the enumeration would start returning garbage. It was considered the program's responsibility not to do that. Life is a lot easier when you are a single-tasking system.) However, moving these functions to a network protocol or a non-FAT file system created problems. When enumerating files over a network, the server needs to allocate additional memory to track the enumeration, memory which it needs to free when the enumeration is complete. Similarly, non-FAT file systems may have additional state to track that (1) doesn't fit in the reserved space in the find structure, (2) **shouldn't** be stored in the reserved space (because it would compromise security), or (3) **can't** be stored in the reserved space (because the kernel needs to update it asynchronously when the state of the directory changes). Since MS-DOS has no "Find close" function, how do these alternate file systems know when it is safe to free the memory associated with the file search? [Context clarified – 10am] Clairevoyance is not yet a viable option, so the file system is forced to guess. Typically, a file enumeration is considered "abandoned" if it is not used for "a long time" or if too many pending file enumerations are in progress. Let's see how depth-first search compares to breadth-first search on these points. In a typical depth-first search, you recurse into a subdirectory as soon as you see it. Thus, if you are, say, six levels deep in recursion, you will have six active enumerations, one for each level. The most recent one is most active, and the one at the root is least active. This means that the enumeration at the root is likely to be considered "abandoned" by the server because it hasn't been used for the longest time, and because there is a lot of other enumeration activity going on in the meantime. On the other hand, with a breadth-first search, when a directory is found, it is not processed immediately but is rather put on a list for later consideration. Consequently, there is only one active enumeration, and it runs to completion before the next enumeration begins. This means that the enumeration is unlikely to be considered "abandoned" by the server. This problem is not purely theoretical. If you performed a depth-first search on a large directory tree on a network server from a MS-DOS program, you often found that the enumeration ended prematurely because the older enumerations became abandoned by the server. You also saw this from a Win32 program if you were enumerating against an older network server that was designed for MS-DOS clients (and which therefore doesn't implement FindClose). You can still use depth-first enumeration while avoiding the abandonment problem. Instead of recursing into a directory immediately after enumerating it, save it on a list. When you are finished enumerating the files in a directory, go back and process the list. You're still enumerating depth-first, but you are delaying the recursive call until after the directory enumeration has completed.

Explorer uses this "delayed recursion" trick to avoid the problem of prematurely-terminated enumerations when walking through directories on these old servers.

Raymond Chen

Follow