The source of much confusion: "backed by the system paging file"

March 1, 2013

Raymond Chen

Perhaps one of the most misunderstood sentences in the Win32 documentation is this little bit in the documentation for `CreateFileMapping`:

> If *hFile* is **INVALID_HANDLE_VALUE**, the calling process must also specify a size for the file mapping object in the *dwMaximumSizeHigh* and *dwMaximumSizeLow* parameters. In this scenario, **CreateFileMapping** creates a file mapping object of a specified size that is backed by the system paging file instead of by a file in the file system.

When people read the underlined portion, they interpret this to mean "The data in the file mapping object will be written to the system paging file." But that's not what it says. It says that it is *backed by* the system paging file. In other words, "If I need to page this memory out, I will store it in the system paging file."

Note the word "if".

Usually, people get all worked up about the description because "I don't want this data to be written to disk by the creator, and then read from the disk by the consumer. I want this to be stored in RAM, just like the memory I allocate with `HeapAllocate` or `VirtualAlloc`." Of course, what they didn't realize is that memory allocated with `HeapAllocate` and `VirtualAlloc` is *also* backed by the system paging file. If memory allocated by `Heap-Allocate` and `VirtualAlloc` needs to be paged out, the memory manager will write it to the paging file.

In other words, "backed by the system paging file" just means "handled like regular virtual memory."

If the memory is freed before it ever gets paged out, then it will never get written to the system paging file. Just like you wanted.

The documentation was written with kernel-colored glasses. They figured that you knew that paging file-backed memory was just a way of saying "normal pageable memory."

Exercise: What happens if paging is disabled? Where is the memory backed if there is no paging file?

Raymond Chen

Follow