Why is the Heap32Next function incredibly slow on Windows 7?

devblogs.microsoft.com/oldnewthing/20120323-00

March 23, 2012

Raymond Chen

A customer observed that the `Heap32Function` runs much slower on Windows 7 compared to previous versions of Windows. What happened?

Set the wayback machine to 1992. The product is Windows 3.1. One of the new components available in Windows 3.1 went by the name `TOOLHELP`. It let you snoop around the low-level guts of the Windows 3.1 kernel, and the feature that is relevant here is walking the heaps. Since Windows 3.1 was a cooperatively multitasking system, you could ensure that the heap was stable during your calls to `HeapFirst` and `HeapNext` by the simple process of not yielding control.

Mind you, ToolHelp was not part of the kernel itself. It was bolted onto the side. As I recall, ToolHelp arrived late on the scene, and the kernel folks didn't want to destabilize the kernel with any ToolHelp-related changes, so all the work done by ToolHelp was done "from the outside".

Windows 95 introduced 32-bit versions of the ToolHelp functions. I'm not sure why.

Where was I? Oh right, `Heap32Next`.

In the 32-bit version of ToolHelp, you could walk the heap of a process by calling `Heap32-First` and `Heap32Next`. As implemented in Windows 95, the `Heap32First` function allocated some memory to keep track of the state of the heap walk and stored it in the `HEAPENTRY32.dwResvd` field. The `Heap32Next` used this state to find the next heap block, and it finally freed the memory when the end of the heap was reached (and `Heap32Next` returned `FALSE`). This means that if you call `Heap32First` and do not walk the heap to completion but rather abandon the walk partway through, you leaked some memory. Unlike functions like `FindFirstFile` which have an explicit `FindClose` function to indicate that you are done with the enumeration (and allow the operating system to free the tracking state), there was no corresponding `Heap32Close` function. The only way to free the memory was to walk the heap to the end. The Windows 95 implementation also didn't handle the case that the heap changed while you were walking it.

But since the toolhelp library was intended for diagnostic purposes anyway (I mean, it's right there in the name: *tool help*), these weren't considered serious problems. Your debugging plug-in might use it to walk the heap looking for memory leaks, but you wouldn't deploy it in production, right?

The Windows NT folks didn't like that there was a memory leak built into the design. Since there was no way to ensure that the memory allocated by `Heap32First` was freed in the event the application wanted to abandon the heap walk, their solution was simply to free all allocated memory before returning from `Heap32First` and `Heap32Next`. If an application asks to walk the heap, the `Heap32First` takes a snapshot of the heap, returns the first heap block, then frees the snapshot. When the application calls `Heap32Next`, it takes a snapshot of the heap, returns the second heap block, then frees the snapshot. On the second call to `Heap32Next`, it takes a snapshot of the heap, returns the third heap block, then frees the snapshot. You get the idea.

As a result, walking the heap via `Heap32First` / `Heap32Next` is an $O(n^2)$ operation.

So why did this become slower in Windows 7?

Prior to Windows 7, the snapshot was taken in a fixed-size buffer that held information for around a quarter million heap entries. As a result, there was a hard limit on the worst-case cost of walking the heap via the toolhelp functions. In Windows 7, this hard limit was lifted because there were some diagnostic tools which were bumping into this limit. The kernel folks decided that it was better to have the functions be slow but correct rather than fast and incomplete. Since the limit was lifted, so too was the cap on the worst-case cost of walking the heap with `Heap32First` / `Heap32Next`.

Toolhelp was designed back in the days of co-operative multitasking and hasn't aged well. At this point, he's sort of this unwanted distant relative in the kernel. Nobody actually likes him, but when he shows up at the family reunion, you have to let him in.

By the way, the recommended way to walk the contents of the heap is to use the `HeapWalk` function. The `HeapWalk` function does not suffer from this problem; enumerating the entire heap via repeated calls to `HeapWalk` has total running time proportional to the number of heap blocks. Note that `HeapWalk` can only enumerate heap blocks from the current process. If you're doing cross-process heap walking for diagnostic purposes, then you're stuck with `Heap32First` / `Heap32Next`, but since you're just doing it for diagnostic purposes, correctness should be more important to you than performance.

Raymond Chen

Follow