How can I find the heap that a memory block originally came from, so I can free it properly?

devblogs.microsoft.com/oldnewthing/20210812-00

August 12, 2021

Raymond Chen

It is not uncommon for a program to split allocations among multiple heaps. For example, you might dedicate a separate heap for each thread, to reduce heap contention,[1] on the theory that the vast majority of memory blocks are allocated and freed by the same thread. But every so often, memory will be allocated by one thread and freed by another. How do you make sure that the memory gets freed back to the correct heap?

A customer wondered if they could start by calling `HeapFree` and freeing the memory from the heap associated with the current thread. If that fails, then iterate through all the other heaps until one of them accepts it. Is this a good idea?

No, it's not a good idea.

The `HeapFree` function requires that the pointer is to an active heap block from the same heap. If you pass a pointer to an already-freed heap block, or to a heap block from some other heap, or to something that isn't a heap block at all, then the behavior is undefined, and anything at all can happen.

There is no function to find out whether any particular pointer points to an active heap block from a particular heap. This is something that the heap expects you the program to be keeping track of. Heaps are not optimized for answering the question "Is this a pointer to an active heap block?" efficiently, preferring to optimize for other things like fast allocation, fast freeing, and fragmentation-resistance.

The standard solution for this problem is to wrap the heap allocations inside another allocation which remembers how the memory block was allocated.

```
struct alignas(MEMORY_ALLOCATION_ALIGNMENT) ALLOCATION_HEADER
{
    HHEAP heap;
};

void* MultiplexAllocate(DWORD flags, SIZE_T size)
{
  ASSERT(!(flags & ~(HEAP_GENERATE_EXCEPTIONS | HEAP_ZERO_MEMORY)));
  auto heap = ChooseHeap();
  auto header = reinterpret_cast<ALLOCATION_HEADER*>(
            HeapAlloc(heap, flags,
                      size + sizeof(ALLOCATION_HEADER)));
  if (!header) return nullptr;

  header->heap = heap;
  return header + 1;
}

void MultiplexFree(DWORD flags, void* p)
{
  ASSERT(flags == 0);
  if (p) {
    auto header = reinterpret_cast<ALLOCATION_HEADER*>(p) - 1;
    HeapFree(header->heap, flags, header);
  }
}
```

Bonus chatter: You might be tempted to call `GetProcessHeaps()` and use range checking to identify which heap a block of memory belongs to. However, this doesn't work because a heap is not required to consume contiguous memory. For example, large allocations typically get allocated via `VirtualAlloc` into memory blocks separate from the rest of the heap. And if the heap needs to grow, the memory allocation for the expanded heap is unlikely to be adjacent to the rest of the heap.

Bonus reading: Here's an old story about using multiple heaps to reduce fragmentation.

[1] The introduction of the low fragmenetation heap largely addresses this problem.

Raymond Chen

Follow