# How can I obtain the heap from which a heap block was allocated?

devblogs.microsoft.com/oldnewthing/20180531-00

May 31, 2018

Raymond Chen

Suppose profiling reveals that heap contention is a bottleneck in your program. Furthermore, suppose that your program's heap usage pattern is that most heap blocks are allocated and freed by the same thread. One way to address this problem is to create a pool of heaps, say *N* of them. Each thread gets assigned one of these heaps pseudo-randomly, so that contention is reduced by a factor of *N*. And if you have fewer than *N* threads, then you reduced contention almsot down to zero, leaving only the case where memory is freed from a thread different from the thread it was allocated from.

Allocating the memory from the correct heap is easy. When the thread wants to allocate memory, you look up the heap that was pseudo-randomly assigned to it and use that heap to allocate the memory.

Freeing the memory from the correct heap is harder. Given a heap block, how do you get the heap from which it was allocated? Is there some `HeapFindFromBlock` function that will give me a heap handle given a memory block?

Alas, there is no such function from the Windows heap manager. You'll have to keep track of the heap handle yourself. Typically this is done by over-allocating memory by a small amount and putting the handle of the originating heap in the extra memory. The code that frees the memory consults this value to know how to free the block.

Pseudo-code:

```
struct DECLSPEC_ALIGN(MEMORY_ALLOCATION_ALIGNMENT) HeapPrefix
{
    HANDLE heap;
};

void* malloc_thread_affine(size_t size)
{
  // Not shown here: Integer overflow detection.
  size_t actualSize = size + sizeof(HeapPrefix);

  HANDLE heap = GetPreferredHeapForCurrentThread();
  HeapPrefix* prefix = (HeapPrefix*)HeapAlloc(heap, flags, actualSize);
  if (!prefix) return NULL;

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

void free_thread_affine(void* p)
{
  if (!p) return;

  HeapPrefix* prefix = (HeapPrefix*)p - 1;
  HeapFree(p->heap, flags, prefix);
}
```

We define a `HeapPrefix` structure that goes in front of the memory we return to the client. In this structure, we record the heap from which the memory was allocated, and we consult this structure when freeing the memory to ensure that we free it from the proper heap.

Note that the `HeapPrefix` must be padded to ensure proper alignment of the client memory. We accomplish this by declaring a structure alignment.

**Exercise**: Why does the bookkeeping go at the front of the client memory, instead of at the end?

**Exercise 2**: Windows Vista introduced the `GetCurrentProcessorNumber` function. With it, you can set *N* equal to the number of processors and have each thread allocate from the heap associated with the processor it is running on at the moment. Discuss whether this is a good idea.

**Bonus reading**: Unmanaged Memory Fragmentation — an old story.

Raymond Chen

**Follow**