Memory allocation functions can give you more memory than you ask for, and you are welcome to use the freebies too, but watch out for the free lunch

devblogs.microsoft.com/oldnewthing/20120316-00

March 16, 2012

Raymond Chen

Memory allocation functions like `HeapAlloc`, `GlobalAlloc`, `LocalAlloc`, and `CoTaskMemAlloc` all have the property that they can return more memory than you requested. For example, if you ask for 13 bytes, you may very well get a pointer to 16 bytes. The corresponding `XxxSize` functions return the actual size of the memory block, and you are welcome to use all the memory in the block up to the actual size (even the bytes beyond the ones you requested). But watch out for the free lunch.

Consider the following code:

```
BYTE *GetSomeZeroBytes(SIZE_T size)
{
 BYTE *bytes = (BYTE*)HeapAlloc(GetProcessHeap(), 0, size);
 if (bytes) ZeroMemory(bytes, size);
 return bytes;
}
```

So far so good. We allocate some memory, and then fill it with zeroes. That gives us our zero-initialized memory.

Or does it?

```
BYTE *bytes = GetSomeZeroBytes(13);
SIZE_T actualSize = HeapSize(GetProcessHeap(), 0, bytes);
for (SIZE_T i = 0; i < actualSize; i++) {
 assert(bytes[i] == 0); // assertion fires!?
}
```

When you ask the heap manager for 13 bytes, it's probably going to round that up to 16, and when you call `HeapSize`, it may very well say, "Hey, I gave you three extra bytes. Don't need to thank me."

The problem comes when you try to reallocate the memory:

```
BYTE *ReallocAndZero(BYTE *bytes, SIZE_T newSize)
{
 return (BYTE*)HeapReAlloc(bytes, GetProcessHeap(),
                           HEAP_ZERO_MEMORY, newSize);
}
```

Here, you said, “Dear heap manager, please make this memory block bigger, and zero out the new bytes. Kthxbai.” And, assuming the heap manager was successful, you will indeed have a larger memory block, and the new bytes will have been zeroed out.

But the memory manager won’t zero out the three bonus bytes it gave you when you called `HeapAlloc`, because those bytes aren’t new. In fact, the heap manager assumes that you knew about those three extra bytes and were actively using them, and it would be rude to zero out those bytes behind your back.

Those bytes you didn’t know about since you didn’t check.

You might think the problem is that you mixed zero-allocation modes. You allocated the memory as “Go ahead and give me garbage, I’ll zero it out myself”, and then you reallocated it as “Can you zero it out for me?” The problem is that you and the heap manager disagree on how big *it* is. While you assume that the size of *it* is “the exact number of bytes I asked for”, the heap manager assumes that the size of *it* is “the exact number of bytes I gave you.” Those bytes in the middle fall through the cracks.

Therefore, you might try to fix it by changing your function like this:

```
BYTE *ReallocAndZero(BYTE *bytes, SIZE_T newSize)
{
 SIZE_T oldSize = HeapSize(GetProcessHeap(), bytes);
 BYTE *newBytes = (BYTE*)HeapReAlloc(bytes, GetProcessHeap(),
                                     0, size);
 if (newBytes && newSize > oldSize) {
  ZeroMemory(newBytes + oldSize, newSize - oldSize);
 }
 return newBytes;
}
```

But this doesn’t work, because of the reason we gave above: Your call to `HeapSize` will return the *actual* block size, not the requested size. You will therefore forget to zero out those three bytes you didn’t know about.

The real problem is in the `GetSomeZeroBytes` function. It decided to manually zero out the bytes it received, but it zeroed out only the bytes that were requested, not the actual bytes received.

One solution is to make sure to zero out *everything*, so that if it is reallocated, the extra bytes gained in the reallocation will also be zero.

```
BYTE *GetSomeZeroBytes(SIZE_T size)
{
 BYTE *bytes = (BYTE*)HeapAlloc(GetProcessHeap(), 0, size);
 if (bytes) ZeroMemory(bytes,
                       HeapSize(GetProcessHeap(), bytes));
 return bytes;
}
```

Another solution is to take advantage of the memory manager's `HEAP_ZERO_MEMORY` flag, which tells the memory manager to zero out the *entire block* of memory when it is allocated:

```
BYTE *GetSomeZeroBytes(SIZE_T size)
{
 return (BYTE*)HeapAlloc(GetProcessHeap(),
                         HEAP_ZERO_MEMORY, size);
}
```

... and to use the same flag when reallocating:

```
BYTE *ReallocAndZero(BYTE *bytes, SIZE_T newSize)
{
 return (BYTE*)HeapReAlloc(bytes, GetProcessHeap(),
                           HEAP_ZERO_MEMORY, size);
}
```

Most of the heap functions let you specify that you want the heap manager to zero out the memory for you, and that includes the bonus bytes. For example, you can use `GMEM_ZEROINIT` with the `GlobalAlloc` family of functions, and `LMEM_ZEROINIT` with the `LocalAlloc` family of functions. The annoying one is `CoTaskMemAlloc`, since it does not provide a flag for zero-allocation. You have to zero out the memory yourself, and you have to do it right. (The inspiration for today's article was a bug caused by not zeroing out the memory correctly.)

There are other implications of these bonus bytes. For example, if you use `CreateStreamOnHGlobal` to create a stream on an existing `HGLOBAL`, the function uses `GlobalSize` to determine the size of the stream it should create. And that value includes the bonus bytes, even though you may not have realized that they were there. Result: You create a stream of 13 bytes, but somebody who tries to read from it will get 16 bytes. You need to make sure that the code which reads from the stream won't get upset by those extra bytes. (For example, if you passed it to a function that concatenates streams, you just inserted three bytes of garbage between the streams.) You also need to be careful that those extra bytes don't leak any sensitive information if you, say, put the memory block on the clipboard for everyone to see.

Bonus chatter: It appears that at some point, the kernel folks decided that these "bonus bytes" were more hassle than they were worth, and now they spend extra effort remembering not only the actual size of the memory block but also the requested size. When you ask, "How big is this memory block?" they lie and return the requested size rather than the actual size.

In other words, the free bonus bytes are no longer exposed to applications by the kernel heap functions. Note, however, that *this behavior is not contractual*; future versions of Windows may start handing out free bonus bytes again. Note also that not all heap managers have done the extra work to remember the requested size, and they will continue to hand out bonus bytes. Therefore, you must continue to code defensively and assume that bonus bytes may exist (even if they usually don't). (And note that heap debugging tools may intentionally generate "bonus bytes" to help flush out bugs.)

Double extra bonus chatter: Note that this gotcha is not specific to Windows.

```
// resize a block of memory originally allocated by calloc
// and zero out the new bytes
void *crealloc(void *bytes, size_t new_size)
{
 size_t old_size = malloc_size(bytes);
 void *new_bytes = realloc(bytes, new_size);
 if (new_bytes && new_size > old_size) {
  memset((char*)new_bytes + old_size, 0, new_size - old_size);
 }
 return new_bytes;
}
```

Virtually all heap libraries have bonus bytes.

Raymond Chen

Follow