How to solve Memory Fragmentation

Asked 16 years, 3 months ago Modified 6 years, 1 month ago

Viewed 54k times

51

We've occasionally been getting problems whereby our long-running server processes (running on Windows Server 2003) have thrown an exception due to a memory allocation failure. Our suspicion is these allocations are failing due to memory fragmentation.

Therefore, we've been looking at some alternative memory allocation mechanisms that may help us and I'm hoping someone can tell me the best one:

1) Use Windows Low-fragmentation Heap

2) jemalloc - as used in Firefox 3

3) Doug Lea's malloc

Our server process is developed using cross-platform C++ code, so any solution would be ideally cross-platform also (do *nix operating systems suffer from this type of memory fragmentation?).

Also, am I right in thinking that LFH is now the default memory allocation mechanism for Windows Server 2008 / Vista?... Will my current problems "go away" if our customers simply upgrade their server os?

c++ windows memory

Share

Improve this question

Follow

asked Sep 13, 2008 at 21:04

Alan

13.7k ● 9 ● 45 ● 51

10 Answers

Sorted by: Highest score (default)

38

First, I agree with the other posters who suggested a resource leak. You really want to rule that out first.

Hopefully, the heap manager you are currently using has a way to dump out the actual total free space available in the heap (across all **free** blocks) and also the total number of blocks that it is divided over. If the average free block size is relatively small compared to the total free space in the heap, then you do have a fragmentation problem. Alternatively, if you can dump the size of the largest free block and compare that to the total free space, that will accomplish the same thing. The largest free block would be small relative to the total ***free*** space available across all blocks if you are running into fragmentation.

To be very clear about the above, in all cases we are talking about ***free*** blocks in the heap, not the allocated blocks in the heap. In any case, if the above conditions are not met, then you *do* have a leak situation of some sort.

So, once you have ruled out a leak, you could consider using a better allocator. ***Doug Lea's malloc*** suggested in the question is a very good allocator for general use applications and very robust *most* of the time. Put another way, it has been time tested to work very well for most any application. However, no algorithm is ideal for *all* applications and any management algorithm approach can be broken by the right pathelogical conditions against it's design.

Why are you having a fragmentation problem? - Sources of fragmentation problems are *caused* by the behavior of an application and have to do with greatly different allocation lifetimes in the same memory arena. That is, some objects are allocated and freed regularly while other types of objects persist for extended periods of time all in the same heap.....think of the longer lifetime ones as poking holes into larger areas of the arena and thereby preventing the coalesce of adjacent blocks that have been freed.

To address this type of problem, the best thing you can do is logically divide the heap into sub arenas where the lifetimes are more similar. In effect, you want a transient heap and a persistent heap or heaps that group things of similar lifetimes.

Some others have suggested another approach to solve the problem which is to attempt to make the allocation sizes more similar or identical, but this is less ideal because it creates a different type of fragmentation called

internal fragmentation - which is in effect the wasted space you have by allocating more memory in the block than you need.

Additionally, with a good heap allocator, like Doug Lea's, making the block sizes more similar is unnecessary because the allocator will already be doing a power of two size bucketing scheme that will make it completely unnecessary to artificially adjust the allocation sizes passed to malloc() - in effect, his heap manager does that for you automatically much more robustly than the application will be able to make adjustments.

Share Improve this answer

Follow

edited May 5, 2014 at 19:47

James Bedford

29k ● 8 ● 60 ● 64

answered Sep 14, 2008 at 2:12

Tall Jeff

9,984 ● 7 ● 46 ● 61

1 I like the idea of having both a transient and a persistent heap. This seems similar to the 'Generation' concept in modern garbage collectors. Is this idea widely implemented in manual memory allocation schemes? – Waylon Flinn Apr 16, 2009 at 23:27

@Waylon Flinn - Creating sub heaps is very common solution to memory fragmentation problems in embedded systems. Alternatively, many embedded systems attempt to solve such problems by allocating as much as possible up front and never freeing such longer living objects....in effect static allocation, at least to some extent. Not a bad approach if you have enough memory to dedicate to specific sub-

systems for the life of the system. – Tall Jeff Apr 25, 2009 at 20:44

17

I think you've mistakenly ruled out a memory leak too early. Even a tiny memory leak can cause a severe memory fragmentation.

Assuming your application behaves like the following:
Allocate 10MB
Allocate 1 byte
Free 10MB
(oops, we didn't free the 1 byte, but who cares about 1 tiny byte)

This seems like a very small leak, **you will hardly notice it when monitoring just the total allocated memory size**. But this leak eventually will cause your application memory to look like this:

.
.
Free – 10MB
.
.
[Allocated -1 byte]
.
.
Free – 10MB
.
.
[Allocated -1 byte]

.

.

Free – 10MB

.

.

This leak will not be noticed... until you want to allocate 11MB
Assuming your minidumps had full memory info included, I recommend using DebugDiag to spot possible leaks. In the generated memory report, **examine carefully the allocation count (not size)**.

Share Improve this answer Follow

edited Apr 29, 2012 at 9:07

answered Oct 12, 2008 at 10:46

Tal

1,769 ● 2 ● 12 ● 22

5

As you suggest, Doug Lea's malloc might work well. It's cross platform and it has been used in shipping code. At the very least, it should be easy to integrate into your code for testing.

Having worked in fixed memory environments for a number of years, this situation is certainly a problem, even in non-fixed environments. We have found that the CRT allocators tend to stink pretty bad in terms of performance (speed, efficiency of wasted space, etc). I

firmly believe that if you have extensive need of a good memory allocator over a long period of time, you should write your own (or see if something like dlmalloc will work). The trick is getting something written that works with your allocation patterns, and that has more to do with memory management efficiency as almost anything else.

Give dlmalloc a try. I definitely give it a thumbs up. It's fairly tunable as well, so you might be able to get more efficiency by changing some of the compile time options.

Honestly, you shouldn't depend on things "going away" with new OS implementations. A service pack, patch, or another new OS N years later might make the problem worse. Again, for applications that demand a robust memory manager, don't use the stock versions that are available with your compiler. Find one that works for *your* situation. Start with dlmalloc and tune it to see if you can get the behavior that works best for your situation.

Share Improve this answer

Follow

answered Sep 14, 2008 at 2:21

Mark

10.2k ● 3 ● 40 ● 41

2

@nsaners - I'm pretty sure the problem is down to memory fragmentation. We've analyzed minidumps that point to a problem when a large (5-10mb) chunk of memory is being allocated. We've also monitored the process (on-site and in development) to check for memory leaks - none were detected (the memory footprint is generally quite low).

Share Improve this answer

Follow

answered Sep 13, 2008 at 21:28

Alan

13.7k ● 9 ● 45 ● 51

2 You have to be careful about your conclusion here. Even a very small leak can cause massive fragmentation of the heap. Think of each 1-byte leak malloc() that is not freed as bullet hole in the heap. The blocks immediately adjacent can never coalesce. – Tall Jeff Sep 14, 2008 at 2:28

2

You can help reduce fragmentation by reducing the amount you allocate deallocate.

e.g. say for a web server running a server side script, it may create a string to output the page to. Instead of allocating and deallocating these strings for every page request, just maintain a pool of them, so your only allocating when you need more, but your not deallocating (meaning after a while you get the situation you not allocating anymore either, because you have enough)

You can use _CrtDumpMemoryLeaks(); to dump memory leaks to the debug window when running a debug build, however I believe this is specific to the Visual C compiler. (it's in crtdbg.h)

Share Improve this answer

Follow

edited Mar 31, 2015 at 6:33

Baldrick

11.8k ● 2 ● 33 ● 37

answered Sep 13, 2008 at 21:21

Fire Lancer
30.1k • 33 • 122 • 184

I'd suspect a leak before suspecting fragmentation.

For the memory-intensive data structures, you could switch over to a re-usable storage pool mechanism. You might also be able to allocate more stuff on the stack as opposed to the heap, but in practical terms that won't make a huge difference I think.

I'd fire up a tool like valgrind or do some intensive logging to look for resources not being released.

Share Improve this answer

Follow

answered Sep 13, 2008 at 21:10

nsanders
12.6k • 3 • 42 • 48

The problem does happen on Unix, although it's usually not as bad.

The Low-framgmentation heap helped us, but my co-workers swear by Smart Heap (it's been used cross platform in a couple of our products for years). Unfortunately due to other circumstances we couldn't use Smart Heap this time.

We also look at block/chunking allocating and trying to have scope-savvy pools/strategies, i.e., long term things

here, whole request thing there, short term things over there, etc.

Share Improve this answer

Follow

answered Sep 13, 2008 at 21:59

maccullt
2,829 ● 1 ● 19 ● 15

1

As usual, you can usually waste memory to gain some speed.

This technique isn't useful for a general purpose allocator, but it does have it's place.

Basically, the idea is to write an allocator that returns memory from a pool where all the allocations are the same size. This pool can never become fragmented because any block is as good as another. You can reduce memory wastage by creating multiple pools with different size chunks and pick the smallest chunk size pool that's still greater than the requested amount. I've used this idea to create allocators that run in O(1).

Share Improve this answer

Follow

answered Sep 13, 2008 at 22:55

dicroce
46.7k ● 31 ● 105 ● 149

I disagree with this idea. Increasing the size of all allocations creates INTERNAL fragmentation. If you can waste memory, it is better to just increase the total size of the heap - which by itself might avoid fragmentation from becoming an issue. ie: The heap unstability is avoided completely – Tall Jeff Sep 14, 2008 at 2:33

-1

if you talking about Win32 - you can try to squeeze something by using LARGEADDRESSAWARE. You'll have ~1Gb extra defragmented memory so your application will fragment it longer.

Share Improve this answer

Follow

answered Nov 29, 2016 at 11:09

Danil

885 ● 12 ● 11

-1

The simple, quick and dirty, solution is to split the application into **several process**, you should get fresh HEAP each time you create the process.

Your memory and speed might suffer a bit (swapping) but fast hardware and big RAM should be able to help.

This was old UNIX trick with daemons, when threads did not existed yet.

Share Improve this answer

Follow

answered Nov 3, 2018 at 17:40

kris2k

345 ● 3 ● 6