
Subject: Re: Visual C++ compiler option for >1GB of memory
Posted by [Martin Downing](#) on Thu, 14 Mar 2002 23:23:54 GMT
[View Forum Message](#) <> [Reply to Message](#)

I would have thought it entirely reasonable that the limit to process size is set by the OS, not VC++ or IDL. The article below implies that all Windows 2K variants can be set to use at least 4GB of data per process. I leave you to digest it as I have yet to use a machine with sufficient >1Gb RAM so have not looked into it further!

Martin

taken from: <http://www.unisysworld.com/monthly/2002/01/xpdefrag.shtml>

"Win2K addresses scalability in several ways. First, Win2K gives to memory-intensive applications the means to use larger amounts of virtual and physical memory than was possible in NT 4.0. How does more memory help an application's scalability? Most server applications must quickly process large amounts of data to perform well. For example, for a database server to scale, the server must handle large databases. Because accessing disks is slow compared with accessing main memory, a server performs best when the data it must access for a database query is located, or cached, in physical memory. NT (and Win2K) is a 32-bit OS that divides its 4GB of virtual memory that is addressable with 32 bits into a lower half, in which NT assigns to applications, and an upper half, where OS and device-driver code and data reside. Thus, NT 4.0 effectively limits an application to managing at most 2GB of data (3GB with the /3GB boot.ini switch on NT Server, Enterprise Edition-NTS/E). A Win2K enhancement, Address Windowing Extensions (AWE-some Microsoft marketing literature refers to AWE as Advanced Windowing Extensions), lets an application manage much more data.

AWE consists of four APIs that applications use to allocate and deallocate physical memory and to obtain references, or windows, in their address space to portions of physical memory. For example, on a system with 4GB of physical memory, a database application might allocate the majority of the memory for its cache. The application then creates windows to the portions of the cache that it must access as it processes database queries. When a query completes, the application closes the windows it created. Figure 1, page 46, shows an example of an application that has allocated physical memory and defined a window to a portion of the physical memory.

Intel recently introduced x86 processors and motherboards that support Physical Address Extension (PAE), a mechanism that lets OSs and applications access more than 4GB of physical memory, even though the processors still use 32-bit virtual addressing. PAE uses 36-bit physical addressing to support up to 64GB of physical memory, so an application using the AWE APIs can create data caches that are close to 16 times as large as the cache sizes possible in NT 4.0. At press time, Microsoft plans to make the AWE

APIs available for all versions of Win2K (i.e., Win2K Professional-Win2K Pro, Win2K Advanced Server-Win2K AS, and Win2K Datacenter Server-Datcenter). However, Win2K Pro and Win2K Server will support only as much as 4GB of physical memory. Win2K AS will support as much as 8GB of physical memory, and Datacenter will support as much as 64GB of physical memory."

--

Martin Downing,
Clinical Research Physicist,
Grampian Orthopaedic RSA Research Centre,
Woodend Hospital, Aberdeen, AB15 6LS.
Tel. 01224 556055 / 07903901612
Fax. 01224 556662

m.downing@abdn.ac.uk

"Peter Mason" <p.mason@syd.dem.csiro.au> wrote in message
news:Kh9k8.3\$IT2.4574@news0.optus.net.au...

> "Mark Rivers" <rivers@cars.uchicago.edu> wrote:

> <...>

>> That thread is not quite what I'm looking for. It implies that IDL
should

>> be able to allocate 2GB under NT 4.0. However, I have machines with 1GB
> of

>> RAM and 3GB of virtual memory, but the maximum memory IDL can allocate
is

>> almost exactly 1GB. I know I remember seeing a message that said that
>> Visual C++ has a compiler or linker switch to control the maximum amount
> of

>> heap space that an application can use. The message said that the
default

>> is 1GB, and it can be raised to 2GB.

>>

>> I want to write a simple C program to test this, seeing if I can
malloc()

>> more than 1GB in a Windows application. If so, then the next step is to

>> twist RSI's arm to build the next minor release of IDL for Windows with
> this

>> switch. In my current application the difference between 1GB and 2GB is
>> very significant.

>>

>> The longer-term solution is 64-bit IDL on the Itanium processors and
> 64-bit

>> Linux or Windows.

>>

>> Mark

>

> Hi Mark,

>

> I think that I might have posted the message that you mean. The post was

> sent by a previous version of this particular product (of society) that

was

> even more ignorant than the current version, believe it or not. (Even

the

> current version - Peter Mason version 43.2b - needs a BS flush to work

> properly.)

> But to get to the point...

> In my previous post I think I mentioned that I didn't have a PC with

> sufficient clout to test large allocations. That is still the case.

But

> I have done some actual digging this time to try to figure out how the

> Visual C heap works. Documentation on the nitty-gritty is scarce in

parts,

> but I think I have the general idea.

>

> According to the CRT source code (heapinit.c), VC versions 5 and 6 create

a

> win32 heap for their allocations.

> They create this heap with a maximum size of 0. In other words, this is

a

> growable Win32 heap. But if you read the "PlatSDK" entry for

HeapCreate()

> you'll see that even a growable heap like this will not deal with an

> allocation of 0x7FFF8 bytes (roughly half a megabyte) or more itself.

> Reportedly, such allocations are passed on to the Win32 function

> VirtualAlloc().

> If anything can allocate a large block of memory in Win32, I would think

> that it would have to be VirtualAlloc().

> If you have the "memory" (RAM + page file), then I suspect that the only

> thing that would prevent you from allocating the bulk of the accessible

2GB

> address space in one go, IN A VC PROGRAM, is the fragmentation of this

> address space. Unfortunately, such fragmentation is quite likely.

There

> might be some loaded DLLs (e.g., that pesky MFC DLL) or other allocations

> that are chopping it up in such a way that there just isn't a single free

> block much greater than 1GB. (On the other hand, I think that it is

> unlikely that you would need a single huge, contiguous slab of memory in

> your program?)

>

> I suggest that a basic test of your computer's address space would be to

> write a small stand-alone executable in C that used VirtualAlloc() to

> attempt the allocation of about 1.5GB of memory.

> A more realistic test (assuming that you are working with a number of
arrays
> and stuff) would be to try several allocations (like maybe 10-100MB each)
> totalling to about 1.7 GB or so.
>
> If you can allocate over 1GB in a stand-alone executable, then (as you
point
> out) the implication is that there is something that IDL is doing that is
> spoiling things. Some years back I when I was using Windows 95, I
noticed
> that IDL was using the "SmartHeap" memory manager on this platform. They
> might still be using a product like this, or they might have written their
> own memory-management routines. If so, a wild guess is that this
software
> might be reserving (but not necessarily committing) slabs of the address
> space for its heaps.
> As far as trying things out *here* is concerned, I'm not really clear on
how
> calling malloc() inside a DLL attached to IDL would work. (A DLL doesn't
> have its own stack or heap, reportedly.) But I would think that calling
> VirtualAlloc() from within the DLL would bypass anything that IDL is doing
> and allow you to test your PC's address space.
>
> A final suggestion: I don't know what the deal is with Windows 2000, but
> (reportedly) if you have Windows NT4 *SERVER*, Enterprise Edition rather
> than Windows NT Workstation, there's a boot-time option to get a 3GB user
> address space instead of a 2GB one. It might be worth your while to try
> this out.
>
>
> Hope this helps,
> Peter Mason (version 43.2 beta)
>
>
