Subject: Re: Memory problems update.
Posted by ali on Fri, 10 Jan 1992 00:14:18 GMT
View Forum Message <> Reply to Message

Bill Thompson (thompson@stars.gsfc.nasa.gov) asks several questions
about how IDL handles dynamic memory. I will try to explain what is
happening. To the best of my knowledge, this discussion applies
equally to PV~WAVE, although the version numbers are different from IDL.
If Unix people replace the term "pagefile quota" with "swap space",
this discussion applies equally to them.

> To test the memory limitations of IDL version 1 versus version 2.2.1, I
> ran the following program to see where it would crash.
>
> for i = 1,100
> j = 100*i
> print,j
> a = fltarr(j,j)
> endfor
> end
>
> I have a pagefile quota of 78471 blocks. I chose this method primarily
> because I had heard rumors that if one recreates an array that already
> exists with a size bigger than it had before, the old memory is not
> deallocated.
>
> IDL version 1 was unable to obtain enough memory to build the 2000x2000 array,
> while IDL version 2 was unable to build the 1400x1400 array.
>
> This seems to be consistent with IDL version 2 not deallocating the memory,
> since 1400x1400 would then be where the accumulated total would exceed the
> pagefile quota.
>
> Just saying "a = fltarr(2000,2000)" in a separate IDL 2.2.1 session does work,
> however, as does
>
> a = fltarr(1900,1900)
> a = fltarr(2000,2000)
>
> Does anybody know what causes this? Is the rumor I heard true?
>
> Bill Thompson

No, the rumor is not true. IDL does free its dynamic memory when it is
done with it, and the IDL code has has consistency checks that guard
against dynamic memory leaks. You can use the IDL command

to see how much memory IDL has allocated. However, just because IDL frees its memory does not mean that its virtual address space will shrink and reduce its demand on the pagefile. In fact, it will not.

This seeming contradiction is due to the fact that IDL uses the C language malloc(3) and free(3) functions to allocate and free memory. When a process calls free, the freed memory is not returned to the operating system. It is simply placed into a free list. The next time malloc is called, it will try to satisfy the request from this free list before it asks the operating system for more memory. Thus, the process size (and demand on the pagefile) represents the high water mark of memory usage, not its current usage.

It might be nice if free would return memory to the operating system and reduce the amount of pagefile currently claimed by the process, but this is not done. To show why, consider an example in which a process allocates two arrays, and then frees the first one. Here is what would happen:

1) The first request causes malloc to request memory from the operating system. The process address space is enlarged and the request is satisfied.
2) The second request causes the address space to be extended again.
3) The process frees the first array. free() wants to return its memory to the operating system and shrink the process size, but it can't because the second array follows it in the virtual address space and this would leave a hole.
4) At this point, free() could try to make things work by copying the second array into the space occupied by the first, but then it would have to search the process address and fix up any pointers to the second array to point at the new location. This is impossible because there is no way to tell such pointers from unrelated memory locations that happen to contain the same bit pattern. Some small memory systems such as the Apple Mac use double indirection for dynamic memory to get around such problems, but the resulting complexity and computational inefficiencies would rule against it in a larger system (in fact, Version 7 of the Mac OS has virtual memory and I expect use of "memory handles" to die off).

So, in Bill's example above, the dynamic memory is being freed, and it is gathering in the free list. Since each request is larger than the one before it, malloc makes the process grow to satisfy it, until the pagefile quota is hit. Arguably, malloc/free should be trying to

coalesce the free list to make large memory chunks out of small ones,
but for reasons known only to their author, it is not.

Summary:

Q: Is IDL freeing it's memory after use?
Yes

Q: Why does IDL Version 1 have better performance in this regard?
As a VMS-only program, IDL used different dynamic memory allocation
facilities (LIB$GET_VM() and LIB$FREE_VM()) which have different
behavior.

Q: Could IDL avoid using malloc for VMS?

No. It is hard to write a C program that uses the
C library and not use malloc.

Q: Given the behavior of Bill's above example, why does saying
"a = fltarr(2000,2000)" in a separate IDL 2.2.1 session
work, as does

a = fltarr(1900,1900)
a = fltarr(2000,2000)

This works because you haven't fragmented the virtual address
space with all the smaller requests that preceded the larger ones.

Q: What can be done?

- Use less memory.
- Reorder your memory requests to reduce fragmentation.
- Make more memory available by increasing your systems
pagefile size and process pagefile quotas.

As a side comment, a pagefile quota of 78471 (~39MB) is small to
moderate by current standards. We usually recommend that the pagefile
be 3-4 times the size of the systems primary memory. On a 16MB system,
you would probably make it 48-64 MB, or even more if you had lots of
users and/or large datasets to process.

I hope the above helps...

- Ali, RSI