Subject: Re: Memory allocation problem:
Posted by Dr. G. Scott Lett on Mon, 23 Feb 1998 08:00:00 GMT
View Forum Message <> Reply to Message

Followup report:
Unfortunately (or not), on all the other unix platforms, the situation is as David described it, and IDL will exhibit memory hysteresis.

Cheers,
Scott

Dr. G. Scott Lett wrote:

> I haven't yet checked this problem on all platforms, but IDL 5.1 beta frees
> memory on Linux and Windows. I'll let you know what I find out about
> other platforms next week.
>
> Regards,
> Scott
>
> I~nigo Garcia wrote:
>
>> Well, I was afraid of something like this... I still find it a bug, whatever you
>> say, they should find a way of freeing that memory !!! Can it be done with
>> pointers ?? In a simple way, please, my brain is too small to fight with those
>> beings.
>>
>> I~nigo.
>>
>> David Fanning wrote:
>>> This is a result of IDL being written in C and using the C library
>>> functions (malloc and free) for memory allocation. In most C libraries,
>>> memory that is freed is NOT returned to the operating system. The C
>>> program retains this memory and will reuse it for future calls to malloc
>>> (assuming that the new allocation will fit in the freed block).
>>>
>>> Another way of considering it is in terms of how memory allocation is
>>> done under UNIX. New memory is allocated using brk() or sbrk() which
>>> control the size of the data segment. These routines are called by
>>> malloc().
>>>
>>> Suppose you allocate 3 1 MB regions of memory under C.
>>>
>>> char *p1=(char *)malloc(3*1024*1024);
>>> char *p2=(char *)malloc(3*1024*1024);
>>> char *p3=(char *)malloc(3*1024*1024);
>>>
>>> Here's what your data segment would look like assuming malloc had to call

```
>>>  sbrk().
>>>
>>>   ------------------------------------------------------------ ---
>>>  prev stuff | overhead | 3MB | overhead | 3MB | overhead | 3MB |
>>>   ------------------------------------------------------------ ---
>>>                      ^               ^               ^     ^
>>>                      p1              p2              p3    end of
>>>  segment.
>>>
>>>  Now we free(p1).
>>>
>>>   ------------------------------------------------------------ ----
>>>  prev stuff | overhead | free | overhead | 3MB | overhead | 3MB |
>>>   ------------------------------------------------------------ ----
>>>                                    ^               ^     ^
>>>                                    p2              p3    end of
>>>  segment
>>>
>>>
>>>  Notice that the free memory is still in the data segment. If free had
>>>  called brk to reduce the size of the segment, the 3MB pointed to my p3
>>>  would be outside the data segment! SIGSEGV city! If free had moved the
>>>  allocated memory to lower addresses so the segment size could be reduced
>>>  without losing data, then p2 and p3 would point to invalid addresses, and
>>>  we'd be forced to use handles rather than pointers and call
>>>  GetPointerFromHandle() every time we wanted to access the memory. Ick!
>>>  Just like Windows!
>>>
>>>  Cheers,
>>>
>>>  David
```

--
======================
Dr. G. Scott Lett
slett@holisticmath.com
http://holisticmath.com/
======================