Subject: Re: Memory allocation problem:
Posted by Inigo Garcia on Fri, 20 Feb 1998 08:00:00 GMT
View Forum Message <> Reply to Message

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
> ---
> David Fanning, Ph.D.
> Fanning Software Consulting
> E-Mail: davidf@dfanning.com
> Phone: 970-221-0438
> Coyote's Guide to IDL Programming: http://www.dfanning.com/

```
--
                          \\|//
                          (o o)
 +----------------------oOOo-(_)-oOOo---------------------- --+
| I~nigo Garcia Ruiz                                          |
| Kapteyn Instituut             Phone:  +31-(0)50-3634083  |
| Landleven 12                  Fax:    +31-(0)50-3636100  |
| 9747 AD GRONINGEN (Netherlands)  e-mail: iruiz@astro.rug.nl |
 +----------------------------------------------------------- --+
```

Subject: Re: Memory allocation problem:
Posted by davidf on Fri, 20 Feb 1998 08:00:00 GMT
View Forum Message <> Reply to Message

I~nigo Garcia (iruiz@astro.rug.nl) writes:

> I think this is a bug in IDL, probably someone else has noticed it before:

Alas, it has been noticed, but it is no bug. :-)

> If I create a huge array, and then delete it, the allocated memory still remains
> !!! Look a clear example:
>
> IDL> a=fltarr(10000,50000)
> IDL> a=0
>
> The array is not there any more, so the allocated memory should be freed,

> shouldn't it ? But it is not. And I don't like the idea of exiting IDL everytime I
> decide to use some big temporary arrays, I find it ridiculous. If these 2 lines
> are within a routine, the problem is exctly the same.
>
> I am in a Sun UltraSparc, with Solaris and IDL 5.0.2.
>
> Please, any solutions will be appreciated.

Here is the relevant article by Eric Korpela from the IDL FAQ.

By Eric Korpela of Berkeley

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

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Memory allocation problem:
Posted by Dr. G. Scott Lett on Sat, 21 Feb 1998 08:00:00 GMT

View Forum Message <> Reply to Message

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

Subject: Re: Memory allocation problem:
Posted by Peter Mason on Sun, 22 Feb 1998 08:00:00 GMT
View Forum Message <> Reply to Message

On Fri, 20 Feb 1998, 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.

Since version 5, IDL has been using a 3rd-party memory allocation library
called "SmartHeap". (At least, I know this is true for Win95 and suspect
it's true for other platforms.)
(Check out http://www.microquill.com/ for info on SmartHeap.)

SmartHeap functions differently on different platforms, especially with "large"
allocations (>64K). One would hope that a product like this would generally
be able to return large allocations to the operating system, but apparently
they haven't yet got this right for some operating systems.
Still, I'd say that RSI is on the right track using this stuff (mostly for
other reasons). If there was an easy-to-use solution to the memory-freeing
problem that some of IDL's platforms still exhibit, I'd expect it to be in
a product like this, sooner or later.

I don't think that there's a way around this right now.
I admit that I haven't tried to do this kind of thing, but from what I know,
if there was a solution (and it would be platform-specific) I think it would be
an ugly and very risky one involving C code and a sound knowledge of IDL
internals. Also, it would only address the variables you know about. IDL
goes through a lot of temporary variables, and it would still handle these its
own way.

Peter Mason

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/
======================

Subject: Re: Memory allocation problem:
Posted by Karl Krieger on Mon, 23 Feb 1998 08:00:00 GMT
View Forum Message <> Reply to Message

On Fri, 20 Feb 1998, I~nigo Garcia wrote:

> I think this is a bug in IDL, probably someone else has noticed it before:
> If I create a huge array, and then delete it, the allocated memory still
> remains
> ...
> The array is not there any more, so the allocated memory should be freed,
> shouldn't it ? But it is not. And I don't like the idea of exiting IDL
> everytime I are within a routine, the problem is exctly the same.
>
> I am in a Sun UltraSparc, with Solaris and IDL 5.0.2.

The reason why memory is not returned to the system under UNIX is

has already been answered here. There are malloc/free routines with
garbage collection available for UNIX, which could be used by the
RSI people, however, I don't see the benefit at least for UNIX:

UNIX does a quite good job with respect to memory management. If you check
the process table using ps you will notice that only a fraction of a
process is kept in memory. Even if the total process memory space keeps
it's size after freeing a variable, the unused process pages are swapped
to disk if the physical memory is needed for other tasks.

For Windoze derivates and MacOS this might be a different story though.

Karl
--
Max-Planck-Institute for Plasma Physics
Boltzmannstr.2, 85740 Garching, Germany   Email: krieger@ipp.mpg.de

---

Subject: Re: Memory allocation problem:
Posted by Helge.Rebhan on Mon, 23 Feb 1998 08:00:00 GMT

View Forum Message <> Reply to Message

In article <34ED873B.BAB44C2E@astro.rug.nl>, I~nigo Garcia
<iruiz@astro.rug.nl> wrote:

> I think this is a bug in IDL, probably someone else has noticed it before:
> If I create a huge array, and then delete it, the allocated memory still
remains
> !!! Look a clear example:
>
> IDL> a=fltarr(10000,50000)
> IDL> a=0
>
> The array is not there any more, so the allocated memory should be freed,
> shouldn't it ? But it is not. And I don't like the idea of exiting IDL
everytime I
> decide to use some big temporary arrays, I find it ridiculous. If these
2 lines
> are within a routine, the problem is exctly the same.
>
> I am in a Sun UltraSparc, with Solaris and IDL 5.0.2.
>

Lucky Dude ;-) Try this kind of examples on a Mac where the poor memory
alloction
of IDL combines with the even worser memory management of the MacOS !

Things are most annoying for me with IDL 5.0 and ENVI 2.7 ! There is absolutly

no catch up of memory shortage in this software ! Oh yes , you get in ENVI
a message box saying 'Memory is getting low' watching at the same time
IDL crashing in the background ;-((This problem should be platform
independend (?)

BTW: Does anybody known about the 'optimised' memory settings for ENVI on the
Mac ? There are a few cache, buffer... settings which are not quite clear
to me ??

Servus, Helge

--
Sorry for this but please adjust e-mail address for direct reply