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/