Subject: Re: Idl is greedy, once it gets some memory it will NOT release it.
Posted by korpela on Tue, 16 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

In article <4de71m$et5@netline-fddi.jpl.nasa.gov>,
Ramanujam Raghunath <rxr@pacific.jpl.nasa.gov> wrote:
> Hello all:
> I have a question on IDL's memory management. My problem could also be in the
> swap space management in the SUNOS and IRIX 5.3, but I doubt it.
> The issue is, after an array is deleted in IDL, the memory it occupied does not
> seem to be released (is the impression you get when you make an enquiry on the
> swap space from the O/S (swap -s in IRIX)) until the IDL session is exited;
> but seems to have been when you make an enquiry on the memory in IDL, using
> help, /memory.

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!

Eric

--
Eric Korpela | An object at rest can never be
korpela@ssl.berkeley.edu | stopped.

Click here for more info.