Subject: Re: yet another idl memory question
Posted by Jean H. on Thu, 04 Jun 2009 16:29:58 GMT
View Forum Message <> Reply to Message

Hi,

Unless you are using pointers, you can not be sure that the same memory space will be used in the different function calls.... and you don't really have to care either, especially under Linux.

If you do the same processing on data of the same size, it is likely because your code is using memory and not releasing it. Two things come to mind: 1) you are creating a pointer but never destroy it and 2) you are saving your results... in which case you may want to save the variable(s) to a file, free the memory (data=0B) and start processing the next file.

Jean

jw2519@columbia.edu wrote:
> How do I force that IDL set aside a chunk of contiguous memory to be
> refilled by a variable of equal size at each iteration of a procedure?
>
> More specifically, I have a code which consists of three procedures.
> One of these serves as the wrapper, one uses the IDL OpenDAP utility
> to download the relevant data, and the third processes the data into
> the grid and format that I want. The code will iterate for as long as
> the wrapper indicates that processed data files need to be created. I
> am running this on a 64-bit linux machine (IDL 7.0).
>
> Each iteration of the code performs four OpenDAP calls; each call
> downloads a large (~750MB) structure. After several iterations, the
> code exits with an insufficient memory to allocate array error (or
> sometimes "error reading value array: read is for xxx bytes, but there
> are only yyy in the buffer"; always this error occurs within an
> OpenDAP call). Using help,/memory and several of the tricks and tools
> posted elsewhere, I have been able to substantially reduce the max
> memory usage, but this only nets me another one and a half iterations
> or so. So my guess is that the problem is that there is no longer a
> sufficiently large block of contiguous memory into which to download
> the structure. Each of the four structures is of equal size at each
> iteration. How can I set aside a block of contiguous memory so that
> IDL will reuse the same chunk for each structure every time i call the
> procedure? I've tried using a solitary common block (i.e. just within
> the procedure in question), and I've attempted to use pointers. The
> program runs in both cases, but still exits with the same error. Of
> course, I am not at all certain that I used either of these
> correctly...

>
> Any suggestions are greatly appreciated.

Subject: Re: yet another idl memory question
Posted by David Fanning on Thu, 04 Jun 2009 17:06:09 GMT
View Forum Message <> Reply to Message

Jean H. writes:

> in which case you may want to save the
> variable(s) to a file, free the memory (data=0B) and start processing
> the next file.

Of course, data=0B doesn't free *all* the memory,
and doing this many times leads, I suspect, to the memory
fragmentation that is the heart of the problem. I suggest
you use UNDEFINE. That really does release *all* the memory
associated with a variable. And it elegantly indicates
what the code is actually doing, too, a significant
advantage for people reading your code after you have
run off to the tropics with that hot financial analyst
over in the head shed. :-)

http://www.dfanning.com/programs/undefine.pro

Cheers,

David

--
David Fanning, Ph.D.
Coyote's Guide to IDL Programming (www.dfanning.com)
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: yet another idl memory question
Posted by Karl[1] on Thu, 04 Jun 2009 20:06:28 GMT
View Forum Message <> Reply to Message

On Jun 4, 11:06 am, David Fanning <n...@dfanning.com> wrote:
> Jean H. writes:
>> in which case you may want to save the
>> variable(s) to a file, free the memory (data=0B) and start processing
>> the next file.
>
> Of course, data=0B doesn't free *all* the memory,

> and doing this many times leads, I suspect, to the memory
> fragmentation that is the heart of the problem. I suggest
> you use UNDEFINE. That really does release *all* the memory
> associated with a variable. And it elegantly indicates
> what the code is actually doing, too, a significant
> advantage for people reading your code after you have
> run off to the tropics with that hot financial analyst
> over in the head shed. :-)
>
>   http://www.dfanning.com/programs/undefine.pro
>
> Cheers,
>
> David
>
> --
> David Fanning, Ph.D.
> Coyote's Guide to IDL Programming (www.dfanning.com)
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

I'm not sure that's the explanation. IDL variables consist of a small (about 40 bytes) structure. If the variable has a dynamic part, as in arrays, the dynamic part is allocated separately and the pointer placed in the IDL variable structure. data=0B just frees the dynamic part. Further, I think that the variable structures are allocated in groups (pools) so that IDL does not have to constantly alloc and free these small structures as variables activate and deactivate. So, an undefine will probably just put the variable back into a pool, with no storage impacts at all.

Fragmentation could be the issue, if the right pattern of dynamic allocations and frees exist. But given that this is on 64-bit Linux, I rather doubt it, as the virtual address space is huge and would be difficult to fragment with 750MB blocks in "a few iterations".

To the OP:

Who is actually allocating the memory? OpenDAP? Or your IDL application?

As mentioned before, be sure you are freeing the memory before the next iteration. Regardless of who allocates it, leaving it unfreed will probably bump you up against your process size limit in Linux.

Does OpenDAP allow you to pass in a variable that you have defined/ allocated yourself? You seem to imply that that this is possible since you mention creating your own pointer vars, etc. Are you sure that OpenDAP is actually using the block you pass in and not allocating a new one anyway?

I think you are on the right track for the ideal solution. You want to determine the required size and allocate the block once. Then convince OpenDAP to use it to store the data, and then just keep reusing it. You are fortunate here in that the data size does not change. If it were variable, you'd either have to alloc a worst-case block and live with that or possibly suffer fragmentation issues by allocing/freeing each time.

I think that the key to the answer is determining how OpenDAP allocs this memory and how to convince it to reuse the same block. And you'd need to know more about OpenDAP to determine all of that. Sorry, I don't know much about it.

Karl

Subject: Re: yet another idl memory question
Posted by pgrigis on Mon, 08 Jun 2009 18:22:10 GMT

View Forum Message <> Reply to Message

On Jun 4, 1:06 pm, David Fanning <n...@dfanning.com> wrote:
> Jean H. writes:
>> in which case you may want to save the
>> variable(s) to a file, free the memory (data=0B) and start processing
>> the next file.
>
> Of course, data=0B doesn't free *all* the memory,
> and doing this many times leads, I suspect, to the memory
> fragmentation that is the heart of the problem. I suggest
> you use UNDEFINE. That really does release *all* the memory
> associated with a variable.

Really? Certainly it's not a substitute for ptr_free, is it?
At least not in my system:

```
help,/mem
;heap memory used:     924074, max:    1972734, gets:      574,
frees:      135

;create pointer
a=ptr_new(bytarr(2LL^28))
help,/mem
;heap memory used:  269360586, max:  269360761, gets:      582,
frees:      140

.comp ~/undefine.pro   ;makes sure I am using DF's undefine
```

```
;% Compiled module: UNDEFINE.
undefine,a

help,a
;A               UNDEFINED = <Undefined>

help,/mem
;heap memory used:  269362262, max:  270427272, gets:      613,
frees:      164

heap_gc
help,/mem
;heap memory used:     926762, max:  269362382, gets:      624,
frees:      174
```

Ciao,
Paolo

> And it elegantly indicates
> what the code is actually doing, too, a significant
> advantage for people reading your code after you have
> run off to the tropics with that hot financial analyst
> over in the head shed. :-)
>
>   http://www.dfanning.com/programs/undefine.pro
>
> Cheers,
>
> David
>
> --
> David Fanning, Ph.D.
> Coyote's Guide to IDL Programming (www.dfanning.com)
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")