Subject: Releasing temporary variables created with IDL_MakeTempArray()
Posted by dg86 on Wed, 01 Apr 2015 11:36:31 GMT
View Forum Message <> Reply to Message

Dear Folks,

A DLM I've written in C uses
    IDL_MakeTempArray()
to allocate an array, let's call it ARR, that gets passed back to IDL.
I'd like to release that temporary variable in IDL without having
to call a routine in the DLM that invokes IDL_Deltmp() .

1. Can I just do something like
    IDL> ARR = 0
to release the temporary variable? Or is there another IDL-based method?

2. At a deeper level, say that I've created a pointer to ARR with
    IDL> PARR = ptr_new(ARR, /no_copy)
Will freeing the pointer with
    IDL> ptr_free, PARR
also release the temporary variable?

3. Is there a way to see how many temporary variables I have checked
out? Is there a limit to that number? What's the limit?

Many thanks,

David

Subject: Re: Releasing temporary variables created with IDL_MakeTempArray()
Posted by dg86 on Thu, 02 Apr 2015 00:38:54 GMT
View Forum Message <> Reply to Message

On Wednesday, April 1, 2015 at 7:36:36 AM UTC-4, David Grier wrote:
> Dear Folks,
>
> A DLM I've written in C uses
>   IDL_MakeTempArray()
> to allocate an array, let's call it ARR, that gets passed back to IDL.
> I'd like to release that temporary variable in IDL without having
> to call a routine in the DLM that invokes IDL_Deltmp() .
>
> 1. Can I just do something like
>   IDL> ARR = 0
> to release the temporary variable? Or is there another IDL-based method?
>
> 2. At a deeper level, say that I've created a pointer to ARR with

> IDL> PARR = ptr_new(ARR, /no_copy)
> Will freeing the pointer with
> IDL> ptr_free, PARR
> also release the temporary variable?
>
> 3. Is there a way to see how many temporary variables I have checked
> out? Is there a limit to that number? What's the limit?
>
> Many thanks,
>
> David

Following up on my earlier post, I have some additional insights
and a couple of questions for folks who really understand IDL internals.

As originally written, my program allocated a steady stream of temporary variables
using IDL_MakeTempArray() at a rate of 30 allocations per second. After about 20 minutes,
the whole system would lock up hard (no mouse or keyboard, no ssh). This behavior
was reproducible.

Just doing the math suggests that lock-ups occur after 2^15 allocations.

If, for the sake of argument, the number of allocated temporary variables were
maintained internally with a signed short integer, failures might arise when that
counter rolls over.

Attempting to release the temporary variable with methods 1 and 2 as I originally
proposed does not fix the problem. I even tried passing the temporary
array back to C to delete the variable (using IDL_DelTemp) , but the variable no
longer has its "temporary" flag set.

What works is to release the variable explicitly at the end of the same
C routine that creates it, just like in the documentation and example code.
This isn't useful, however, because I want to work with the variable in IDL.

This leaves me with a question: What is the "correct" way to allocate a temporary
variable in C, pass the variable back to IDL, and then free the variable's resources
in IDL?

Also, am I right about the upper limit on the number of temporary variables that can
be allocated in IDL?

All the best,

David

P.S. I've fixed my immediate problem by allocating my temporary variable
just once and feeding it back to my C library for updating (30 times per second).

The C code is messier, but seems to run reliably.

Subject: Re: Releasing temporary variables created with IDL_MakeTempArray()
Posted by chris_torrence@NOSPAM on Thu, 02 Apr 2015 01:46:58 GMT
View Forum Message <> Reply to Message

On Wednesday, April 1, 2015 at 6:38:58 PM UTC-6, David Grier wrote:
> Following up on my earlier post, I have some additional insights
> and a couple of questions for folks who really understand IDL internals.
>
> As originally written, my program allocated a steady stream of temporary variables
> using IDL_MakeTempArray() at a rate of 30 allocations per second. After about 20 minutes,
> the whole system would lock up hard (no mouse or keyboard, no ssh). This behavior
> was reproducible.
>
> Just doing the math suggests that lock-ups occur after 2^15 allocations.
>
> If, for the sake of argument, the number of allocated temporary variables were
> maintained internally with a signed short integer, failures might arise when that
> counter rolls over.
>
> Attempting to release the temporary variable with methods 1 and 2 as I originally
> proposed does not fix the problem. I even tried passing the temporary
> array back to C to delete the variable (using IDL_DelTemp) , but the variable no
> longer has its "temporary" flag set.
>
> What works is to release the variable explicitly at the end of the same
> C routine that creates it, just like in the documentation and example code.
> This isn't useful, however, because I want to work with the variable in IDL.
>
> This leaves me with a question: What is the "correct" way to allocate a temporary
> variable in C, pass the variable back to IDL, and then free the variable's resources
> in IDL?
>
> Also, am I right about the upper limit on the number of temporary variables that can
> be allocated in IDL?
>
> All the best,
>
> David
>
> P.S. I've fixed my immediate problem by allocating my temporary variable
> just once and feeding it back to my C library for updating (30 times per second).
> The C code is messier, but seems to run reliably.

Hi David,

I'm at home right now, so I can't check the C code to see if there is a hard limit on the # of temporary variables, but I'm a bit confused as to how you could get into this situation.

Normally, if you write a C routine for IDL, at the end of the routine, you return your result as a temporary variable (created say using IDL_MakeTempArray, MakeTempVector, Gettmp, etc.). As soon as you pass the variable back to IDL, you no longer own it, and you should never need to Deltmp it yourself. If your IDL code takes that function result and assigns it to a new variable, then that new variable will no longer be a temporary, and the temp will be immediately returned to the pool (the actual array data will get handed off to the new variable).

So, if your code looks like:

result = MyCFunction(...)

Then result is now a named variable containing the array data, and your temporary is long gone.

Doing this in a loop should work "forever". So, your code must look somewhat different?

-Chris

Subject: Re: Releasing temporary variables created with IDL_MakeTempArray()
Posted by chris_torrence@NOSPAM on Fri, 03 Apr 2015 15:44:54 GMT
View Forum Message <> Reply to Message

On Thursday, April 2, 2015 at 4:24:15 AM UTC-6, David Grier wrote:
> I'll push the revised (stably working) code onto github later today. The new
> version has the clunker signature
> IDL> image = idlpgr_retrievebuffer(context, pgrimage, image)
> and has been running without incident for the past 12 hours, which is
> essentially forever.
>
> All the best,
>
> David

Hi David,

I looked over your code on github, and I'm still puzzled. You should be able to call your idlpgr_retrievebuffer indefinitely with no problem (it should be the same as calling something like findgen). So my gut feeling is that there is something else going on - perhaps your camera routines are not releasing memory properly, and memory is getting fragmented?

-Chris

Subject: Re: Releasing temporary variables created with IDL_MakeTempArray()

Posted by Jim Pendleton on Fri, 03 Apr 2015 21:08:10 GMT
View Forum Message <> Reply to Message

One other thing to look for... IDL_MakeTempArray() obviously uses some internal bookkeeping above and beyond the underlying malloc() calls to flag data for subsequent heap-freeing operations.

Data allocated this way shows up in HELP, /MEM and the MEMORY() function at the IDL level. You can watch the high water mark rise and fall over time if these data are being freed correctly.

Data allocated outside the context of IDL or with lower level routines such as malloc() will not be accounted for in HELP, /MEM for example.

Depending on your OS, you might use top, Task Manager or Process Explorer to watch the memory use for the entire IDL process. If you find that your process memory grows but HELP, /MEM doesn't show an increase in the high water mark, then the culprit is likely elsewhere in the DLM.

Jim P.
"I work for Exelis, too"

Subject: Re: Releasing temporary variables created with IDL_MakeTempArray()
Posted by dg86 on Mon, 06 Apr 2015 14:52:48 GMT
View Forum Message <> Reply to Message

On Friday, April 3, 2015 at 5:08:14 PM UTC-4, Jim P wrote:
> One other thing to look for... IDL_MakeTempArray() obviously uses some internal bookkeeping above and beyond the underlying malloc() calls to flag data for subsequent heap-freeing operations.
>
> Data allocated this way shows up in HELP, /MEM and the MEMORY() function at the IDL level. You can watch the high water mark rise and fall over time if these data are being freed correctly.
>
> Data allocated outside the context of IDL or with lower level routines such as malloc() will not be accounted for in HELP, /MEM for example.
>
> Depending on your OS, you might use top, Task Manager or Process Explorer to watch the memory use for the entire IDL process. If you find that your process memory grows but HELP, /MEM doesn't show an increase in the high water mark, then the culprit is likely elsewhere in the DLM.
>
> Jim P.
> "I work for Exelis, too"

Dear Jim and Chris,

Thanks for these very helpful pointers. I've been AFK, and will be testing possible solutions

later this week. I'll follow up with updates on success (or failure).

All the best,

David

Subject: Re: Releasing temporary variables created with IDL_MakeTempArray()
Posted by dg86 on Thu, 16 Apr 2015 16:02:39 GMT
View Forum Message <> Reply to Message

On Monday, April 6, 2015 at 10:52:52 AM UTC-4, David Grier wrote:
> On Friday, April 3, 2015 at 5:08:14 PM UTC-4, Jim P wrote:
>> One other thing to look for... IDL_MakeTempArray() obviously uses some internal bookkeeping above and beyond the underlying malloc() calls to flag data for subsequent heap-freeing operations.
>>
>> Data allocated this way shows up in HELP, /MEM and the MEMORY() function at the IDL level. You can watch the high water mark rise and fall over time if these data are being freed correctly.
>>
>> Data allocated outside the context of IDL or with lower level routines such as malloc() will not be accounted for in HELP, /MEM for example.
>>
>> Depending on your OS, you might use top, Task Manager or Process Explorer to watch the memory use for the entire IDL process. If you find that your process memory grows but HELP, /MEM doesn't show an increase in the high water mark, then the culprit is likely elsewhere in the DLM.
>>
>> Jim P.
>> "I work for Exelis, too"
>
> Dear Jim and Chris,
>
> Thanks for these very helpful pointers. I've been AFK, and will be testing possible solutions
> later this week. I'll follow up with updates on success (or failure).
>
> All the best,
>
> David

I've figured out what works, and what doesn't, but not why. Everything hinges on what I do with the data returned from my DLM.

1. What doesn't work: creating a pointer to the temporary variable that was allocated in C:

```
ptr_free, self._data
self._data = ptr_new(idlpgr_RetrieveBuffer(self.context, self.image), /no_copy)
```

This is a small snippet from the definition of the dgghwpointgrey object. The whole thing is on github at https://github.com/davidgrier/idlpgr . The idlpgr_RetrieveBuffer routine allocates a temporary variable to return an array of image data. The quoted stanza frees the pointer
to the previous image and creates a new pointer to the new image.

After about 20 minutes of acquiring images at 30 frames per second, this stanza causes my system
to lock up hard. No mouse, no keyboard, no network, nothing. Cycling the power is the only way to recover. Strangely enough, HELP,/MEM shows nothing untoward, right up to the point of failure.
Memory usage is constant, and well below the max.

2. What does work: copying the data from the temporary variable that was allocated in C:

```
*self._data = idlpgr_RetrieveBuffer(self.context, self.image)
```

This runs for at least 24 hours without incident. Here, self._data is a pointer to a previously allocated array of the correct size and type to accept the data from the DLM's subroutine.

I'm satisfied enough with this solution, even though I'd've liked to have avoided copying the data for the sake of efficiency.

My one remaining question is: Why does approach #1 fail, and why does it take so long to fail?

All the best,

David

Subject: Re: Releasing temporary variables created with IDL_MakeTempArray()
Posted by Matthew Argall on Fri, 24 Apr 2015 17:06:13 GMT
View Forum Message <> Reply to Message

> My one remaining question is: Why does approach #1 fail, and why does it take so long
> to fail?

This may be off the wall, but I have always wondered about it. Heap variables have a heap identifier number. By creating and freeing so many heap variables, could you be rolling over the integer heap identifier?

You could use the /GET_HEAP_IDENTIFIER keyword to Ptr_Valid() to check.
