Subject: Re: IDL Memory Leaks
Posted by David Fanning on Mon, 05 Nov 2001 18:22:14 GMT
View Forum Message <> Reply to Message

Myron Brown (brownmz1@jhuapl.edu) writes:

> Recently, I have noticed that my IDL programs leak memory, but I never
> use pointers directly. This is true when running with IDL on a Windows
> PC or on an SGI workstation. Widgets seem to be one source of
> problems. File I/O seems to be another, but I'm not yet sure. Due to
> the problems I'm having with memory leaks, my long runs eventually die
> when memory is exhausted.
>
> Does anyone have any hints on ways to avoid memory leaks in IDL?

Well, be *very* careful whose programs you use.
Those ones from that Coyote site are notorious
for having memory leaks. :-)

Having embarrassed myself with memory leaks far too
often for it to be amusing anymore, I have learned
a couple of things about the subject. Here are some
rules of thumb I use.

1. In widget programs put your clean-up routines
in a procedure that is called when the top-level
base dies. (In other words, use the CLEANUP keyword
to the XMANAGER call.) *Don't* put your clean-up
routines in a QUIT button event handler. People
don't exit your programs with the QUIT button!

2. Put your CLEANUP procedure (in widget or object
programs) VERY close to the GUI or INIT procedures
in your program file.

Most memory leaks come from adding a pointer to the
program somewhere during development and forgetting
to put the complementary cleanup in the CLEANUP
procedure. Having it very close by helps a lot.

3. Don't create a pointer without *immediately*
adding the line that cleans it up in the CLEANUP
routine (see 2, above).

4. If you are putting something new into a pointer,
always do this:

```
*ptr = somethingNew
```

Or, if you have to, this:

```
Ptr_Free, ptr
ptr = Ptr_New(somethingNew)
```

But, never this:

```
ptr = Ptr_New(somethingNew)
```

5. Before you show your program to *anyone*,
(a) start a new IDL session, (b) run your program,
(c) exit the program, and (d) type "HELP, /HEAP".
If anything exists on the heap, immediately read
steps 1-4 again.

That (and damn careful programming) should help. :-)

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL Memory Leaks
Posted by John-David T. Smith on Mon, 05 Nov 2001 18:24:53 GMT
View Forum Message <> Reply to Message

Myron Brown wrote:
>
> Recently, I have noticed that my IDL programs leak memory, but I never
> use pointers directly. This is true when running with IDL on a Windows
> PC or on an SGI workstation. Widgets seem to be one source of
> problems. File I/O seems to be another, but I'm not yet sure. Due to
> the problems I'm having with memory leaks, my long runs eventually die
> when memory is exhausted.
>
> Does anyone have any hints on ways to avoid memory leaks in IDL?
>
> Please reply to my e-mail address, since I don't often use newsgroups.
>
> Thanks.

Myron:

Side point: comp.lang.idl is an altogether different newsgroup, unrelated to RSI's IDL. Interface Definition Language, or some such. Just FYI, because I'm sure they're as confused by our posts as we are by theirs.

I can suggest a few places to look:

1. Pointers. Are you *sure* the widgets you use aren't compound widgets with pointers in them, or that all your I/O routines are pointer-free? You can, at any point, find out what's on the object/pointer heap by:

IDL> help,/heap

If anything is on the heap, look for dangling pointer variables with:

IDL> heap_gc,/verbose

which will tell you *which* pointers were causing memory leaks. Repeating this a few times can help you zero in on the culprit.

2. Widget's with large UVALUEs inside. Widgets are in many ways just like pointers: they live on a global heap, are referenced by a unique ID, and can point to not one, but several different values of different sizes. This is perhaps why David Fanning used empty base widget's UVALUEs as pointers in the good old days before "real" pointers were introduced by RSI (possibly prompted by the embarrassment he was causing them ;). If you're sticking very large things (like arrays) inside widget UVALUE's, and not properly destroying these widgets, you'll be in exactly the same boat as with pointer leaks.

Unfortunately, unlike pointers, there's not a nice way of investigating what's on the "widget heap" at a given time, that I know of. You can use the help,/MEMORY command to try to isolate big jumps, and even to analyze the size being added. If it's close to an array size you're sticking in a widget, then that's a very strong clue.

Good luck,

JD

Subject: Re: IDL Memory Leaks
Posted by Myron Brown on Mon, 05 Nov 2001 20:57:41 GMT
View Forum Message <> Reply to Message

Thanks for the reply. Here's more information...

1. One source of leaks was widgets, so as a first cut, I removed that code. I can later go back and fix that problem, but for now, widgets are not needed or used.
2. I have no calls to the PTR routines in my code at all.
3. Put simply, my code consists of an IDL routine that calls another IDL routine over and over in a for loop. The routine that is run over and over (1) reads data, (2) performs some processing of that data and (3) writes results to file. This continues until memory is exhausted.
4. The code includes repetitive reassignments of variables to large arrays, even within a routine (still in scope). I tried to assign variables to zero each time, but it doesn't seem to make a difference. Should this really be necessary?
5. File I/O is done using ASSOC.
6. Structures are used.

It "appears" that something isn't being deallocated somewhere and that these things add up to consume all of the available memory. Since I'm not using widgets anymore and I'm not using pointers directly, the only thing I can guess is that IDL may have issues with memory leaks under "some" circumstances, or perhaps there are routines that, when used, one should be careful with. If that's true, then I'd just like to program around these circumstances. Otherwise, I'm at a loss.

Thanks again.

Myron

David Fanning wrote:

> Myron Brown (brownmz1@jhuapl.edu) writes:
>
>> Recently, I have noticed that my IDL programs leak memory, but I never
>> use pointers directly. This is true when running with IDL on a Windows
>> PC or on an SGI workstation. Widgets seem to be one source of
>> problems. File I/O seems to be another, but I'm not yet sure. Due to
>> the problems I'm having with memory leaks, my long runs eventually die
>> when memory is exhausted.
>>
>> Does anyone have any hints on ways to avoid memory leaks in IDL?
>
> Well, be *very* careful whose programs you use.
> Those ones from that Coyote site are notorious
> for having memory leaks. :-)
>
> Having embarrassed myself with memory leaks far too
> often for it to be amusing anymore, I have learned

> a couple of things about the subject. Here are some
> rules of thumb I use.
>
> 1. In widget programs put your clean-up routines
> in a procedure that is called when the top-level
> base dies. (In other words, use the CLEANUP keyword
> to the XMANAGER call.) *Don't* put your clean-up
> routines in a QUIT button event handler. People
> don't exit your programs with the QUIT button!
>
> 2. Put your CLEANUP procedure (in widget or object
> programs) VERY close to the GUI or INIT procedures
> in your program file.
>
> Most memory leaks come from adding a pointer to the
> program somewhere during development and forgetting
> to put the complementary cleanup in the CLEANUP
> procedure. Having it very close by helps a lot.
>
> 3. Don't create a pointer without *immediately*
> adding the line that cleans it up in the CLEANUP
> routine (see 2, above).
>
> 4. If you are putting something new into a pointer,
> always do this:
>
> *ptr = somethingNew
>
> Or, if you have to, this:
>
> Ptr_Free, ptr
> ptr = Ptr_New(somethingNew)
>
> But, never this:
>
> ptr = Ptr_New(somethingNew)
>
> 5. Before you show your program to *anyone*,
> (a) start a new IDL session, (b) run your program,
> (c) exit the program, and (d) type "HELP, /HEAP".
> If anything exists on the heap, immediately read
> steps 1-4 again.
>
> That (and damn careful programming) should help. :-)
>
> Cheers,
>
> David

> --
> David W. Fanning, Ph.D.
> Fanning Software Consulting
> Phone: 970-221-0438, E-mail: david@dfanning.com
> Coyote's Guide to IDL Programming: http://www.dfanning.com/
> Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL Memory Leaks
Posted by David Fanning on Mon, 05 Nov 2001 21:21:51 GMT
View Forum Message <> Reply to Message

Myron Brown (brownmz1@jhuapl.edu) writes:

> 4. The code includes repetitive reassignments of variables to large arrays,
> even within a routine (still in scope). I tried to assign variables to zero
> each time, but it doesn't seem to make a difference. Should this really be
> necessary?

I'd like to see how you are doing this (I hope you are using the TEMPORARY function!), as this is more than likely the source of your problem. You don't say how you know you are "leaking memory", but this is the kind of thing that will eat up process memory if you are not careful.

If you have the same variable on both sides of an equal sign, you really need something like this:

```
image = Temporary(image) * 5
```

And when you are done with it:

```
Undefine, image
```

(The UNDEFINE program can be found on my web page.)

Otherwise, you continually get more and more process memory (with malloc), and there is no way to give it back.

"Is it necessary?". Yes, it is very, very necessary. :-)

Cheers,

David
--
David W. Fanning, Ph.D.

Subject: Re: IDL Memory Leaks
Posted by Myron Brown on Mon, 05 Nov 2001 21:58:50 GMT
View Forum Message <> Reply to Message

David,

Thanks. I am using the TEMPORARY function. I know memory is leaking because the Windows task manager indicates that my IDL process is consuming more and more memory as time goes on. Of course, IDL doesn't realize this (i.e. HELP, /MEMORY indicates no increase in memory usage).

Two questions:

1. How does UNDEFINE differ from setting the variable to zero (a and b below)? The result appears to be the same, but is it?

(a) IF (N_Elements(varname) NE 0) THEN tempvar = Size(Temporary(varname))
(b) varname = 0

2. When my routine returns, does the memory allocated in that routine not get free'd since the variables are no longer in scope? Are there cases when this is not true?

Myron

David Fanning wrote:

> Myron Brown (brownmz1@jhuapl.edu) writes:
>
>> 4. The code includes repetitive reassignments of variables to large arrays,
>> even within a routine (still in scope). I tried to assign variables to zero
>> each time, but it doesn't seem to make a difference. Should this really be
>> necessary?
>
> I'd like to see how you are doing this (I hope you
> are using the TEMPORARY function!), as this is more
> than likely the source of your problem. You don't say
> how you know you are "leaking memory", but this is the
> kind of thing that will eat up process memory if you
> are not careful.
>
> If you have the same variable on both sides of an equal

> sign, you really need something like this:
>
>   image = Temporary(image) * 5
>
> And when you are done with it:
>
>   Undefine, image
>
> (The UNDEFINE program can be found on my web page.)
>
> Otherwise, you continually get more and more process
> memory (with malloc), and there is no way to give it
> back.
>
> "Is it necessary?". Yes, it is very, very necessary. :-)
>
> Cheers,
>
> David
> --
> David W. Fanning, Ph.D.
> Fanning Software Consulting
> Phone: 970-221-0438, E-mail: david@dfanning.com
> Coyote's Guide to IDL Programming: http://www.dfanning.com/
> Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL Memory Leaks
Posted by John-David T. Smith on Mon, 05 Nov 2001 22:22:19 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Myron Brown (brownmz1@jhuapl.edu) writes:
>
>> 4. The code includes repetitive reassignments of variables to large arrays,
>> even within a routine (still in scope). I tried to assign variables to zero
>> each time, but it doesn't seem to make a difference. Should this really be
>> necessary?
>
> I'd like to see how you are doing this (I hope you
> are using the TEMPORARY function!), as this is more
> than likely the source of your problem. You don't say
> how you know you are "leaking memory", but this is the
> kind of thing that will eat up process memory if you
> are not careful.
>
> If you have the same variable on both sides of an equal

> sign, you really need something like this:
>
> image = Temporary(image) * 5
>
> And when you are done with it:
>
> Undefine, image
>
> (The UNDEFINE program can be found on my web page.)
>
> Otherwise, you continually get more and more process
> memory (with malloc), and there is no way to give it
> back.
>
> "Is it necessary?". Yes, it is very, very necessary. :-)
>

While I agree that managing memory leaks can be challenging, I'm not sure these Draconian measures are indicated. Indeed:

image = image * 5

does waste some memory: exactly as much as "image" occupies, since it is made into a temporary variable on the right, then modified, and copied to the variable of the same size on the left. As soon as the temporary variable goes out of scope on the right (i.e. after this call), it's memory is returned. It is true that IDL doesn't give back to the system memory it has allocated. It *does*, however, recycle the memory it already has, and only ask for more memory if it really, really needs it.

Consider:

```
pro mem_assign_leak
  print,'   ===>Startup:'
  help,/memory
  a=dist(1024)
  print,'   ===>First allocation'
  help,/memory
  print,'   ===>Assignment'
  for i=0,4 do begin
     a=a*5
     help,/memory
  endfor
end

IDL> mem_assign_leak
   ===>Startup:
heap memory used:     403269, max:    4597588, gets:     1575,
```

```
frees:     1201
  ===>First allocation
heap memory used:    4597645, max:    4610149, gets:     2091,
frees:     1716
  ===>Assignment
heap memory used:    4597645, max:    8792021, gets:     2092,
frees:     1717
heap memory used:    4597645, max:    8792021, gets:     2093,
frees:     1718
heap memory used:    4597645, max:    8792021, gets:     2094,
frees:     1719
heap memory used:    4597645, max:    8792021, gets:     2095,
frees:     1720
heap memory used:    4597645, max:    8792021, gets:     2096,
frees:     1721
```

Here we see around 4MB allocated for the large array (the 400KB or so just represents IDL startup overheads). Notice that on each subsequent iteration, no additional memory is on the heap at all. Also notice that *max* jumps after the first iteration to 8MB... exactly the amount required for that temporary copy of "a" which was made on the right hand side, and discarded. This 8MB is enough to hold the entire calculation, which is then performed in subsequent iterations in the same memory IDL had already allocated from the system. After the first iteration, no new memory is requested.

Now try it from a fresh session with "a=temporary(a)*5" instead:

```
IDL> mem_assign_leak
  ===>Startup:
heap memory used:     400431, max:     420977, gets:      901,
frees:      653
  ===>First allocation
heap memory used:    4595719, max:    4608223, gets:     1424,
frees:     1172
  ===>Assignment
heap memory used:    4595719, max:    4595719, gets:     1424,
frees:     1172
heap memory used:    4595719, max:    4595719, gets:     1424,
frees:     1172
heap memory used:    4595719, max:    4595719, gets:     1424,
frees:     1172
heap memory used:    4595719, max:    4595719, gets:     1424,
frees:     1172
heap memory used:    4595719, max:    4595719, gets:     1424,
frees:     1172
```

Here we see the same usage (more or less), with the difference that even

the *max* usage remained at 4MB, since no temporary copy of "a" was made in this case (it's simply modified "in place"). There's one more difference. Notice the "gets" and "frees". In this case, after the first allocation, no more memory is "gotten" or "freed". In the former case, everytime through the loop, both "gets" and "frees" are incremented by exactly one: the memory required for that temporary array is gotten and then freed, leaving the max memory constant: i.e. no new system memory is obtained, despite 4 more "gets" and "frees" of memory.

So, with temporary, you can at most halve the max usage (aka system memory being eaten), but you can't correct for an error which grabs and hangs onto more memory within each new iteration. That's a memory leak, and it can be your fault (most common), or a subtle bug in an IDL builtin; hopefully you can find out which.

JD

Subject: Re: IDL Memory Leaks
Posted by Mark Hadfield on Tue, 06 Nov 2001 01:02:14 GMT
View Forum Message <> Reply to Message

From: "Myron Brown" <brownmz1@jhuapl.edu>
> It "appears" that something isn't being deallocated somewhere
> and that these things add up to consume all of the available
> memory. Since I'm not using widgets anymore and I'm not
> using pointers directly, the only thing I can
> guess is that IDL may have issues with memory leaks under
> "some" circumstances, or perhaps there are routines that, when
> used, one should be careful with. If that's true, then I'd just like
> to program around these circumstances. Otherwise, I'm at a loss.

I guess the only way to pin this down is to keep on stripping stuff out of your code until the memory leak stops (or doesn't). Can you get it into a form where you could post it on the group for others to play with? I know David & JD would love to have a go.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield
National Institute for Water and Atmospheric Research

--

Subject: Re: IDL Memory Leaks
Posted by Jaco van Gorkom on Tue, 06 Nov 2001 13:16:46 GMT
View Forum Message <> Reply to Message

Myron Brown wrote:
> ... I know memory is leaking because the
> Windows task manager indicates that my IDL process is consuming more and more
> memory as time goes on. Of course, IDL doesn't realize this (i.e. HELP, /MEMORY
> indicates no increase in memory usage).

This sounds like it could also be a memory fragmentation problem. One possible test and/or remedy for this would be to assign one huge chunk of memory at the start of the IDL session:
  void = BYTARR(VeryLargeNumberOfBytes, /NOZERO)
  DELVAR, void
RSI has a Tech Tip about this topic at
  http://www.rsinc.com/services/output.cfm?tip_id=533 ,
and there is also a Tech Tip on HELP, /MEMORY versus Windows Task Manager at
  http://www.rsinc.com/services/output.cfm?tip_id=1677 .

Cheers,
 Jaco

Subject: Re: IDL Memory Leaks
Posted by Stein Vidar Hagfors H[1] on Tue, 06 Nov 2001 14:23:12 GMT
View Forum Message <> Reply to Message

Myron Brown <brownmz1@jhuapl.edu> writes:

> David,
>
> Thanks. I am using the TEMPORARY function. I know memory is
> leaking because the Windows task manager indicates that my IDL
> process is consuming more and more memory as time goes on. Of
> course, IDL doesn't realize this (i.e. HELP, /MEMORY indicates no
> increase in memory usage).
>
> Two questions:
>
> 1. How does UNDEFINE differ from setting the variable to zero (a and
> b below)? The result appears to be the same, but is it?

In fact, you're quite correct. The amount of space used to store an undefined variable is exacly the same as that used to store a scalar zero..

--

---------- ----------
Stein Vidar Hagfors Haugan
ESA SOHO SOC/European Space Agency Science Operations Coordinator for SOHO

NASA Goddard Space Flight Center, Email: shaugan@esa.nascom.nasa.gov
Mail Code 682.3, Bld. 26, Room G-1, Tel.: 1-301-286-9028/240-354-6066
Greenbelt, Maryland 20771, USA. Fax: 1-301-286-0264
---------- ----------

Subject: Re: IDL Memory Leaks
Posted by pit on Tue, 06 Nov 2001 15:15:27 GMT
View Forum Message <> Reply to Message

Myron Brown <brownmz1@jhuapl.edu> writes:

> 4. The code includes repetitive reassignments of variables to large
> arrays, even within a routine (still in scope). I tried to assign
> variables to zero each time, but it doesn't seem to make a
> difference. Should this really be necessary?

I'd guess this as the problem. At least I found a similar thing in my codes. I had a repetitive

```
bispektrum = complexarr ( 2 * cutoff + 1, 2 * cutoff + 1, $
                          2 * Mask_Depth + 1, Mask_Depth + 1 )
```

in there, and it was eating up all available memory after some time.
I had to change it to setting the variable to zero,

```
bispektrum(*) = null
```

then the problems were gone.

This was in IDL 4.01. I have never checked again if the problem is still there...

Pit

PS: Please try not to do unneccessary full quotes. Thank you

--
Dr. Peter "Pit" Suetterlin http://www.astro.uu.nl/~suetter
Sterrenkundig Instituut Utrecht
Tel.: +31 (0)30 253 5225 P.Suetterlin@astro.uu.nl

Subject: Re: IDL Memory Leaks
Posted by John-David T. Smith on Tue, 06 Nov 2001 17:59:30 GMT

JD Smith wrote:

> 2. Widget's with large UVALUEs inside. Widgets are in many ways just
> like pointers: they live on a global heap, are referenced by a unique
> ID, and can point to not one, but several different values of different
> sizes. This is perhaps why David Fanning used empty base widget's
> UVALUEs as pointers in the good old days before "real" pointers were
> introduced by RSI (possibly prompted by the embarrassment he was causing
> them ;).

Correction, kindly pointed out by an RSI developer... the feature
introduced to put a halt to David's perverse misuse of base widgets was
handles, not pointers. Interestingly, he also filled me in that
handles in fact *were* base widgets internally, stripped of everything
by the uvalue. I had obviously attempted to elide those from my memory,
and not without due cause. Let's hope we remember handles only as a
curious side-excursion in IDL history, and insist on pointers in all our
code.

JD