Subject: Re: CALL_EXTERNAL puzzle (still) ?
Posted by Armand J. L. Jongen on Fri, 04 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

After trying to follow this discussion, there are a few comments which
might be helpfull to you:

>> However, why can't I pass a pointer?
>
> Good question - I would have liked to be able to exploit
> pointers inside external code, but RSI has (at the moment)
> specifically forbidden that:
>
> Direct access to pointer and object reference heap
> variables (types IDL_TYP_PTR and IDL_TYP_OBJREF,
> respectively) is not allowed.
>
> (External dev. guide, "Heap Variables")
>
> So, although the pointer is passed, you cannot do anything
> with it. Please, RSI? Why not some function like
>
> IDL_VPTR IDL_GetHeapVariable(HVID)

There is a way! At least I think :-))
When developing an intermediate DLL to control a framegrabber I ran into
the following problems:

Problem:

The ID of the framegrabber was of an unknown structure (protected by the
manufacturer for future development..... sounds like RSI:-)), but I
wanted to "know" inside IDL..... and pass the pointer to that unknown
structure back into DLL for other procedures.....

Solution:

I defined a pointer in a structure

info={DevPointer : Ptr_New()}

Beacause you cannot change it in the DLL (as RSI told us) I thought I
might just as well let it be returned by the Dll!

Then the C-code looked a bit like this:

```
Unknown_ID WINAPI FG_OpenDevice( LONG lArgc, LPVOID lpvArgv)
{
```

```
 LPLONG    lplArgv = NULL;
 CHAR    szMsg[256];

 lplArgv    = (LPLONG)lpvArgv;

    /* Try to open the device */

if (OpenDevice(&DevicePointer) != 0) {
 return -200;
}
 return DevicePointer;
}
```

where the Unknown_ID was that protected structure. The IDL code to get
the DevPointer was like:

```
Ptr_Free, info.DevPointer
info.DevPointer = Ptr_New(Call_External('idl_fg.dll', 'FG_OpenDevice'))
```

When wanting to pass the DevPointer as a pointer to the DLL (as I didn't
know the structure of the Unknown_ID this was the only way) I passed the
dereferenced pointer !!!!by reference!!!! like

```
lResult = Call_External('idl_fg.dll', 'FG_SomeProcedure',
(*(info.DevPointer)), $
  Value=[0b])
```

Then IDL passes in fact the pointer to the DLL, just what I wanted!!

>> And if I want to pass a pointer, and
>> print the value of another pointer just before the CALL_EXTERNAL, why is the
>> wrong one passed?
>
> These are how your "bad luck" arose, apparently. Probably the
> print statement caused the text to be stored right after the
> space where the first pointer was located? Expect *no*
> consistency from platform to platform, or IDL version to IDL
> version!

When developing the application where I used the above mentioned
technique, at one point everything seemed to work perfectly! Foolproof
so to say. The only thing was that when I started the application for
the second time in one IDL session, it crashed. Restarting IDL made it
work again! After three nights of how, why, hell etc... I found out that
some akward ID of a widget! was used for the ID of a framebuffer and
fortunately (or unfortunately:-(((whichever way you look at it) these
where the same when starting a fresh IDL session!!! The message: if

things work, it is not always so that things are running the way you
think they do!

Hopefully all this is not to much (crap), and maybe a bit usefull.

A last note (It's almost weekend:-):

When passing (BOLD) structures (BOLD) by reference, IDL does not pass
the normal pointer to the DLL. In fact it makes a copy of the structure
and passes the pointer to that copy into the operand. Therefore only the
copy is changed and not the original! A workaround is to make a copy
yourself before passing, and assigning the (altered) copy back to the
structure after your DLL-call. Example:

```
tmp_RefVolt = (*ptr).RefVolt ; Here I expect a value

Result = Call_External('idl_fg.dll', 'FG_GetRefVoltage', $
       (*ptr).InputSource, tmp_RefVolt, Value=[1b,0b])

(*ptr).RefVolt = tmp_RefVolt
```

In these cases it is again better to let the DLL return the wanted
values as shown above. When passing dereferenced pointers by refernce
thing work as expected so e.g.

```
info={image:Ptr_New()}

Ptr_Free, info.image
info.image=Ptr_New(bytarr(256,256))

lResult = Call_External('idl_fg.dll', 'FG_GetFrame', $
  (*(info.Image)), Value=[0b])
```

fills the array with the captured image (provided the DLL is written
properly:-))

Enough for now, have a nice weekend!

Cheers,

--
 ** ************
Armand J.L. Jongen   Academic Medical Centre
                                  Laser Centre
Phone   +31-20-5667419   \\||||//   Meibergdreef 9
Fax   +31-20-6975594 | ~ ~ | 1105 AZ Amsterdam
E-mail a.j.jongen@amc.uva.nl   [| o o |]   The Netherlands
 ****************************o00o***(__)***o00o************* ************

Subject: Re: CALL_EXTERNAL puzzle (still) ?
Posted by davidf on Fri, 04 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Rose (rmlongfield@my-dejanews.com) writes:

> However, why can't I pass a pointer? And if I want to pass a pointer, and
> print the value of another pointer just before the CALL_EXTERNAL, why is the
> wrong one passed?

I'm certainly not going to improve on Stein Vidar's excellent answer, but I did want to give you the short version: you can't pass pointers and (really) variables because those pointers and variables are not really what you *think* they are. In other words, even though we call these things "pointers" in IDL, they are not the same thing as the "pointers" over in your C program. To believe otherwise is to invite strange behavior in your program, as you have discovered.

The same is mostly true of variables as well. A variable in IDL is a fairly complicated structure. What Call_External does is strip out the *data* portion of this structure and pass it to your C program. Since it does this invisibly, it is easy to think that you are "passing your variable". You are doing no such thing. (This is, by the way, why you MUST create the variable or allocate memory for it on the IDL side and not on the C side. Variable creation makes the whole big thing that "describes" the real variable. However, as Stein Vidar points out, this can be done in your C program if you were using LinkImage.) When the data comes back from the C program, IDL puts it back into its larger variable structure. This is why it is essential that your C program doesn't change the size or type of the variable. If it did, the real IDL variable would have bogus information about itself and all hell would break loose. :-)

Cheers,

David

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

Subject: Re: CALL_EXTERNAL puzzle (still) ?
Posted by steinhh on Fri, 04 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

> Thanks for your comments, but ... after some additional testing, I still
> maintain that something is not clear about CALL_EXTERNAL. Firstly,
> regardless of the unconventional setup of my program, it *worked*. What
> bothered me was that I didn't understand why (i.e. it worked for reasons that
> would not be clear if one reads the code).

Hmm - I agree that this is extremely confusing - having
the code work for some plain wrong reason... It's only a freak
thing, as far as I can see. Your original example caused
the following output on my machine (AlphaServer, Digital Unix).

```
IDL> example
de-referenced pointer: 961081302
*IDL____Value of argc: 1
                Segmentation fault
```

As far as I can see, your original IDL program sends (through argv[0]) a
pointer to an IDL_VARIABLE structure containing an IDL_TYP_PTR variable
(see "Type Codes" in external development guide).

My guess is that out of "bad luck", your string was
stored in the memory locations right after this IDL_VARIABLE
structure, and that the actual values in the IDL_VARIABLE
itself came out as non-printable characters. I.e., your
C pointer "filename" pointed to a memory area containing:

```
 [IDL_VARIABLE(as byte values)] [STRING TEXT]
 00 01 01 02 02 01 04 05 02 02  "This is the file name"
```

and when you printf'ed it, it looked like you were doing
the right thing, because what appeared on the screen was
"This is the file name".

> However, why can't I pass a pointer?

Good question - I would have liked to be able to exploit
pointers inside external code, but RSI has (at the moment)
specifically forbidden that:

> Direct access to pointer and object reference heap
> variables (types IDL_TYP_PTR and IDL_TYP_OBJREF,
> respectively) is not allowed.

(External dev. guide, "Heap Variables")

So, although the pointer is passed, you cannot do anything
with it. Please, RSI? Why not some function like

 IDL_VPTR IDL_GetHeapVariable(HVID)

> And if I want to pass a pointer, and
> print the value of another pointer just before the CALL_EXTERNAL, why is the
> wrong one passed?

These are how your "bad luck" arose, apparently. Probably the
print statement caused the text to be stored right after the
space where the first pointer was located? Expect *no*
consistency from platform to platform, or IDL version to IDL
version!

> Any further comments?

I would recomment starting to use the "export.h" file that
defines the IDL_VARIABLE data type, and always accepting
parameters to external code by reference, not value.
This means you'll always get pointers to IDL_VARIABLE
structures, and you can do consistency checks on them etc. Sure,
it adds another "level" of access to get to the data, but
believe me, that's fine: More chances to *crash* things
instead of having spurious "Gosh, this worked, I must be
doing the right thing" experiences.

And then, with some experience of this, the jump isn't too big
(okay, it *is* scary!) to go on to Callable IDL (allows access
to IDL routines from within your external code, very neat!)
and LINKIMAGE, (or the Callable IDL equivalent IDL_AddSystemRoutine,
or even dynamically loadable modules -- hmmm, just discovered
those in the online docs thanks to your question!).

Regards,

Stein Vidar
(Hmh! - I should have chosen to do a PhD *on* IDL, not *with* IDL)

Subject: Re: CALL_EXTERNAL puzzle (still) ?
Posted by rmlongfield on Fri, 04 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Hi all,

In article <6smfdjiqk1@readme.uio.no>,
  steinhh@ulrik.uio.no (Stein Vidar Hagfors Haugan) wrote:
>
> rmlongfield@my-dejanews.com wrote:
>
>> print,'de-referenced pointer: ',*filename
>> result = CALL_EXTERNAL(VALUE=[1B],'example_c.so','example_c', filename)
>
> Whooa - here you're sending a *pointer* to a *string* variable
> into the call_external routine - and it's definitely *not* set
> up to receive it! In fact, your routine would *only* work if
> you sent it a scalar string that is passed by *value* only.
> (E.g. a string literal)

Thanks for your comments, but ... after some additional testing, I still maintain that something is not clear about CALL_EXTERNAL. Firstly, regardless of the unconventional setup of my program, it *worked*. What bothered me was that I didn't understand why (i.e. it worked for reasons that would not be clear if one reads the code).

When I first started working with the strings, I couldn't get the value passed. Now it seems to work, see result_b below :).

 However, why can't I pass a pointer? And if I want to pass a pointer, and print the value of another pointer just before the CALL_EXTERNAL, why is the wrong one passed? (See result_a). Ok, I'm neither a UNIX nor C expert (I'd like to say I'm an experienced user) and working with CALL_EXTERNAL was the first time I had to deal with argc and argv, and strings seemed to be a particular problem...Someone tells me that maybe the stdio and stdin are connected for CALL_EXTERNAL... I use PRINT statements all the time to guide my way through the operation of my code. I am also rather fond of the use of pointers. Any further comments?

Nevertheless, the filename, chosen with an IDL widget, is passed to my C programs, as was my intent. I've gotten rid of the pointer part.

Rose

```
;;;; IDL CODE ;;;;;;;;;;;;;; See previous post for C code;;;;;;;;;;;;
PRO example_idl
day1 = '961081302'
day2 = '999999999'

filename1=(PTR_NEW(STRTRIM(STRING(day1),2)))
filename2=(PTR_NEW(STRTRIM(STRING(day2),2)))

print,'de-referenced pointer: ',*filename1
result_a = CALL_EXTERNAL(VALUE=[1B],'example_c.so','example_c', filename2)
```

```
print,'de-referenced pointer: ',*filename1
;  This works !!!!!!!!!!!!!
result_b = CALL_EXTERNAL(VALUE=[1B],'example_c.so','example_c', day2)

end
```

Subject: Re: CALL_EXTERNAL puzzle (still) ?
Posted by steinhh on Sat, 05 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <EysGr6.ADL@midway.uchicago.edu>
rivers@cars3.uchicago.edu (Mark Rivers) writes:

> In article <6sofdhaef1@readme.uio.no>,
> steinhh@ulrik.uio.no (Stein Vidar Hagfors Haugan) writes:
>
>> I would recomment starting to use the "export.h" file that
>> defines the IDL_VARIABLE data type, and always accepting
>> parameters to external code by reference, not value.
>> This means you'll always get pointers to IDL_VARIABLE
>> structures,
>
> I don't think this is correct. If you pass parameters to CALL_EXTERNAL by
> reference you get a pointer to the value, not a pointer to the structure.
> The only exception is string parameters which are passed by descriptor.

Ooops! You're quite right. Bummer... It's only the LINKIMAGE'd
routines that receive pointers to IDL_VARIABLEs (and the ones
added through IDL_AddSystemRoutine()).

Stein Vidar
(Blushing)

Subject: Re: CALL_EXTERNAL puzzle (still) ?
Posted by rivers on Sat, 05 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <6sofdhaef1@readme.uio.no>, steinhh@ulrik.uio.no (Stein Vidar Hagfors Haugan)
writes:

> I would recomment starting to use the "export.h" file that
> defines the IDL_VARIABLE data type, and always accepting
> parameters to external code by reference, not value.
> This means you'll always get pointers to IDL_VARIABLE
> structures,

I don't think this is correct. If you pass parameters to CALL_EXTERNAL by reference you get a pointer to the value, not a pointer to the structure. The only exception is string parameters which are passed by descriptor.

Mark Rivers (773) 702-2279 (office)
CARS (773) 702-9951 (secretary)
Univ. of Chicago (773) 702-5454 (FAX)
5640 S. Ellis Ave. (708) 922-0499 (home)
Chicago, IL 60637 rivers@cars.uchicago.edu (e-mail)

or:
Argonne National Laboratory (630) 252-0422 (office)
Building 434A (630) 252-0405 (lab)
9700 South Cass Avenue (630) 252-1713 (beamline)
Argonne, IL 60439 (630) 252-0443 (FAX)