Subject: Re: Passing Structures with Pointers with Call_External
Posted by Peter Mason on Tue, 10 Aug 2004 23:12:27 GMT
View Forum Message <> Reply to Message

PeterOut wrote:
<...>
>
 temp={Rows:long(numrows),Columns:long(numcolumns),Data:fltar r(numrows,numcol
umns)}
<...>
> The C code is as follows.
> typedef struct FloatPlane_Struct
> {
>   long   Rows;
>   long   Columns;
>   float   **Data;
> } FloatPlane;
<...>
> If I add
> fprintf(stderr, "fppPlanes->Data[0]=%d\n", fppPlanes->Data[0]);
> idlde crashes, presumably due to a memory write error in the C code.
> Is there any way to stop idlde crashing under such circumstances?
>
> My main question is this. Is there a way to retrieve the IDL variable
> Planes[i].Data within CFunction_cw?

The problem here is that IDL isn't creating the structure quite as you
expect.   There isn't that level of indirection with DATA.   Your C-side
structure should look something like this:

```
    typedef struct FloatPlane_Struct {
      int      Rows;
      int      Columns;
      float   Data[n];
    } FloatPlane;
```

Where the "n" in "Data[n]" is equal to numrows*numcolumns in your IDL-side
structure creation statement.
I think this means that you need a different approach as a C-side structure
definition is fixed at compile time ("n" must be a constant).

You might be wondering about changing your structure definition to use an
IDL "pointer" for the array?   Don't even try it.   The value of an IDL
pointer is like some handle index thing and bears no relation to an actual
memory address.   It's meaningless to external code.

Personally, I'd suggest abandoning the use of a structure and coding a DLM
instead of CALL_EXTERNAL here.   CALL_EXTERNAL is quick and easy but
sometimes it's worth going that extra distance.   In a DLM you would be able

to pull out the dimensions of your DATA array (now 3-dimensional for the frames).   Also, the IDL-side work would probably be more efficient with a straightforward array instead of arrays embedded in structures. Alternatively, stick to CALL_EXTERNAL and pass your C function two parameters: DATA and SIZE(DATA).

Peter Mason

Subject: Re: Passing Structures with Pointers with Call_External
Posted by Bob[2] on Wed, 11 Aug 2004 12:40:57 GMT

View Forum Message <> Reply to Message

Peter, you might try changing the C-side to accept a struct of many Planes, then use call_external with a single argument (the name of the struct) and set /autoglue. E.g.,

```
function MakePlanes, numrows, numcolumns, sNumberOfPlanes
 ; put it together.  Must be named FloatPlane_Struct.
 return,a_planes_struct
end
```

On the C-side,

```
typedef struct FloatPlane_Struct
{
 /* as before
}

IDL_INT work( struct FloatPlane_Struct *a )
{
 ; Do your work with a->Rows[j] and so on
 ; then return to IDL.

 return,0;
}
```

And from the IDL-side again, e.g.

```
my_planes = MakePlanes(33,12,5)

work_result = call_external('SharedLibrary.so','work',my_planes, $
 /AUTO_GLUE,/I_VALUE)
```

Now, as I have set up my example, 'work' does computations on the structure 'my_planes' and maybe changes values in it etc., then returns an IDL_INT when finished. You can make the return type of 'work' to be

something else, like a struct, and instead of /I_VALUE you set
RETURN_TYPE=8.

Have fun working with it.

I am not sure why your other stuff is crashing idlde, though.

```
>  I declare an array of structures thus.
>  function MakeFloatPlane,numrows,numcolumns
>      temp={Rows:long(numrows),Columns:long(numcolumns),Data:fltar
r(numrows,numcolumns)}
>     return,temp
>  end
>
>  Planes=replicate(MakeFloatPlane(nY1,nX1),sNumberOfPlanes)
>
>  I then pass the array to C code through Call_External thus.
>   Result=Call_External('SharedLibrary.so','CFunction_cw',sNumb erOfPlanes,Planes,/unload)
>
>  The C code is as follows.
>  typedef struct FloatPlane_Struct
>  {
>         long    Rows;
>         long    Columns;
>         float   **Data;
>  } FloatPlane;
>
>  extern "C" long CFunction_cw(int argc, void *argv[])
>  {
>         long    lNumberOfPlanes;
>         FloatPlane *fppPlanes;
>         float   *fpData;
>
>         lNumberOfPlanes=*((long *)(argv[0]));
>         fppPlanes=((FloatPlane *)(argv[1]));


>  However, I cannot interpret the result I get for fppPlanes->Data.
>
>  If I add
>  fprintf(stderr, "fppPlanes->Data[0]=%d\n", fppPlanes->Data[0]);
>  idlde crashes, presumably due to a memory write error in the C code.
>  Is there any way to stop idlde crashing under such circumstances?
>
>  My main question is this.  Is there a way to retrieve the IDL variable
>  Planes[i].Data within CFunction_cw?
```

Subject: Re: Passing Structures with Pointers with Call_External
Posted by MajorSetback on Wed, 11 Aug 2004 13:53:13 GMT
View Forum Message <> Reply to Message

"Peter Mason" <drone@spam.com> wrote in message
news:<vDcSc.266$aA.11145@news.optus.net.au>...
```
> PeterOut wrote:
> <...>
>>
>  temp={Rows:long(numrows),Columns:long(numcolumns),Data:fltar r(numrows,numcol
> umns)}
> <...>
>> The C code is as follows.
>> typedef struct FloatPlane_Struct
>> {
>>        long    Rows;
>>        long    Columns;
>>        float   **Data;
>> } FloatPlane;
>  <...>
>> If I add
>> fprintf(stderr, "fppPlanes->Data[0]=%d\n", fppPlanes->Data[0]);
>> idlde crashes, presumably due to a memory write error in the C code.
>> Is there any way to stop idlde crashing under such circumstances?
>>
>> My main question is this.  Is there a way to retrieve the IDL variable
>> Planes[i].Data within CFunction_cw?
>
>
> The problem here is that IDL isn't creating the structure quite as you
> expect.   There isn't that level of indirection with DATA.   Your C-side
> structure should look something like this:
>       typedef struct FloatPlane_Struct {
>         int      Rows;
>         int      Columns;
>         float   Data[n];
>       } FloatPlane;
> Where the "n" in "Data[n]" is equal to numrows*numcolumns in your IDL-side
> structure creation statement.
> I think this means that you need a different approach as a C-side structure
> definition is fixed at compile time ("n" must be a constant).
>
> You might be wondering about changing your structure definition to use an
> IDL "pointer" for the array?   Don't even try it.   The value of an IDL
> pointer is like some handle index thing and bears no relation to an actual
> memory address.   It's meaningless to external code.
>
> Personally, I'd suggest abandoning the use of a structure and coding a DLM
> instead of CALL_EXTERNAL here.   CALL_EXTERNAL is quick and easy but
```

> sometimes it's worth going that extra distance. In a DLM you would be able
> to pull out the dimensions of your DATA array (now 3-dimensional for the
> frames). Also, the IDL-side work would probably be more efficient with a
> straightforward array instead of arrays embedded in structures.
> Alternatively, stick to CALL_EXTERNAL and pass your C function two
> parameters: DATA and SIZE(DATA).
>
> Peter Mason

Hi Peter,

Thanks very much for the reply. I decided to do something along the
lines of the last thing you mentioned. I made a 3D volume with IDL
thus.
Volume=fltarr(nPlanes,nRows,nCols)
I then filled the volume with the plane data and pass it thus.
 Result=Call_External('SharedLibrary.so','CFunction_cw',NumPl
anes,NumRows,NumCols,Volume,/unload)

In the C function, I then have a float pointer that reads the array
thus.
float   *fpData=((float *)(argv[3]));

Hence I have a pointer to the data, in 1D, that can be upwrapped using
the volumetric dimensions.

Thanks again for your help,
Peter.

Subject: Re: Passing Structures with Pointers with Call_External
Posted by MajorSetback on Wed, 11 Aug 2004 13:53:25 GMT

View Forum Message <> Reply to Message

"Peter Mason" <drone@spam.com> wrote in message
news:<vDcSc.266$aA.11145@news.optus.net.au>...
> PeterOut wrote:
> <...>
>>
> temp={Rows:long(numrows),Columns:long(numcolumns),Data:fltar r(numrows,numcol
> umns)}
> <...>
>> The C code is as follows.
>> typedef struct FloatPlane_Struct
>> {
>>   long   Rows;
>>   long   Columns;
>>   float   **Data;

```
>>  } FloatPlane;
>   <...>
>>  If I add
>>  fprintf(stderr, "fppPlanes->Data[0]=%d\n", fppPlanes->Data[0]);
>>  idlde crashes, presumably due to a memory write error in the C code.
>>  Is there any way to stop idlde crashing under such circumstances?
>>
>>  My main question is this.  Is there a way to retrieve the IDL variable
>>  Planes[i].Data within CFunction_cw?
>
>
>  The problem here is that IDL isn't creating the structure quite as you
>  expect.   There isn't that level of indirection with DATA.   Your C-side
>  structure should look something like this:
>         typedef struct FloatPlane_Struct {
>           int      Rows;
>           int      Columns;
>           float   Data[n];
>         } FloatPlane;
>  Where the "n" in "Data[n]" is equal to numrows*numcolumns in your IDL-side
>  structure creation statement.
>  I think this means that you need a different approach as a C-side structure
>  definition is fixed at compile time ("n" must be a constant).
>
>  You might be wondering about changing your structure definition to use an
>  IDL "pointer" for the array?   Don't even try it.   The value of an IDL
>  pointer is like some handle index thing and bears no relation to an actual
>  memory address.   It's meaningless to external code.
>
>  Personally, I'd suggest abandoning the use of a structure and coding a DLM
>  instead of CALL_EXTERNAL here.   CALL_EXTERNAL is quick and easy but
>  sometimes it's worth going that extra distance.   In a DLM you would be able
>  to pull out the dimensions of your DATA array (now 3-dimensional for the
>  frames).   Also, the IDL-side work would probably be more efficient with a
>  straightforward array instead of arrays embedded in structures.
>  Alternatively, stick to CALL_EXTERNAL and pass your C function two
>  parameters:  DATA and SIZE(DATA).
>
>  Peter Mason
```

Hi Peter,

Thanks very much for the reply. I decided to do something along the
lines of the last thing you mentioned. I made a 3D volume with IDL
thus.

```
Volume=fltarr(nPlanes,nRows,nCols)
```

I then filled the volume with the plane data and pass it thus.

```
 Result=Call_External('SharedLibrary.so','CFunction_cw',NumPl
```

anes,NumRows,NumCols,Volume,/unload)

In the C function, I then have a float pointer that reads the array
thus.
float   *fpData=((float *)(argv[3]));

Hence I have a pointer to the data, in 1D, that can be upwrapped using
the volumetric dimensions.

Thanks again for your help,
Peter.

Subject: Re: Passing Structures with Pointers with Call_External
Posted by MajorSetback on Wed, 11 Aug 2004 14:34:11 GMT
View Forum Message <> Reply to Message

"Peter Mason" <drone@spam.com> wrote in message
news:<vDcSc.266$aA.11145@news.optus.net.au>...

```
>  The problem here is that IDL isn't creating the structure quite as you
>  expect.   There isn't that level of indirection with DATA.   Your C-side
>  structure should look something like this:
>         typedef struct FloatPlane_Struct {
>           int      Rows;
>           int      Columns;
>           float    Data[n];
>         } FloatPlane;
>  Where the "n" in "Data[n]" is equal to numrows*numcolumns in your IDL-side
>  structure creation statement.
>  I think this means that you need a different approach as a C-side structure
>  definition is fixed at compile time ("n" must be a constant).
>
```

Just a follow up question I thought of. Is it possible to return a C
structure with pointers to IDL and have IDL interpret it correctly?

Thanks again,
Peter.

Subject: Re: Passing Structures with Pointers with Call_External
Posted by Peter Mason on Wed, 11 Aug 2004 23:44:17 GMT
View Forum Message <> Reply to Message

PeterOut wrote:
...
> Just a follow up question I thought of. Is it possible to return a C

> structure with pointers to IDL and have IDL interpret it correctly?

No, I can't see that working.   An array in an IDL structure is "right there" in the structure's data block; it does not hang off a memory-pointer.   (...Except for a string scalar which might be considered to be an array of sorts but is really a special kind of thing in IDL - it hangs off a descriptor.)   If you're thinking "IDL pointer" rather than memory-pointer then again - you can't.   For some reason, RSI have not exposed any functions to work with IDL pointers in external code, and digging around in exports.h doesn't reveal anything interesting.   (Note I haven't got IDL6.1 yet so I don't know if this is still the case.)

I sense a certain regret in abandoning the structure approach?   :-)
I imagine that you have ancillary frame information that you'd like to keep together with the primary data, or something like that?
If you really would like to keep all this info in a single IDL structure array then I think that you could work with it in a C routine if you don't mind getting your hands a bit dirty.   What I'm thinking here is accessing the structure data in a raw sense.   Your C CALL_EXTERNAL routine gets passed a memory pointer to the structure array's data block and can interpret it any way that it likes.   In your particular example, you could just cast it to an int pointer *IP.   You'd have to track which frame you were at.   Given that the DATA array has to be the same size in each structure element, you could calculate a frame-size as IFS=IP[0]*IP[1] + 2.
To get at the DATA of frame I, use a float pointer *FDATA and set it to FDATA=(float *)(I*IFS + 2).
If you go with this approach you will have to take great care in how you define the structure in IDL.   Your example from the other day just had LONGs and FLOATs so all elements were 4 bytes long and the structure had no padding bytes.   If the structure got more complicated then things could get a lot messier.

Cheers
Peter Mason