Subject: Re: C Alignment/IDL structures
Posted by Randall Skelton on Mon, 21 Mar 2005 23:03:27 GMT
View Forum Message <> Reply to Message

> Ok, this probably answers my question. I was hoping I could create
> an array of structs, but this is maybe not so memory efficient so
> I might try to create one structure with multiple arrays.

Good news Joey, you can create an array of structs in C and pass them back to IDL so you are not trying to do the impossible. I'm not entirely sure what is going wrong in the code you posted. Are you seg faulting IDL, getting garbage in the resulting array of structures or getting an undefined variable returned?

Download the code at :

http://brutus.uwaterloo.ca/~skelton/Code/TestStruct.tar.gz

The code assumes a *nix operating system but modifying it to work with windows shouldn't be too difficult. The code gives two methods of creating an array of structures in C and pass them to back to IDL. The first is quite similar to what you are trying to do (uses IDL_ImportArray) and the second handles the case where the C structure doesn't cleanly map to the new IDL structure (slightly more complicated and requires some pointer arithmetic).

Let me know if it helps.

Cheers,
Randall

Subject: Re: C Alignment/IDL structures
Posted by Nigel Wade on Tue, 22 Mar 2005 10:34:18 GMT
View Forum Message <> Reply to Message

joey@swri.edu wrote:

> Randall Skelton <randall.skelton@gmail.com> wrote:
>> I'm not sure I completely understand what you are doing... can you post
>> a snipit of code? I trust that you are passing back an unnamed
>
> Here's my code that does the IDL/C interaction. I have a vector
(_dataIDL)
> which has all the values I want into IDL within it as a malloc'ed sets of
> memory.
>
> // Copy the real data

```
>
>      unsigned long pos = 0;
>      unsigned char *myStructureThatLooksLikeTags = malloc
(_totalSpaceNeeded
>                                                 * _dataIDL.size
());
>      for (unsigned int i = 0; i < _dataIDL.size (); i++) {
>   memcpy (&(myStructureThatLooksLikeTags [pos]), _dataIDL [i],
>                 _totalSpaceNeeded);
>   pos += _totalSpaceNeeded;
>      }
>
>      char idl_struct_name [100];
>      sprintf (idl_struct_name, "%s_K%ld_V%d", dbVirtualName (data_key),
>                 data_key, version);
>
>      void *idl_struct = IDL_MakeStruct (idl_struct_name, tags);
>      IDL_LONG dims;
>      dims = _dataIDL.size (); // number of elment in the array of
structures
>      IDL_VPTR ivReturn = IDL_ImportArray (1, &dims, IDL_TYP_STRUCT,
>                                    myStructureThatLooksLikeTags,
>                                    cleanUpIDL, idl_struct);
>
>>  In general, when I have to pass C structures from existing code back to
>>  IDL I do it by creating a shadow structure (in C) that uses all the
>>  defined IDL types and copying the data.  You really cannot rely on
>>  generic C variables having the same size as thier IDL counterparts
>>  (take a look at the definition of IDL_ALLTYPES in idl_export.h).
>
>  Ok, this probably answers my question. I was hoping I could create an
array
>  of structs, but this is maybe not so memory efficient so I might try to
>  create one structure with multiple arrays.
>
>  Joey
```

It is possible to do what you want to do. But you have to realize that there are some caveats, and they are pretty serious ones.

The padding within a C structure is not defined by the language. It is an implementation issue, and is entirely at the discretion of the compiler developer. There is no guarantee that the padding inserted by one C compiler will be the same as the padding inserted by another compiler. You can't even guarantee that the next minor release of your compiler will use the same padding as the current release. It's a pretty safe bet, but not guaranteed.

In your code you copy memory from two disimilar data structures using memcpy() taking no account of alignment. Your memory allocation uses the construct _dataIDL.size(), what is that? Since you haven't shown us the definition of the storage for _dataIDL or your IDL tags, then it's impossible to say what's wrong. But my guess would be your data types or alignment in _dataIDL doesn't match what you've specified in your IDL structure tags.

It is possible to generate arrays of structures in C and pass them back to IDL, but you have to be very careful and you have to undertand its limitations.

--
Nigel Wade, System Administrator, Space Plasma Physics Group,
University of Leicester, Leicester, LE1 7RH, UK
E-mail : nmw@ion.le.ac.uk
Phone : +44 (0)116 2523548, Fax : +44 (0)116 2523555

Subject: Re: C Alignment/IDL structures
Posted by joey on Wed, 23 Mar 2005 16:20:11 GMT
View Forum Message <> Reply to Message

Nigel Wade <nmw@ion.le.ac.uk> wrote:
>>
>> // Copy the real data
>>
>> unsigned long pos = 0;
>> unsigned char *myStructureThatLooksLikeTags = malloc
> (_totalSpaceNeeded
>> * _dataIDL.size
> ());
>> for (unsigned int i = 0; i < _dataIDL.size (); i++) {
>> memcpy (&(myStructureThatLooksLikeTags [pos]), _dataIDL [i],
>> _totalSpaceNeeded);
>> pos += _totalSpaceNeeded;
>> }
>>

> In your code you copy memory from two disimilar data structures using
> memcpy() taking no account of alignment. Your memory allocation uses the
> construct _dataIDL.size(), what is that? Since you haven't shown us the

_dataIDL is actually a C++ vector of unsigned char *'s. This group of char * is created from a C++ map of data. The map is a tagname and data value. Complex, isn't it? I'd like to think its elegant since I can very simply wrap any C/C++ and interface it with IDL. However, if one of the

items in the map is a structure itself, I think it has problem.

Randall: I got your code and I am going to look at it. I appreciate the example! I am going to try throwing a structure with a double in there and see if I can generate the same problem I am seeing on my code.

Cheers,
Joey

Subject: Re: C Alignment/IDL structures
Posted by Nigel Wade on Thu, 31 Mar 2005 10:46:57 GMT
View Forum Message <> Reply to Message

joey@swri.edu wrote:

> Nigel Wade <nmw@ion.le.ac.uk> wrote:
>>>
>>> // Copy the real data
>>>
>>> unsigned long pos = 0;
>>> unsigned char *myStructureThatLooksLikeTags = malloc
>> (_totalSpaceNeeded
>>> *
_dataIDL.size
>> ());
>>> for (unsigned int i = 0; i < _dataIDL.size (); i++) {
>>> memcpy (&(myStructureThatLooksLikeTags [pos]), _dataIDL [i],
>>> _totalSpaceNeeded);
>>> pos += _totalSpaceNeeded;
>>> }
>>>
>
>> In your code you copy memory from two disimilar data structures using
>> memcpy() taking no account of alignment. Your memory allocation uses the
>> construct _dataIDL.size(), what is that? Since you haven't shown us the
>
> _dataIDL is actually a C++ vector of unsigned char *'s. This group of
> char * is created from a C++ map of data. The map is a tagname and data
> value. Complex, isn't it? I'd like to think its elegant since I can very
> simply wrap any C/C++ and interface it with IDL. However, if one of the
> items in the map is a structure itself, I think it has problem.

Perhaps it's overly complicated?

You have an array of tags called tags which you pass into IDL_MakeStruct. These are the tag names which will be set in the structure it creates. Why does the data structure _dataIDL also have tag names in it? I'm confused.

Are you trying to create a structure which contains a structure?. If so,
what I do is set the tag type in the IDL_STRUCT_TAG_DEF array for the
parent structure to IDL_TYP_STRUCT. For the child structure I use
IDL_MakeStruct to create it's IDL structure, and then set the parent
structure type for that tag to be this value.

E.g. this creates a structure containing a structure. The display_tags
structure contains a structure with protocol_tags.

```
IDL_STRUCT_TAG_DEF protocol_tags[] = {
  { "VERSION", 0, (void *)IDL_TYP_LONG },
  { "REVISION", 0, (void *)IDL_TYP_LONG },
  { 0 },
};

IDL_STRUCT_TAG_DEF display_tags[] = {
    { "NAME", 0, (void *)IDL_TYP_STRING },
    { "VENDOR", 0, (void *)IDL_TYP_STRING },
    { "PROTOCOL", 0, (void *)IDL_TYP_STRUCT },
                /* set this to protocol_s later */
    { 0 },
};
void *protocol_s, *display_s;

protocol_s = IDL_MakeStruct(0, protocol_tags);
display_tags[2].type = protocol_s;

display_s = IDL_MakeStruct(0, display_tags);
```

IDL_TYP_STRUCT in the tags array is really just a place-holder and comment,
it doesn't do anything as it's replaced before the tags array is used to
create the parent structure.

--
Nigel Wade, System Administrator, Space Plasma Physics Group,
University of Leicester, Leicester, LE1 7RH, UK
E-mail : nmw@ion.le.ac.uk
Phone : +44 (0)116 2523548, Fax : +44 (0)116 2523555