Subject: Re: how does /no_copy work???
Posted by steinhh on Fri, 04 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <MPG.11c106a5c342a5ab9897d0@news.frii.com>
davidf@dfanning.com (David Fanning) writes:
> John Persing (persing@frii.com) writes:
>
>> But let me ask, how can this be possible when deal with a variable that
>> "starts" on the stack and "ends up" on the heap? If B is an ordinary array
>> and A is property of an object, then this is what will occur. The heap and
>> stack are entirely different memory locations.
>
> I'm rapidly getting out of my depth here, but it seems to me that
> the *object* itself is on the heap, but that the actual data that
> fields in the object point to can be anywhere in process memory.
> All that has to be stored in the object field is a pointer
> (a *real* pointer, not an IDL pointer) to the real data. This
> is what is passed, isn't it, when a variable is passed by
> reference? If that wasn't the case, how else could a variable
> be stored in a widget user value with NO_COPY, which to my
> mind is equivalent to the heap (I.e, a global memory location)?
>
> And keep in mind that "stack" and "heap" have meanings in IDL
> that *may* not correspond to what you usually think about when you
> use these terms.
>
> Whew, I can't feel the bottom any more! :-(

I think you're OK, David - just don't try to breathe while your
head is below water.. :-)

Let me se if I can add anything to this.

An IDL variable (within the current scope) or expression is always
associated (*) with a block of data called an IDL_VARIABLE
structure. Even if it's undefined - in fact, "undefined" is a data
type in IDL...

For all *scalar* *numeric* data types, the value is stored
within that structure. For strings & arrays, the data itself is
stored another place - in some part of some heap memory - and
the IDL_VARIABLE contains a (true) pointer to the data.

"Passing parameters by reference" means that the parameters are
sent to subroutines by means of (true) pointers to IDL_VARIABLE
data blocks representing the parameters. Thus in fact *all*
parameters are passed by reference (none are passed by value!).

It's just that an IDL_VARIABLE structure that represents
"expressions" do not correspond to a named variable, and the
IDL_VARIABLE structure has a flag set to indicate this fact.

For normal variables & expressions (inside functions), I guess the
IDL_VARIABLE structures are allocated as slots in some "variable
stack" (and not necessarily the processor stack, as David points
out). These slots are deallocated when a subroutine returns.

So what's up with pointers & objects? Well, such beasts are IDL
variables like all the others, so if "my_ptr" is a pointer, it's
associated with an IDL_VARIABLE slot on the variable stack, and
you would look up that slot (given the variable name) just like
for all other variables.

But the IDL_VARIABLE associated with "my_ptr" doesn't contain the
value of "*my_ptr", it contains a "magic number".

The magic number is like a variable name in some *global*
scope. Internally, IDL can use the magic number to find the
location of an IDL_VARIABLE structure that represents this global
variable. This structure does *not* reside on the variable stack,
so when a subroutine returns, it's not deallocated.

Everyone who knows the magic number can look up the IDL_VARIABLE
structure associated with it. You can share the magic number by
making copies of the IDL_VARIABLE structure containing the magic
number (the "value" of "my_ptr"), and the data can be shared
between different scopes.

I guess I should leave it to the reader as an exercise to figure
out what the difference between a null pointer and a pointer
to an undefined variable is... :-)

Regards,

Stein Vidar

(*) At least after you've attempted to look up that variable..