Subject: Re: HANDLE_FREE: when to use? is it necessary?
Posted by steinhh on Fri, 26 Jul 1996 07:00:00 GMT
View Forum Message <> Reply to Message

In article <Pine.SOL.3.91.960722114953.671B-100000@mapper.mbt.washington.edu>, Russ Welti <rwelti@mapper.mbt.washington.edu> writes:
|>
|> I was told once by RSI that if one is only about to reassign
|> a handle to point to a NEW piece of data (when it already
|> points to some other data) it is *not* necessary to call
|> HANDLE_FREE.
|>
|> Also, interestingly, using grep in idl's home /lib dir and in
|> several pub domain dirs which have hundreds of IDL source files
|> and applications, I can find only 3 or 4 occurrences of 'handle_free'
|> OR 'HANDLE_FREE'. Why does it not get used? Why does the manual
|> never say or show how to use it? Is it really so unimportant?
|>

I find 8 lines with handle_free (case insensitive search), 4 lines with handle_create, 23 occurrences of handle_value, 15 handle_info lines and 1 line with handle_move. Pretty good programming practice by RSI staff, if you ask me. This is for IDL 3.6.1c.

When I started using handles for my own purposes, I did a (too?) quick check to se what effects handle_free could have. As it turns out, the handle *numbers* appeared to be always increasing regardless of handle_free calls, so I figured that RSI just hadn't gotten around to actually freeing any handles. Still, in my own programs, I insist on using handle_free whenever an "object" using a handle is "decommissioned", simply because it's good programming practice, and because I expected RSI to get it right in some future version.

When reading your questions, I did a further check like this:

Start a fresh IDL session, then run the following:

```
PRO handle_test1,N

  time = systime(1)

  large_array = fltarr(10000)
  null = 0
  time = systime(1)

  FOR i = 0L,N DO BEGIN
    handle = handle_create()
    ;; Leaves large_array undefined
```

```
    handle_value,handle,large_array,/set,/no_copy

    ;; Get back large_array, make *very* sure no optimization skips
    ;; everything
    handle_value,handle+null,large_array,/no_copy

    handle_free,handle
  END

  PRINT,systime(1)-time
END
```

No matter how many times you run this, the timing is quite impressive,
and it stays constant at about 5 seconds for N=100000 on my machine:

```
** Structure !VERSION, 3 tags, length=48:
   ARCH            STRING    'alpha'
   OS              STRING    'OSF'
   RELEASE         STRING    '3.6.1c'
```

Now, comment out the line with handle_free, and run it again.
Try with N=1000 first, you're going to be in for a surprise.
Each time you run the program, the time goes up and up and up.
If you plot the timings, they make an almost straigh line.
Then, run the original version (with handle_free in place).
This will have slowed down considerably, depending on how
many times you executed the non-handle-freeing version, but
now the times will stabilize.

Conclusion: If you're disposing with a handle, then *USE FREE_HANDLE*,
or else you'll be slowing down every application that uses
handles, even if *they* are sticking to good programming practice.

Now, it should be okay to reuse handles without freeing them, as
far as this is done "locally", i.e., you reuse the handle just
about straight away, like in the same piece of code that could
have opted for the use of free_handle.

I guess the problem here is that IDL doesn't keep a list of "free"
handles, but goes through it's allocated handles sequentially to see if
they're free or not. This should be fixed, RSI!! (It might have
been corrected in IDL 4 already, though).

|> I ask because my app is getting memory hungry, and its memory
|> consumption currently monotonically grows over the course
|> of an extended session of using it, even though when user
|> 'opens' a new 'doc', I set all large arrays to scalar 0.
|>

As someone else mentioned, do you use the /no_copy switch?
It *really* speeds up your execution, avoiding large memory
copying operations, and *could* help with the memory growth
isue. You have to be aware of the fact that it
makes the original "source" variable go undefined, though, and
at times you may loose data in crashes (they are recoverable,
though, just go through handles 1,....,handle_create()-1 to see
if they're valid, use handle_info, and try to get at their values).

Other than that, I really think that memory fragmentation is
more to blame than a pileup of handle memory space. Some
OS versions never really free memory once it's allocated to a
given process.

|> If freeing handles is really the answer, and is a required practise,
|> then why so few occurrences of its use in IDL programs?
|>

It's used in more places than handle_create, so the newness
of the whole concept is the cause.

|> Anyone ever gotten a good FAQ on IDL memory management, beyond the
|> few pointers given in the User's Guide, and what can be gleaned
|> from the C programming info in the Advanced Dev. Guide??
|>

There should be one, but I haven't heard of one.

I just pray that RSI will respond to my cry for some decent handle
notation changes, that will make a three-statement thing like

```
  handle_value,handle,val,/no_copy
  result = val * 5.2
  handle_value,handle,val,/set,/no_copy
```

into the following (they can use whatever "special character" or
notation they like, as long as it fits in a single statement).

```
  result = @handle * 5.2 ;;
```

Why they haven't made this available in the first place? I don't
know, but I guess it's attributable to most of them having grown
up with f77. IDL used to be written in Fortran, but now they've
switched to C, from what I hear.

Stein Vidar Haugan
