Subject: Re: Problems with IDL call_external to C shared object
Posted by Mark Rivers on Thu, 12 Oct 2000 07:00:00 GMT
View Forum Message <> Reply to Message

I've done a lot of this, and it's not that hard, don't give up!

Here are some tricks:

- All arrays which need to be passed between IDL and C must be allocated in IDL, as J.D. Smith said. This includes both arrays being passed from IDL to C and from C back to IDL. Sometimes this requires an initial call to the C code to return the array sizes which IDL will allocate, if the array sizes are not known to IDL beforehand.

- Don't deallocate any arrays which were passed from IDL.

- Don't pass strings, rather pass byte arrays. It is much simpler. Convert strings to byte arrays in IDL before or after the CALL_EXTERNAL call.

- Convert all output variables to the data type which C is expecting in the CALL_EXTERNAL call.

> - What is the effect of the /CDECL keyword to CALL_EXTERNAL ?
> I tried with and without but no success.

This controls the calling convention. If your C function is being called then you probably have this set correctly.

> - Is it possible that the C program "forgets" something between
> the IDL CALL_EXTERNALs ?

- As J.D. Smith said, it will forget anything which is not global or static.

> - How can I return an array via CALL_EXTERNAL or have I always
> to loop over calls returning scalars ? The EZCA library (channel
> access to EPICS control system) manages to return arrays, but I
> couldn't figure out how.

My EZCA code is rather opaque, since it uses macros which allow it to work on both IDL and PV-WAVE, on Unix, VMS and Windows.

Here is a simple example. It is C code which computes the Mandelbrot set, and is called from IDL.
argv[7] is a 2-D array.

```
void mandelbrot(int argc, void *argv[])
{
 int nr = *(int *) argv[0];
 int ni = *(int *) argv[1];
 double rstart = *(double *) argv[2];
 double istart = *(double *) argv[3];
 double dr = *(double *) argv[4];
 double di = *(double *) argv[5];
 int max_iter = *(int *) argv[6];
 int *result =  argv[7];
int i, j, count;
 double real, imag, rz, iz, sz2, rz2, iz2;
    for (i=0; i<ni; i++) {
      imag = istart + i*di;
      for (j=0; j<nr; j++) {
           real = rstart + j*dr;
           rz = 0.;
           iz = 0.;
           sz2 = 0.;

           count = 0;
           while ((count < max_iter) && (sz2 < 4.0)) {
                rz2 = rz * rz;
                iz2 = iz * iz;
                iz = 2.0 * rz * iz + imag;
                rz = rz2 - iz2 + real;
                sz2 = rz2 + iz2;
                count++;
           }
           *result++ = count;
      }
    }
}
```

Here is the IDL code which calls the C code:

```
function mandelbrot1, xcenter, ycenter, radius, size, max_iter, xout, yout
if (n_elements(size) eq 0) then size=100
if (n_elements(max_iter) eq 0) then max_iter=255
dx = double(radius)*2/size
xstart = double(xcenter - radius)
xstop = double(xcenter + radius)
ystart = double(ycenter - radius)
ystop = double(ycenter + radius)
result = lonarr(size, size)
xout = xstart + findgen(size)*dx
yout = ystart + findgen(size)*dx
s = call_external('mandelbrot.dll', 'mandelbrot', $
                   long(size), $
                   long(size), $
```

```
              double(xstart), $
              double(ystart), $
              double(dx), $
              double(dx), $
              long(max_iter), $
              result)
return, result
end
```

Subject: Re: Problems with IDL call_external to C shared object
Posted by John-David T. Smith on Thu, 12 Oct 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Streun Andreas wrote:
>
> Hello -
>
> who is experienced in running IDL with C shared objects?
>
> I'm trying to make an IDL GUI for a rather complex C batch
> program. The effects are rather strange: sometimes it works
> perfectly, but mostly it doesn't: suddenly on the C-side strange
> and wrong number appear in calculations leading to crashes. It seems
> like something is initialized or dereferenced in a wrong way.
> However the behaviour is determinsitic: a small change in the IDL
> program like declaring a new variable anywhere causes the crash,
> after undoing the change it works well again. Maybe a memory conflict ?
>
> The C-program alone in batch mode runs reliably. It does a lot of
> mallocs but never frees any memory (because it is batch).
> IDL communicates via the CALL_EXTERNAL function.
> I'm rather sure that I have checked the variables on both sides of the
> fence are really of same type. (However I'm a poor C-programmer...)
> I'm using IDL 5.3 on a Linux system and the GNU C-compiler.
>
> Now the questions:
>
> - Is it possible that IDL overwrites or frees memory allocated by the C
> shared object ? Is there a general way to prevent it from doing so ?
>
> - What is the effect of the /CDECL keyword to CALL_EXTERNAL ?
> I tried with and without but no success.
>
> - Is it possible that the C program "forgets" something between
> the IDL CALL_EXTERNALs ?
>
> (important:)

> - Is there an opinion whether this problem can be solved in principle
> and within finite time ?!
>
> (has nothing to do with the problem but I would like to know:)
> - How can I return an array via CALL_EXTERNAL or have I always
> to loop over calls returning scalars ? The EZCA library (channel
> access to EPICS control system) manages to return arrays, but I
> couldn't figure out how.
>
> Thanks for any help.

The best way to use call_external I've found is to allocate all arrays, variables, and strings on the IDL side and directly manipulate them within the C program. Most variable types do *not* map directly between IDL and C. Did you take a good look at $IDL_DIR/external/call_external/C/, which contains lots of (small) examples? Also see the "external.h" header for lots of info.

Another bit of confusion: IDL simply calls the function directly from the shared library specified... the function is not at all linked in (other than existing in a shared program stack), and variables will not be preserved through successive function calls (unless they are declared static or global).

An example of passing an array as a variable:
```
IDL_LONG showarray(int argc, void *argv[]) {
  float *arr;
  IDL_MEMINT *n_elem,i;
  arr=(float *) argv[0];
  n_elem=(IDL_MEMINT *) argv[1];
  printf("%d\n",*n_elem);
  for(i=0;i<*n_elem;i++)
 printf("%d: %f\n",i,arr[i]); /* Don't printf, it's not nice! */
  return 1;
}
```

which would be called via, e.g.:

```
IDL> ret=call_external('mylib.so','showarray',findgen(10),10)
```

Presumably if your code allocates it's own memory without cleaning up after itself, it will be rather unstable. Unless you need to return arrays of dynamic size/type, the originate-all-data-in-IDL method will much simplify your life.

Good luck,

JD

```
--
 J.D. Smith                  |   WORK: (607) 255-6263
 Cornell Dept. of Astronomy  |         (607) 255-5842
 304 Space Sciences Bldg.    |    FAX: (607) 255-5875
 Ithaca, NY 14853            |
```

Subject: Re: Problems with IDL call_external to C shared object
Posted by Nigel Wade on Fri, 13 Oct 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Mark Rivers wrote:
>
> I've done a lot of this, and it's not that hard, don't give up!
>
> Here are some tricks:
>
> - All arrays which need to be passed between IDL and C must be allocated in
> IDL, as J.D. Smith said. This includes both arrays being passed from IDL to
> C and from C back to IDL. Sometimes this requires an initial call to the C
> code to return the array sizes which IDL will allocate, if the array sizes
> are not known to IDL beforehand.
>

Alternatively, you can use the LINKIMAGE or DLM interface where the C
code
can create any IDL variable required. Albeit at the expense of a greater
learning curve.

```
--
-----------------------------------------------------------
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 2523568, Fax : +44 (0)116 2523555
```