
Subject: Re: Problem Accessing Shared Objects
Posted by [Allan Whiteford](#) on Thu, 10 Jan 2008 10:38:48 GMT
[View Forum Message](#) <> [Reply to Message](#)

David Fanning wrote:

> Folks,
>
> I had an occasion today to access a shared object library
> on a UNIX machine. This worked correctly in IDL 6.3, but
> completely crashed IDL 7 when I tried to access it today.
>
> I've looked on my web page, and in the IDL newsgroup
> archives for information, but have completely struck out.
> And yet, I vaguely recall something about some kind of
> header file in IDL changing in the transition from IDL 6.3
> to 6.4, and need for re-compiling, etc. It's just not
> coming in clearly.
>
> We have tried re-compiling the SO, but we haven't yet
> been successful in not crashing IDL. Am I dreaming, or
> does someone else remember something about this?
>
> Cheers,
>
> David
>

David,

I can run the same shared object file (albeit not the same one as you)
in IDL 6.2 and IDL 7.0.

Glancing over recent changes to export.h, there isn't anything which
glaringly stands out but it does, of course, all depend on exactly what
you're doing. My suspicion would be there there is an error in your
shared object and you were just lucky before.

If it is because the shared object needs recompiled (I don't think it is
though) then make sure that you're using the idl_export.h file from
IDL7, check Makefiles etc.

You can usually find out where your shared object is crashing by typing
magic UNIX incantations...

If you have some C which looks like this:

```
#include <stdlib.h>  
#include <stdio.h>
```

```
#include "idl_export.h"
```

```
IDL_VPTR testing(int argc, IDL_VPTR argv[])  
{  
  int a[1];  
    a[10000000]=1;  
  return 0;  
}
```

which should segfault then you normally compile it something like this:

```
gcc -I/usr/local/rsi/idl/external/include -shared \  
testing.c -o testing.so
```

(probably with itervis instead of rsi)

and then run it inside IDL (command line version) like this:

```
IDL> junk=call_external('testing.so','testing')
```

or like this:

```
IDL> linkimage,'testing','testing.so'  
IDL> testing
```

and it gives you the helpful message of:

Segmentation fault

you can get the message to be slightly more helpful. First, compile the code using "-g" thus:

```
gcc -g -I/usr/local/rsi/idl/external/include -shared \  
testing.c -o testing.so
```

you can now choose to get yourself a core file or to run IDL inside a debugger.

To get a core file type something like this:

```
ulimit -c unlimited (if using bash)  
limit coredumpsize unlimited (if using csh)
```

before running IDL.

Now, your error message should say this:

Segmentation fault (core dumped)

and a file (probably quite large) should be sitting there called core.?????. (????? was the PID of the thing which dumped it).

You can now type something like:

```
gdb /usr/local/rsi/idl/bin/bin.linux.x86/idl core.?????
```

and get output which looks like this:

```
... lots of nonsense ....  
#0 testing (argc=0, argv=0xbfe65c20) at testing.c:17  
17      a[10000000]=1;
```

telling me that the problem occurred on line 17 of testing.c. You are now inside gdb, the two commands you probably want now are either "bt" or "quit" - I'd recommend "quit" :).

You can also run IDL inside of the debugger so it crashes in real time without having to produce a core - this involves messing around with LD_LIBRARY_PATH and has similar woes as when people mess around with their IDL_PATH. I prefer this latter approach but it's more problematic to setup.

I have no idea what the procedure for the above would be inside the IDL workbench but I hope the above is helpful anyway.

Thanks,

Allan
