
Subject: Re: Call_external and link libraries - SunOS

Posted by [afl](#) on Wed, 19 Apr 1995 07:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

In article <1995Apr19.092214.22596@rahman.earth.ox.ac.uk>, keith@earth.ox.ac.uk (Keith Refson) writes:

|> I wonder if anyone can offer any suggestions on the following problem. I
|> am attempting to use CALL_EXTERNAL to interface to a FORTRAN
|> subroutine, using IDL 3.5 and Sun Fortran 1.4 running under SunOS
|> 4.1.3_U1. I have a C wrapper to pass the arguments to the FORTRAN.
|>
|> The problem is that the FORTRAN code makes calls to the math library
|> - simply the cosine function _Fcos. However when I build the
|> shared-object file, viz
|>
|> % cc -c -pic idl-link.c
|> % f77 -pic -c -O approx.f
|> % ld -o approx.so -assert pure-text *.o /usr/lang/SC1.0/libF77.so.1.4.1
|>
|> and call "approx.so", IDL crashes because the binary makes an
|> unsatisfied reference to _Fcos.
|>
|> Allright, _Fcos is in the math library /usr/lang/SC1.0/libm.a so I
|> should link *that* into assert.so just like I did with the FORTRAN
|> support library above. BUT in SunOS, there is only a static version
|> of this library! (ie no libm.so).

Keith,

Maybe you can explain this to me someday, but what I have done (successfully under SOLARIS) is to link the static libraries into my shared object! Huh? Yeah, that's what I thought (and still do), but it works, and that is the name of the game. If you figure out how/why, please let me know. Here is an example I posted awhile back.

The first program is an IDL procedure. Note the compilation and link statements that are executed using spawn. Note also that I am using Solaris, and my libraries are in /usr/lib.

The second program is a simple FORTRAN program which uses numerous mathematical intrinsic functions.

Good luck and let me know how it goes!

Andy Loughe

```

;=====
; Originator: Andrew F. Loughe
;
; *** SUN SOLARIS TEST ***
; Procedure to call a FORTRAN program from within IDL.
; The first 100 primes are computed in the FORTRAN program and
; passed into an IDL vector called prime_nums.
; We pass into the subroutine the number of primes desired.
;
; NOTE: A large number of "nonsense" function calls are made
;       from within the FORTRAN subroutine in order to test
;       that the link is robust. It found a problem with
;       atan2 and datan2.
;
; **** Set these for yourself ****
DIR = '/bjer3/afl/ensemble/weights/src/idl/'
LIB_DIR = '/usr/lib/'

; This doesn't quite look like a shared object library!!
; May not need all three libraries, but for other tests I did need them.
compile = 1
if (compile eq 1) then begin
    SRC = DIR + 'primes.f '
    OBJ = DIR + 'primes.o '
    OUT = DIR + 'primes.so '
    LIB = LIB_DIR + 'libV77.a ' + LIB_DIR + 'libF77.a ' + $
        LIB_DIR + 'libsunmath.a '

    spawn, 'f77 -c -Kpic ' + SRC
    spawn, 'ld -G -o ' + OUT + OBJ + LIB
endif

num_primes = 100L ; Want 100 primes.
prime_nums = lonarr(num_primes) ; Initialize the prime_nums vector.

; Call a FORTRAN program to do the computation.
a = CALL_EXTERNAL(DIR + 'primes.so', 'primes_', num_primes, prime_nums)

print, prime_nums(*)

end
C This routine accepts input from IDL's CALL_EXTERNAL Function.
C  argc = The number of paramters being passed in from call_external
C  argv = The vector of paramters being passed in from call_external

Subroutine primes(argc, argv) ! Called by IDL

```

Real argc, argv(*) ! Argc and argv are reals

Integer num_expected
parameter (num_expected = 2) ! Number of parameters
 ! expected by programmer

C Obtain # of arguments passed-in and check that this is correct.

```
j = LOC(argc)
if (j .ne. num_expected) then
  return
endif
```

C Call subroutine with two parameters passed in.

```
call primes1( %val(argv(1)), %val(argv(2)) )
```

```
return
end
```

C=====

C Originator: Andrew F. Lough

C

C *** SUN SOLARIS TEST ***

C A rather simple, inefficient, poorly nested, quickly written,
C subroutine (apology accepted?) to compute the first 100 primes.

C It is used to demonstrate the ability of IDL to call a FORTRAN
C subroutine to accomplish some task, accepting an input parameter,
C and returning some values.

C num_primes is passed into this subroutine from IDL.

C

C NOTE:

C Some nonsense function calls are added to see if our link is robust.

C From this test I learned that atan2 and datan2 are symbols which
C could not be found.

```
subroutine primes1(num_primes, prime)
```

```
implicit none
```

```
integer i, j, icount, num_primes
```

```
integer prime(num_primes)
```

```
real r, r2
```

```
double precision d, d2
```

```
prime(1) = 2           ! By definition 1 is not prime.
```

```
prime(2) = 3
```

```
prime(3) = 5           ! Simple method requires specification
```

```
prime(4) = 7      ! of primes under 10.  
icount = 4
```

C Loop through a large number of integers.

C Return only "num_primes" primes.
do 100 i = prime(icount)+2, 1e8, 2

C Test for an even divisor.

```
do 200 j = 3, int( sqrt( float(i) ) ), 2  
  if ( mod(i,j) .eq. 0 ) goto 100      ! Number not prime.  
200  continue
```

C A prime has been found!

```
icount = icount + 1  
prime(icount) = i  
if (icount .gt. num_primes-1) goto 300  ! Only want num_primes
```

```
100  continue
```

C SOME NONSENSE FUNCTION CALLS:

C Sometimes a particular symbol is not found, so the CALL_EXTERNAL

C routine fails. Let's do some nonsense function calls to see if

C our link is robust. Sorry, not all FORTRAN functions are tested.

```
300  i = 100  
      r = 100.  
      d = 100.
```

```
i = iabs(i)  
r = abs(r)  
d = dabs(d)
```

```
i = max0(i, 2)  
r = amax1(r, 3.)  
d = dmax1(d, d*d)
```

```
i = min0(i, 2)  
r = amin1(r, 3.)  
d = dmin1(d, d*d)
```

```
r = sqrt(r)  
d = dsqrt(d)
```

```
r = exp(r)  
d = dexp(d)
```

```
r = alog( abs(r) )
d = dlog( dabs(d) )
```

```
r = alog10( abs(r) )
d = dlog10( dabs(d) )
```

```
r = sin(r)
d = dsin(d)
```

```
r = cos(r)
d = dcos(d)
```

```
i = 100
r = 100.
d = 100.
r2= .5
d2= .5
```

```
r = tan(r)
d = dtan(d)
```

```
r = asin(r)
d = dasin(d)
```

```
r = acos(r)
d = dacos(d)
```

```
r = atan(r)
d = datan(d)
```

C COULD NOT FIND THESE SYMBOLS

C r = atan2(r, r2)

C d = datan2(d, d2)

```
r = sinh(r)
d = dsinh(d)
```

```
r = cosh(r)
d = dcosh(d)
```

```
r = tanh(r)
d = dtanh(d)
```

```
return
end
```

--

Andrew F. Loughé email: afl@cdc.noaa.gov
University of Colorado, CIRES voice: (303) 492-0707
Campus Box 449 fax: (303) 497-7013
Boulder, CO 80309-0449 USA
