The fortran_dynamic_loader module

Neil N. Carlson

June 2013
Version 1.0

Abstract

The `fortran_dynamic_loader` module defines an object-oriented Fortran interface to the system dynamic loader as implemented by the POSIX C functions *dlopen*, *dlclose*, *dlsym*, and *dlerror*.

1 Synopsis

Usage
: `use fortran_dynamic_loader`

Derived Type
: `shlib`

Parameters
: `RTLD_LAZY`, `RTLD_NOW`, `RTLD_LOCAL`, `RTLD_GLOBAL`

Linking
: Link with the system DL library (`-ldl` on Linux) to resolve the symbols *dlopen*, *dlclose*, *dlsym*, and *dlerror*.

2 The shlib derived type

The derived type `shlib` implements the dynamic loading of a shared library and access to data and procedures defined by the library.

2.1 Type bound subroutines

The derived type has the following type bound subroutines. Each subroutine has the optional intent-out arguments `stat` and `errmsg`. If the integer `stat` is present, it is assigned the value 0 if the subroutine completes successfully, and a nonzero value if an error occurs. In the latter case, the allocatable character string `errmsg`, if present, is assigned the error string returned by the underlying system dl library. If `stat` is not present and an error occurs, the error string is written to the preconnected error unit and the program exits with a nonzero status.

`open(filename, mode [,stat [,errmsg]])`

loads the shared library file named by the character argument `filename` and associates it with the `shlib` object. If `filename` contains a slash (`/`), then it is interpreted as a relative or absolute pathname. Otherwise the dynamic loader searches a certain list of directories for the library; see *dlopen*(3) for a detailed description of the search process.

One of the following two values must be passed as the `mode` argument:

`RTLD_LAZY`

Only resolve symbols as the code that references them is executed (lazy binding).

`RTLD_NOW`

All undefined symbols in the library are resolved before the `open` procedure returns. An error occurs if this is not possible. This is also the behavior if the environment variable `LD_BIND_NOW` is set to a nonempty string.

One of the following values may optionally be or'ed with the preceding values before being passed as the `mode` argument; e.g., `mode=ior(RTLD_LAZY,RTLD_GLOBAL)`.

`RTLD_GLOBAL`

The symbols defined by this library will be made available for symbol resolution of subsequently loaded libraries.

`RTLD_LOCAL`

This is the converse of `RTLD_GLOBAL` and the default. Symbols defined by this library are not made available to resolve references in subsequently loaded libraries.

See *dlopen*(3) for more details.

`close([stat [,errmsg]])`

decrements the reference count on the shared library. When the reference count reaches zero, the shared library is unloaded. See *dlclose*(3) for a detailed description of the behavior.

`func(symbol, funptr [,stat [,errmsg]])`

returns the memory address where the specified function symbol from the shared library is loaded. The character argument `symbol` gives the symbol name, and the address is returned in the `type(c_funptr)` argument `funptr`. The caller is responsible for converting this C function pointer value to an appropriate Fortran procedure pointer using the subroutine `c_f_procpointer` from the intrinsic `iso_c_binding` module.

`sym(symbol, symptr [,stat [,errmsg]])`

returns the memory address where the specified data symbol from the shared library is loaded. The character argument `symbol` gives the symbol name, and the address is returned in the `type(c_ptr)` argument `symptr`. The caller is responsible for converting this C pointer value to an appropriate Fortran data pointer using the subroutine `c_f_pointer` from the intrinsic `iso_c_binding` module.

3 Example

```
use fortran_dynamic_loader
use,intrinsic :: iso_c_binding, only: c_funptr, c_f_procpointer

abstract interface
  real function f(x)
    real, value :: x
  end function
end interface
procedure(f), pointer :: cbrtf

type(shlib) :: libm
type(c_funptr) :: funptr

!! Load the C math library libm.so and calculate the cube
!! root of 8 using the function cbrtf from the library.
call libm%open ('libm.so', RTLD_NOW)
call libm%func ('cbrtf', funptr)
call c_f_procpointer (funptr, cbrtf)
if (cbrtf(8.0) /= 2.0) print *, 'error'
call libm%close
```

4 Bugs

Bug reports and improvement suggestions should be directed to `neil.n.carlson@gmail.com`