Experience with MELCOR user defined extensions in C and Lua

Paul Boneham - Jacobsen Analytics Ltd
www.jacobsen-analytics.com

Presentation at EMUG
Zagreb Croatia - 25 to 27 April 2018

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Overview

- MELCOR capabilities – external shared libraries
- Shared libraries in FORTRAN
- Shared libraries in C
- Embedding the Lua interpreter
- Examples

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

MELCOR capabilities – external shared libraries

- According to the reference manual for MELCOR 2.2.9541 control functions may be defined by the user in an external shared library* (Linux) or DLL (Windows)
  - * This presentation describes work performed on the Linux OS, so will refer to shared libraries
- Sample FORTRAN code and a make file is supplied with the MELCOR code

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Shared libraries in FORTRAN

- MELCOR is compiled with the Intel FORTRAN compiler (version 11.1 – released in 2009)

- Not successful compiling with gfortran using the sample UDF files
  - So need a work around, or purchase Intel FORTRAN compiler and try with that

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Hello World in FORTRAN - *do I really want to do this anyway*?

```
C
C
      IMPLICIT NONE
      WRITE (6,100)
 100  FORMAT(12HHello world!)
      STOP
      END
C
C
```

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Inspection of UDF source code sample reveals …

```
function FUNn (ptr_cftype,ARG,IERROR) bind(C, name="funn")
```

- FORTRAN is telling the compiler to build a C interface for the shared library
- So, why not create the shared library from C source?

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Working C shared library source

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

////////////////////////////////////////////////////////////////////////
// Working shared library C source (Linux)
////////////////////////////////////////////////////////////////////////

double funn(char * ptr_cftype, double * args, long * ierror ) {
    double result;
    result = 10.0;  // doesn't do any calculations all FUNn return 10.0
    return result;
}

void get_pedigree(char * str) {
    strcpy(str, "Description of shared library for MELCOR");
    printf("Hello world!\n"); // not needed, but let's us see when this function is called
}
```

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Compile the single C source file

```
# Compile and link shared library from C source
gcc -fPIC -fno-omit-frame-pointer -m32 -c melcor_user_extension.c -o melcor_user_extension.o
gcc -fPIC -fno-omit-frame-pointer -m32 -shared melcor_user_extension.o -o melcor_user_extension.so
```

-m32 - make a 32 bit shared library. MELCOR is a 32 bit executable. Build on i386 Linux distribution, problems experienced on x86_64 even with 32 bit support.

Use gcc version 4.7 – shared library built with later versions segfaults on loading by MELCOR

- *suspect* this is related to 2009 version of Intel compiler being incompatible with changes to gcc code generation on Linux more recently

Works on Debian 7 – does not work on Debian 8 and later. Incompatible ABI on more modern Linux distributions.

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Hello World from User Defined Extension

Run MELGEN …

```
 Do you want to overwrite (O) or abort (A)
o

 COMMAND-LINE:    ../bin/melgen test.inp
 COMMAND-LINE ARGUMENTS:
Hello world!
 Opening user input file test.inp
Hello world!
 Restart written  TIME =  -1.800000E+03 CYCLE=         0
```

Says hello – twice …

… MELGEN and MELCOR actually load and unload the shared library several times initially, carrying out various checks on linkability of expected functions

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Do I really want to do this in C?

```c
#include "stdio.h"

void main(int arg_count, char ** args) {
        printf("Hello world!\n");
}
```

Whereas in Lua a fully functional program to do the same is

```
print "Hello world!"
```

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

So what is Lua anyway? And does anyone use it?

- Scripted language, no compile – run cycle
- Easy to embed (more later)
- Very popular extension language for games
- Used as extension language in NGINX web server (runs around 30% of sites on internet)
- Luajit "just in time" compiler runs Lua scripts ~20 times faster than Python (another popular scripting language)

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

How to embed Lua into our C extension (1)

- Include luajit headers (assuming we want to use luajit):

```
#include "luajit-2.0/lua.h"
#include "luajit-2.0/lualib.h"
#include "luajit-2.0/lauxlib.h"
```

- Add code to the shared library source:
  - Create a “Lua state” object for communication with Lua
  - Load Lua script
  - Call functions in loaded script

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

How to embed Lua into our C extension (2)

- Can be done in around 60 lines of code
  - including error checks and validity checks on loaded Lua script
- Requires some interaction with a “stack” to pass function arguments and extract return values
- Once done and shared library compiled, can forget about all this and just write extensions in Lua

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Examples (1)

```
function fun2(arg1, arg2, arg3, arg4, arg5)
    -- updates steady condition and calculates reactor power, depending on time
    -- steady condition status available in enviroment, may be used by other funn
    if environment.steady == nil then
        environment.steady = 1
    end
    if arg1 >= 0.0 then
        environment.steady = 0
    end
    if environment.steady == 1 then
        -- return full power
        return 20000.0
    else
        -- return decay power
        return 1000.0 - arg1 * 0.00667 -- simplified function
    end
end
```

Calculates heat input and keeps track of state of simulation

environment.steady is a global variable which is “persistent” (stored) from one call of fun2 to another

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Comment on example (1)

- MELCOR does not call dlclose() when a simulation is finished
- Call to dlclose() gives external library opportunity to carry out book-keeping tasks such closing files
- So, in previous example, should add a call to flush() each timestep, otherwise risk of truncated output

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Examples (2)

Store plot variables in a simple text file, suitable for import to Excel or plotting with gnuplot

```
function fun1(arg1, arg2, arg3, arg4, arg5)
    if environment.save_exec_time == nil then
        environment.f:write(arg1)
        environment.save_exec_time = arg1
    end
    if arg1 > environment.save_exec_time then
        environment.f:write("\n", arg1)
        environment.save_exec_time = arg1
    end
    environment.f:write("\t",arg2,"\t",arg3,"\t",arg4,"\t",arg5)
end
```

Only writes 1st column (time) on first call on each iteration

Allows function to be re-called several times by different CF in MELCOR – this way, an arbitrary number of plot variables can be saved

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Other possibilities

- Simple simulator – report variables to user, allow input, keep simulation time aligned to real time
  - MELCOR has to wait for any pause in UDF execution
- Use lua sockets or lua SQL to send data to other PCs on network (centralised store of output data or run status etc)
- Monte Carlo simulation – use Lua functions to randomly assign different values to sampled variables

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Summary, conclusions

- Use of user defined extensions lead to creation of working C shared library, then embedding of Lua interpreter
- Easy, effective way to write custom functions
- Allows to reduce number of CF needed for some tasks
  - – e.g., implementing initial conditions such as LOCA/non-LOCA, SLB, LOFW much simplified with Lua functions that can store state variables (approx. 50% reduction in number of CF for these tasks)
- Various interesting/useful possibilities to explore

Jacobsen analytics
ENGINEERING RISK SOLUTIONS

Some issues (which should be solvable)

- Restriction to gcc 4.7, i386 and older Linux distributions (Debian 7)
  - Might be solved if MELCOR is recompiled with a newer version of Intel FORTRAN or even gfortran
- dlcose() issue: MELCOR does not call dlclose() when a simulation is finished
  - This should probably be fixed …

Jacobsen analytics
ENGINEERING RISK SOLUTIONS