CS35L Software Construction Laboratory

Lab 1: Nandan Parikh

Week 7; Lecture 2

Example – Creating/Using Static Library

1. Create C file for your library functions
   lib_mylib.c
2. Create Header file for your library
   lib_mylib.h
3. Compile these files (No linking)
   gcc -c lib_mylib.c -o lib_mylib.o
4. Create a Static Library
   ar rcs lib_mylib.a lib_mylib.o
   (The archiver, also known simply as ar, is a Unix utility that maintains groups of files as a single archive file)

5. Compile the main code
   gcc -c main.c -o myMain.o
6. Compile and Link the main with the static library
   gcc –o mainCode myMain.o –L. –l_mylib

Creating/Using Shared Libraries: Dynamic Linking

1. Create C file for your library functions

   lib_mylib.c
2. Create Header file for your library

   lib_mylib.h
3. Compile these files (No linking)

   gcc –fPIC -c lib_mylib.c -o lib_mylib.o
4. Create a Dynamic Library

   gcc –shared –o lib_mydynamiclib.so lib_mylib.o
5. Get the main code

   main.c
6. Compile and Link the dynamic library from the path specified

   gcc -Wl,-rpath=$PWD -o myCode main.c -L. -l_mydynamiclib

GCC Flags

- -fPIC: Compiler directive to output position independent code, a characteristic required by shared libraries.
- -l*library*: Link with "lib*library*.a"
  - Without -L to directly specify the path, /usr/lib is used.
- -L: At compile time, find the library from this path.
- -Wl,rpath=.: -Wl passes options to linker.
  - -rpath at runtime finds .so from this path.
- -c: Generate object code from c code but do not link
- -shared: Produce a shared object which can then be linked with other objects to form an executable.
- https://gcc.gnu.org/onlinedocs/gcc/Link-Options.html#Link-Options

How are libraries dynamically loaded ?

Table 1. The Dl API

Function	Description
dlopen	Makes an object file accessible to a program
dlsym	Obtains the address of a symbol within a `dl`opened object file
dlerror	Returns a string error of the last error that occurred
dlclose	Closes an object file

Source : https://www.ibm.com/developerworks/library/l-dynamic-libraries/

Dynamic loading - Basics

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

int main(int argc, char* argv[]) {
 void (*myfunc)(); void *error; void *dl_handle;

 dl_handle = dlopen("lib_mydynamiclib.so", RTLD_LAZY);


 if(!dl_handle) {
   printf("Error");
   return 1;
}
 myfunc = dlsym(dl_handle, "myLibrary"); error = dlerror();

 if(error != NULL) {
  printf("Error getting fn");
  return 1;
 }
 myfunc();

 return 0;
}
```

Creating/Using Shared Libraries: Dynamic Loading

Steps 1 to 4 similar as the Dynamic Linking! Relief!

5. Get main code which has dlopen/dlclose…
6. Compile the code, at runtime find the dynamic lib

```
gcc –ldl -Wl,-rpath=$PWD -o myCode main.c
```

Attributes of Functions

- Used to declare certain things about functions called in your program
  - Help the compiler optimize calls and check code
- Also used to control memory placement, code generation options or call/return conventions within the function being annotated
- Introduced by the attribute keyword on a declaration, followed by an attribute specification inside double parentheses
- Reference:

https://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Function-Attributes.html

Attributes of Functions

- __attribute__ ((__constructor__))
  - Is run when dlopen() is called
- __attribute__ ((__destructor__))
  - Is run when dlclose() is called
- Example:

```
__attribute__ ((__constructor__))
void to_run_before (void) {
        printf("pre_func\n");
}
```

Homework 7 : (Use lnxsrv09)

- Split randall.c into 4 separate files
- Stitch the files together via static and dynamic linking to create the program
- randmain.c must use dynamic loading to link up with randlibhw.c and randlibsw.c
- randcpuid to be used as a static library
- Write the randmain.mk makefile to do the linking

Homework 7

- randall.c outputs N random bytes of data
  - Look at the code and understand it
    - main function
      - Checks number of arguments (name of program, N)
      - Uses helper function to check for HW support
      - Uses helper functions to generate random number using HW/SW
    - Helper functions that check if hardware random number generator is available, and if it is, generates number
      - HW RNG exists if RDRAND instruction exists
      - Uses cpuid to check whether CPU supports RDRAND (30th bit of ECX register is set)
    - Helper functions to generate random numbers using software implementation (/dev/urandom)

Homework 7

- Divide randall.c into dynamically linked modules and a main program. Don't want resulting executable to load code that it doesn't need (dynamic loading)
- randall.c = randcpuid.c + randlibhw.c + randlibsw.c + randmain.c
  - randcpuid.c: contains code that determines whether the current CPU has the RDRAND instruction. Should include randcpuid.h and include interface described by it.
  - randlibhw.c: contains the hardware implementation of the random number generator. Should include randlib.h and implement the interface described by it.
  - randlibsw.c: contains the software implementation of the random number generator. Should include randlib.h and implement the interface described by it.
  - randmain.c: contains the main program that glues together everything else. Should include randcpuid.h (as the corresponding module should be linked statically) but not randlib.h (as the corresponding module should be linked after main starts up). Depending on whether the hardware supports the RDRAND instruction, this main program should dynamically load the hardware-oriented or software-oriented implementation of randlib.

Homework 7 – randmain.mk

- Create shared libraries (Check the example explained in class for hitns)
  - randlibsw.so :
  - randlibhw.so:
- Create library for static linking
  - randcpuid.o:
- Create object file for randmain
  - randmain.o:
- Build randmain
  - randmain: -ldl -Wl,-rpath=${PWD}
  - If you used ar to create static library, use –lstaticlibrary option to statically link the library and optionally use –L option to specify the path for the statically linked library