CS 35L

LAB 8,

TA: Sucharitha Prabhakar
EMAIL ID: prabhakarsucharitha@gmail.com

Outline

Static linking

Dynamic linking

Gcc flags

-fPIC: Compiler directive to output position independent code, a characteristic required by shared libraries.

-lXXX: Link with "libXXX.so”

Without -L to directly specify the path, /usr/lib is used.

-L: At compile time, find .so from this path.

Gcc flags

-Wl : passes options to linker

rpath : -rpath at runtime finds .so from this path.

-c: Generate object code from c code.

-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?

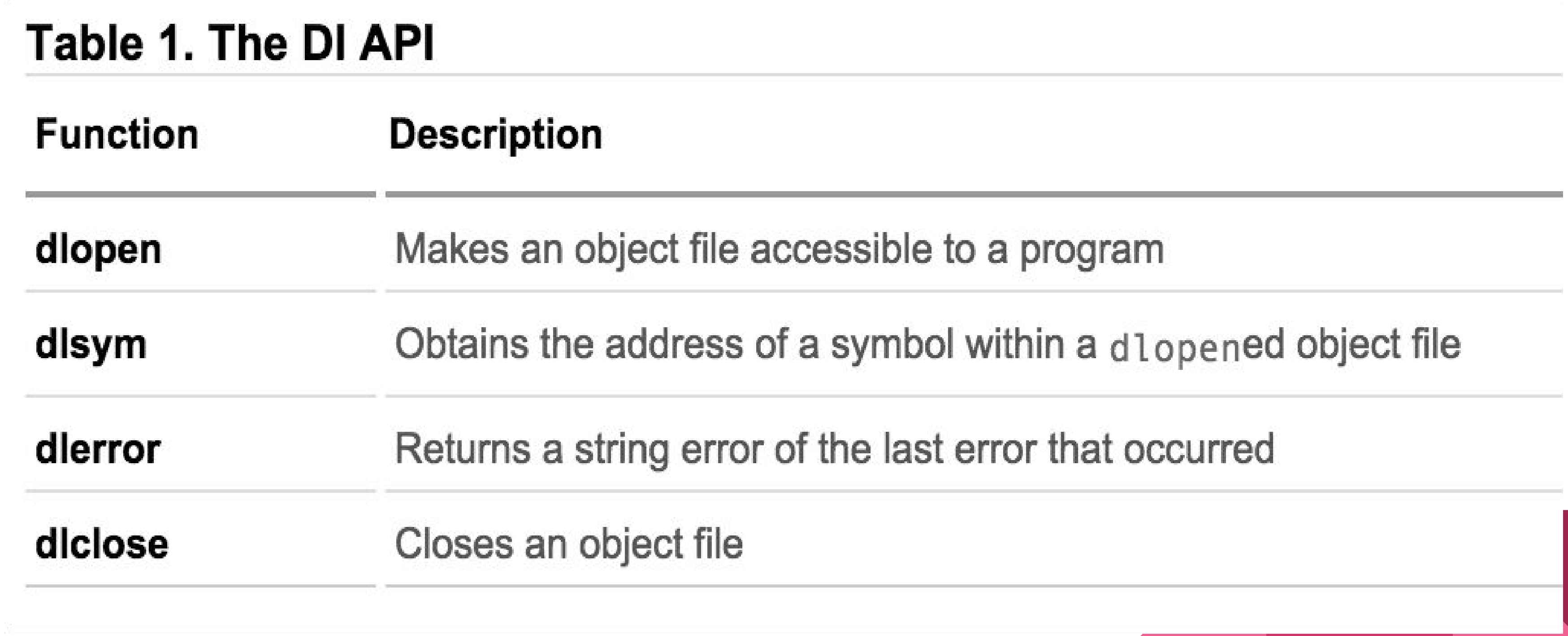

Table 1. The Dl API

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

Sample program

calc_mean.h

```
double mean(double, double);
```

Sample program

Calc_mean.c

```
double mean(double a, double b) {
  return (a+b)/2;
}
```

Sample program

```
#include <stdio.h>

#include "calc_mean.h"

int main(int argc, char* argv[]) {

  double v1, v2, m; v1 = 5.2; v2 = 7.9;

  m  = mean(v1, v2);

  printf("The mean of %3.2f and %3.2f is %3.2f\n", v1, v2, m);

return 0; }
```

Creating and using a static library

```
gcc -c calc_mean.c -o calc_mean.o

ar rcs libmean.a calc_mean.o

gcc -static main.c  -L.  -lmean  -o  statically_linked
```

Creating and using a dynamic library

```
gcc -c -fPIC calc_mean.c -o calc_mean.o

gcc -shared -o libmean.so  calc_mean.o

gcc calc_mean.c -o dynamically_linked  -L.  -lmean

LD_LIBRARY_PATH=/u/cs/class/cs35l/cs35lt9/Documents/lab8
./dynamically_linked
```

(**LD_LIBRARY_PATH** is an environment variable you set to give the run-time shared library loader (ld.so) an extra set of directories to look for when searching for shared libraries. Multiple directories can be listed, separated with a colon (:))

```
gcc main.c -o dynamically_linked -L. -lmean
-Wl,-rpath=/u/cs/class/cs35l/cs35lt9/Documents/lab8
```

Creating and using a dynamic library

```
gcc main.c -o dynamically_linked -L. -lmean

-Wl,-rpath=/u/cs/class/cs35l/cs35lt9/Documents/lab8
```

rpath - Add a directory to the runtime library search path. This is used when linking an ELF executable with shared objects. All -rpath arguments are concatenated and passed to the runtime linker, which uses them to locate shared objects at runtime.

Why -L and -rpath ?

The linker needs to know what symbols should be supplied by the dynamic library.

For this reason, -L is needed to specify where the file to link against is.

-rpath comes into play at runtime, when the application tries to load the dynamic library. It informs the program of an additional location to search in when trying to load a dynamic library.

-fPIC and -shared

-shared (linker option) - Produce a shared object which can then be linked with other objects to form an executable. For predictable results, you must also specify the same set of options used for compilation (-fpic, -fPIC) when you specify this linker option

-fpic (compile time option) - Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machine. Such code accesses all constant addresses through a global offset table (GOT). The dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of GCC; it is part of the operating system)

Attributes of a function

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

Attributes of a function

In GNU C, you declare certain things about functions called in your program which help the compiler optimize function calls and check your code more carefully.

The keyword __attribute__ allows you to specify special attributes when making a declaration

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

Attributes of a function

Example:

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

Homework

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, dynamic linking to link up with randlibhw.c and randlibsw.c (using randlib.h)

Write the randmain.mk makefile to do the linking

Homework

- randall.c outputs N random bytes of data
- Look at the code and understand it; 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)
- main function
  - Checks number of arguments (name of program, N)
  - Converts N to long integer, prints error message otherwise
  - Uses helper functions to generate random number using hw/sw

Homework

- 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)
  - 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.