CS35L – Winter 2019

Slide set:	7.2
Slide topics:	Dynamic linking
Assignment:	7

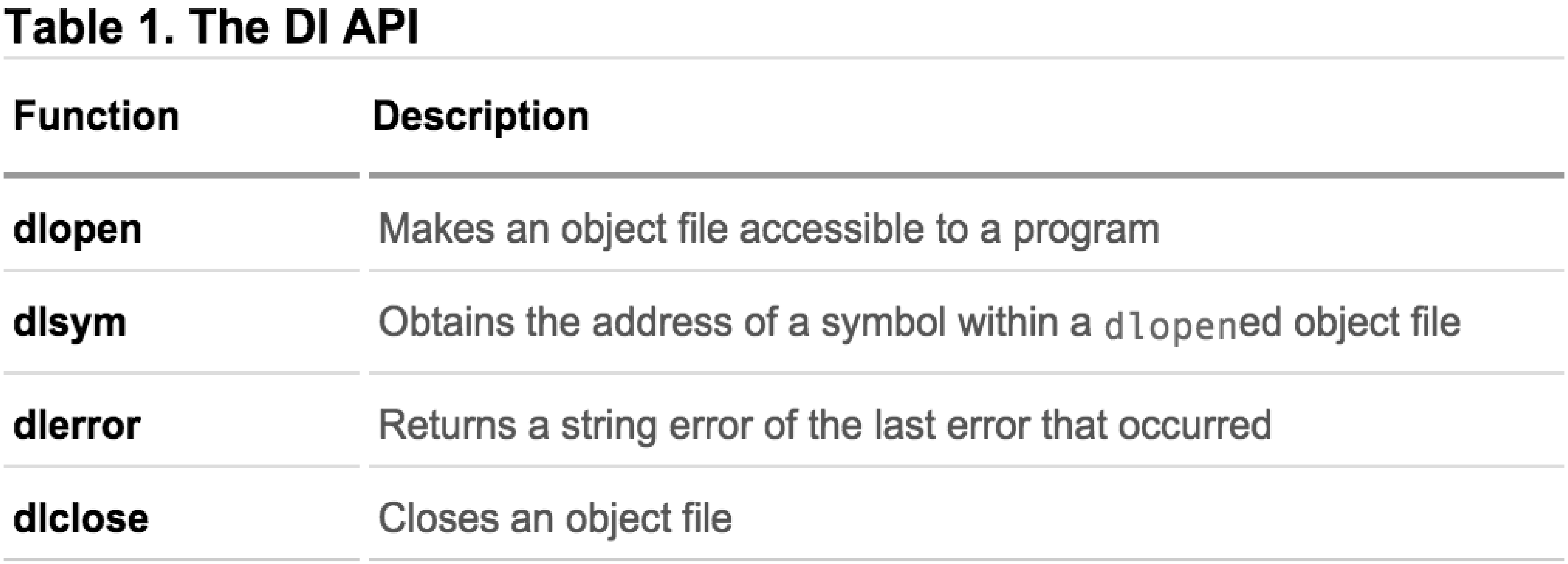

Table 1. The Dl API

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

HOW ARE LIBRARIES DYNAMICALLY LOADED?

Dynamic loading

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

int main(int argc, char* argv[]) {
  int i = 10;
  void (*myfunc)(int *); void *dl_handle;
  char *error;
  dl_handle = dlopen("libmymath.so", RTLD_LAZY);//RTLD_NOW
  if(!dl_handle) {
   printf("dlopen() error - %s\n", dlerror()); return 1;
  }
  //Calling mul5(&i);
  myfunc = dlsym(dl_handle, "mul5"); error = dlerror();
  if(error != NULL) {
   printf("dlsym mul5 error - %s\n", error); return 1;
  }
  myfunc(&i);
  //Calling add1(&i);
  myfunc = dlsym(dl_handle, "add1"); error = dlerror();
  if(error != NULL) {
   printf("dlsym add1 error - %s\n", error); return 1;
  }
  myfunc(&i);
  printf("i = %d\n", i);
  dlclose(dl_handle);
  return 0;
}
```

- Copy the code into main.c `gcc main.c -o main -ldl`
- You will have to set the environment variable **LD_LIBRARY_PATH** to include the path that contains libmymath.so

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

- 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

- 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

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.

- Create shared libraries
  - `randlibsw.o : -fPIC, -c and other existing options`
  - `randlibhw.o : -fPIC, -c and other existing options`
  - `randlibsw.so : -shared option`
  - `randlibhw.so: -shared option`
- Create library for static linking – 2 options
  - `randcpuid.o: -c option, or`
  - `ar` command to create an archive of static libraries
- Create object file for randmain
  - `randmain.o: -c option`
- Build randmain
  - `randmain: -ldl -Wl,-rpath=${PWD}`
  - If you used ar to create static library, use –l*staticlibrary* option to statically link the library and optionally use –L option to specify the path for the statically linked library

Homework 7s – `randmain.mk`