CS 35L

Week 9

TA: Tomer Weiss
March-03-2016

goo.gl/9BYufI

Slides

Announcements

- Student presentations today:
  - Likai: Building Living, Breathing Supercomputers
  - Jiaming: A new algorithm from MIT could protect ships from 'rogue waves' at sea
  - Yu: The Incredible Things Google's New Robot Can Do

  web.cs.ucla.edu/classes/winter16/cs35L/assign/assign10.html

- Next week:
  - Write your topic here
  - Not registering you topic beforehand may result in rescheduling of your presentation
  - For reference on presentation, grading, please refer to this rubric.

Announcements - Please Fill teaching evaluation survey

Students have until 8:00 AM Saturday, March 12 to log into MyUCLA to complete evaluations for your courses listed below:

COM SCI 35L section 6

You have the option to use class time for your students to complete their evaluations with smartphones, iPads, and tablets. Please note that not all classrooms are WI-FI enabled.

For more details about this process, please contact your departmental Evaluation Coordinator or visit our website at:

http://www.oid.ucla.edu/assessment/eip

Thank you!
Evaluation of Instruction Program
eip@oid.ucla.edu

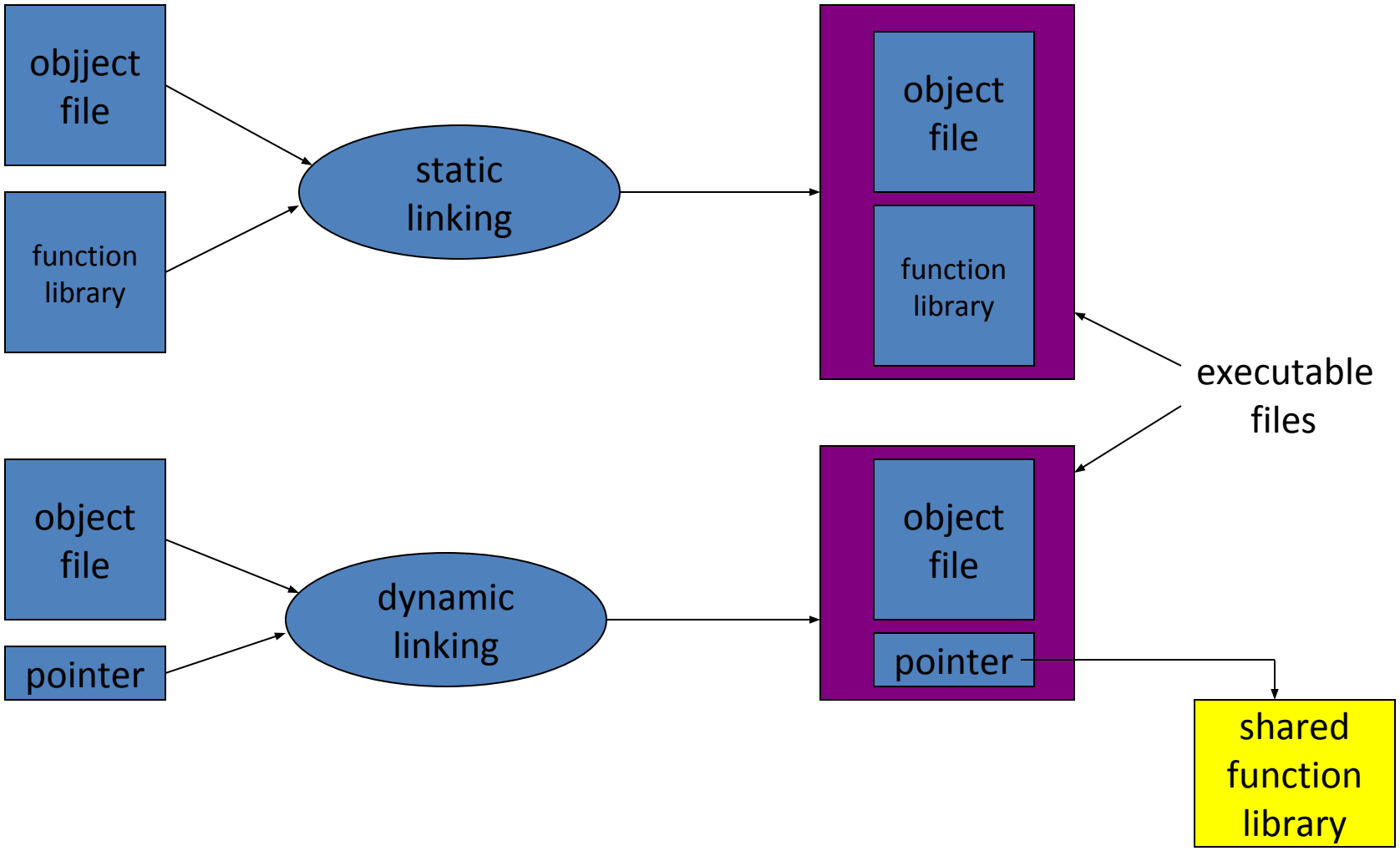
Dynamic Linking

Week 9

Dynamic Linking

- Allows a process to add, remove, replace or relocate object modules during its execution.
- If shared libraries are called:
  - Only copy a little reference information when the executable file is created
  - Complete the linking during loading time or running time
- Dynamic libraries are typically denoted by the .so file extension
  - .dll on Windows

Smaller is more efficient



How are libraries dynamically linked?

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

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

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.
- `-Wl,rpath=.`: `-Wl` passes options to linker. `-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.

Creating static and shared libs in GCC

- mymath.h

```
#ifndef _ MY_MATH_H
#define _ MY_MATH_H
void mul5(int *i);
void add1(int *i);
#endif
```

- mul5.c

```
#include "mymath.h"
void mul5(int *i)
{
  *i *= 5;
}
```

- add1.c

```
#include "mymath.h"
void add1(int *i)
{
  *i += 1;
}
```

- gcc -c mul5.c -o mul5.o
- gcc -c add1.c -o add1.o
- ar -cvq libmymath.a mul5.o add1.o → (static lib)
- gcc -shared -fpic -o libmymath.so mul5.o add1.o → (shared lib)

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

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");
}
```

Use the above in your implementation!

Homework 9

- Divide randall.c into dynamically linked modules and a main program
  - We 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.

Homework 9

- 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 9

- 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 (30^{th} 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

Pointers

- Useful overview of dynamic libraries: Click here
- Man page for DL API: Click here
- www.ibm.com/developerworks/library/l-dynamic-libraries/
- www.ibm.com/developerworks/library/l-lpic1-102-3/
- tldp.org/HOWTO/Program-Library-HOWTO/index.html
- www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html
- man7.org/linux/man-pages/man7/vdso.7.html

Homework

web.cs.ucla.edu/classes/winter16/cs35L/assign/assign9.html