Software Construction Laboratory

Week 8

Lab 3

Mushi Zhou

Winter 2017

UCLA

Prof. Paul Eggert

Outline

- Dynamic Linking
- From source file to executable
- Static Linking
- Dynamic Linking
- Two ways of doing dynamic linking
- Why Dynamic Linking
- Dynamic linking in practice
- How to do dynamic linking in makefile

From Source Files to Executable

- There are two stages
- Compile + Link
- The first stage is compilation
- Compilation turns source code into object modules
- The second is linking
- Linking combines object modules together to form an executable

Why Do We Need Two Stages?

- There are three major reasons:

1. The object files, i.e. *.o files are not readable, this allows third party libraries to be included in your executable without you seeing their source code. i.e. using pre-existing compiled object files from third parties to build your program
2. Object files serve as the middle ground for compiling codes in different languages (C and assembly code for example) and then linking them all together
3. The third reason is we can then use dynamic linking for easier library updates and save resources at runtime (We go into details on this)

How Are the Stages Reflected?

- When use gcc to compile:
- -c option tell the compiler to compile only, without linking
- The output with –c option should be in .o extension
- Without –c option, the two stages are combined and you get executable directly from gcc
- After getting .o files, we can link multiple .o files together to build executable. gcc on .o files produce executables directly
- The default linking is static
- You can use –shared to make .o files to be dynamically linked libraries

Static Linking

- When you statically link a file into an executable, the contents of that file are included at link time
- In other words, the contents of the file are physically inserted into the executable that you will run
- You end up with one single executable file that is not dependent on any other files
- This executable contains all the source file contents you include during the linking process

Dynamic Linking

- When you link dynamically, a pointer to the file being linked in (the file name of the file, for example) is included in the executable and the contents of file are not included at link time
- It's only when you later run the executable that these dynamically linked files are bought into the memory to be executed
- You end up with one executable that depends on shared libraries (i.e. can’t run correctly without the presence of the linked shared libraries
- ldd command shows all dynamic linked libraries for a particular executable

Why Dynamic Linking?

- Statically-linked files are 'locked' to the executable at link time so they never change. They become a part of the executable
- A dynamically linked file referenced by an executable can change just by replacing the file on the disk
- This allows updates to functionality without having to re-link the code
- The loader re-links every time you run your executable
- Good for bug fixings, library update -> no need to recompile everything
- What’s the potential problem here?
- Speed/Memory are not key factors here.
- Flexibility is the major reason, but it comes with drawbacks (file lost?)

Two Ways to Do Dynamic Linking

1. #include at the beginning in your c/c++ source files

   Those included are system library files that pre-complied in system files, include tells the compiler the address of those **system-supplied library** files. Then those files are accessed when necessary during your program execution. But the contents of those files are never copied to your program executable

2. Use dlsym library within your code to access shared library files

   You can build **your own** shared libraries with the –shared option in compilation. Then you can link your program with “-ldl” option allow it to be able to access shared libraries (You are doing this in assignment 8)

First Way to Do Dynamic Linking

- Linking System Libraries

- Most of the c libraries are pre compiled, and are available to your program as shared libraries
- When you state #include <> at the top of your source code, you are telling the gcc to dynamically link these system libraries when build
- There are automatically done, you need to do nothing in your makefile
- Functions in the library are immediately available to use

Second Way to Do Dynamic Linking

- dl library calls

#include <dlfcn.h>

- You build your shared libraries manually
- You access your shared libraries manually in your code by invoking
- -dlopen(“name of library”, mode) => get the library handle
- -dlsym(handl, “name of function in library”) => get function pointer
- -dclose() => close the library handle
- You need to have knowledge of what’s in the library
- You are performing this in assignment 8

Dynamic Linking in Practice

- In window, .dll files are very common, at least before window 10
- Dll => Dynamic Linked Library
- In unix, shared library files are often in .so extension

- For production software distributions, they often comes with a package of a mixture of executables and shared libraries of their own.
- The #include libraries as the same as long as C/C++ environments are provided
- The shared libraries comes with the packages are all self-build libraries and linked using dlsym library calls

Dynamic Linking in Makefile

- Use –shared option to build .so libraries
- Use –o option to specify output file name
- For safety, need to use –fPIC option when compiling
- -fPIC => file format, position independent code (This is because your library functions are called in random locations in your main program)
- -ldl –Wl, -rpath=$(PWD) => Provide the path for where to look for shared libraries when dlsym fuction calls are made (in the particular case, the current directory)

About Assignment 8

- I will be the grader for this assignment for the entire class
- I will post a detailed note on Piazza by tonight
- Follow the notes, you should find many hints
- You will have to study the given program yourself
- This assignment is not too difficult
- You will learn to build shared libraries
- You will learn to do dynamic linking using dlsym library calls
- You will learn to write makefile