Week 11

Announcements

- HW7, ADV7 due by 11:59 PM on Nov 13 (tonight)
- HW8, ADV8 due by 11:59 PM on Nov 20
  - I noticed a trend in the feedback: you don't have to do every single advanced assignment! Check the syllabus for details.

Lecture 10: Libraries

"How do I X?"
"Just use Boost"

Overview

- What are libraries?
- Using existing libraries
- Creating your own

What are libraries?

- Libraries are collections of code and data that can be used by other programs.

compiled executables targeting the host architecture and OS (i.e. not targeting a VM like Java or C#), specifically for C/C++

- C/C++ libraries happen to serve the backbone of a *complete* OS

Types of libraries

Source libraries

- Source code for a library is provided
- Pretty much exactly like a normal project

- Provided as an *archive* of pre-compiled object code
  - Files are named `lib<library name>.a` e.g. `libcoolthing.a`
  - `.a` stands for "archive"

  tossed into the executable
  and won't change wherever the executable goes
  - Incurs a size cost since the library is a part of the executable

Types of libraries

Dynamic/shared libraries

- A collection of object code meant to be shared by multiple programs
  - One file `/lib/libm.so` shared among many programs that use it
  - Files are named `lib<library name>.so` e.g. `libncurses.so`
  - `.so` stands for "shared object" (another name you see is "dynamic shared objects")
  - `.dylib` and `.dll` are macOS and Windows counterparts

is marked as a dependency in the executable
  - You can check this out using `readelf -d` or `ldd` on an executable
  - ELF is the file format used for object code and binary executables on Linux systems (as well as many other systems)

program load time
  - Avoids the static linking size cost at the cost of being dependent on the system for the library
  - You sometimes see them packaged along with applications (ever see `.dll` files come with some program?), or they're listed as dependencies for your package manager to resolve

Using existing libraries

Source libraries

- Trivial: it's just more source code and add it as such
- May have to include the headers in the include path (`-I`)
    - You might've run into this for Adv 7...
- These are so uninteresting that I'm not going to mention them anymore

Using existing libraries

Static and Dynamic Libraries

- Using either is very similar
- The `-l<library name>` linker flag allows you to specify a library
    - Searches through `/lib`, `/usr/lib`, in directories listed by `/etc/ld.so.conf`, and directories in `LD_LIBRARY_PATH`
    - You can specify additional directories with `-L`
    - `-lpng` for `libpng.a` and `libpng.so`
    - `gcc -o myapp $(SRCS) -lm`
    - `gcc -o myapp $(SRCS) -Llib -lstaticlib`
    - (under the hood, `gcc` is passing these linker flags to `ld`)

Static and Dynamic Libraries

But what if they conflict?

- Note how `-l` doesn't care about static vs dynamic
- `.so` has a higher precedence over `.a`

- e.g. `-l:libm.a`
- This is more of a nuclear option
- Beware that this will make it *only link statically*: what if you don't have a static version of the C library?

Creating your own libraries

Static libraries

1. Compile the objects
   - `gcc -c -o somecode.o somecode.c`

- `ar rcs libmylib.a somecode.o morecode.o yaycode.o`
- `r`: command, insert files with replacement (in case the archive already exists)
- `c`: option, "create the archive"
- `s`: option, "write an object file index into the archive"

Creating your own libraries

Dynamic libraries

1. Compile the objects
   - `gcc -c -fPIC -o somecode.o somecode.c`

```
left for EECS 370 and EECS 482
```

- `gcc -shared -fPIC -o libmylib.so somecode.o morecode.o yaycode.o`
- `-shared`: "produce a shared object"
- Versioning (*soname* fun)
- Maintaining binary compatibility
- Really great read
- Recommended by my interviewer during the interview for an internship

Questions?