

Week 12

Announcements

- HW8, ADV8 due by 11:59 PM on Apr 1
- HW9, ADV9 due by 11:59 PM on Apr 8
 - I'll get this out sometime tonight

Short lecture today

This'll be a chill week

I'll spend some time at the end hacking about with libraries

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.

What are libraries?

- Libraries are collections of code and data that can be used by other programs.
- Cool stuff written by other people so you don't have to

What are libraries?

- Libraries are collections of code and data that can be used by other programs.
- Cool stuff written by other people so you don't have to
 - GUI (`libxcb`, `libX11`, `libgtk-3`, `libQtCore`)

What are libraries?

- Libraries are collections of code and data that can be used by other programs.
- Cool stuff written by other people so you don't have to
 - GUI (`libxcb`, `libX11`, `libgtk-3`, `libQtCore`)
 - Graphics (`libvulkan`, `libGL`)

What are libraries?

- Libraries are collections of code and data that can be used by other programs.
- Cool stuff written by other people so you don't have to
 - GUI (`libxcb`, `libX11`, `libgtk-3`, `libQtCore`)
 - Graphics (`libvulkan`, `libGL`)
 - File formats (`libpng`, `libjpeg`, `libmpeg2`)

What are libraries?

- Libraries are collections of code and data that can be used by other programs.
- Cool stuff written by other people so you don't have to
 - GUI (`libxcb`, `libX11`, `libgtk-3`, `libQtCore`)
 - Graphics (`libvulkan`, `libGL`)
 - File formats (`libpng`, `libjpeg`, `libmpeg2`)
 - ...and more

What are libraries?

- Libraries are collections of code and data that can be used by other programs.
- Cool stuff written by other people so you don't have to
 - GUI (`libxcb`, `libX11`, `libgtk-3`, `libQtCore`)
 - Graphics (`libvulkan`, `libGL`)
 - File formats (`libpng`, `libjpeg`, `libmpeg2`)
 - ...and more
- For this lecture we'll be focusing more on the context of compiled executables targeting the host architecture and OS (i.e. not targeting a VM like Java or C#), specifically for C/C++

What are libraries?

- Libraries are collections of code and data that can be used by other programs.
- Cool stuff written by other people so you don't have to
 - GUI (`libxcb`, `libX11`, `libgtk-3`, `libQtCore`)
 - Graphics (`libvulkan`, `libGL`)
 - File formats (`libpng`, `libjpeg`, `libmpeg2`)
 - ...and more
- For this lecture we'll be focusing more on the context of compiled executables targeting the host architecture and OS (i.e. not targeting a VM like Java or C#), specifically for C/C++
 - That being said, the idea of a library is fairly universal
 - 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

Types of libraries

Source libraries

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

Static libraries

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

Types of libraries

Source libraries

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

Static libraries

- Provided as an *archive* of pre-compiled object code
 - Files are named `lib<library name>.a` e.g. `libcoolthing.a`
 - `.a` stands for "archive"
- Executable is linked against this library and the library objects are tossed into the executable

Types of libraries

Source libraries

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

Static libraries

- Provided as an *archive* of pre-compiled object code
 - Files are named **lib<library name>.a** e.g. **libcoolthing.a**
 - **.a** stands for "archive"
- Executable is linked against this library and the library objects are tossed into the executable
- "Static" because the library is a part of final 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

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
- Executable is linked against this library and the library 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)

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
- Executable is linked against this library and the library 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)
- "Dynamic" because these links and dependencies are resolved at 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**)
 - Those of you who used the ADV7.1 example project might've run into this...
- 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`

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`
 - `-lm` for `libm.a` and `libm.so`
 - `-lpng` for `libpng.a` and `libpng.so`

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`
 - `-lm` for `libm.a` and `libm.so`
 - `-lpng` for `libpng.a` and `libpng.so`
- Examples
 - `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`

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`
 - What if I want to link against `libm.a` and not `libm.so`?

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`
 - What if I want to link against `libm.a` and not `libm.so`?
- `-l` has `:<filename>` where you can specify a full filename, extension and all
 - e.g. `-l:libm.a`

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`
 - What if I want to link against `libm.a` and not `libm.so`?
- `-l` has `:<filename>` where you can specify a full filename, extension and all
 - e.g. `-l:libm.a`
- `gcc` also has the `-static` flag
 - 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`

Creating your own libraries

Static libraries

1. Compile the objects

- `gcc -c -o somecode.o somecode.c`
- `-c`: compile but don't link, produces an object code file

Creating your own libraries

Static libraries

1. Compile the objects

- `gcc -c -o somecode.o somecode.c`
- `-c`: compile but don't link, produces an object code file

2. Archive the objects

- `ar rcs libmylib.a somecode.o morecode.o yaycode.o`

Creating your own libraries

Static libraries

1. Compile the objects

- `gcc -c -o somecode.o somecode.c`
- `-c`: compile but don't link, produces an object code file

2. Archive the objects

- `ar rcs libmylib.a somecode.o morecode.o yaycode.o`
- `ar` is an archival tool
- `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`

Creating your own libraries

Dynamic libraries

1. Compile the objects

- `gcc -c -fPIC -o somecode.o somecode.c`
- `-fPIC`: compile as **p**osition **i**ndependent **c**ode

Creating your own libraries

Dynamic libraries

1. Compile the objects

- `gcc -c -fPIC -o somecode.o somecode.c`
- `-fPIC`: compile as **p**osition **i**ndependent **c**ode
- (there's also `-fpic`... if you want to go down the rabbit-hole)

Creating your own libraries

Dynamic libraries

1. Compile the objects

- `gcc -c -fPIC -o somecode.o somecode.c`
- `-fPIC`: compile as **p**osition **i**ndependent **c**ode
- (there's also `-fpic`... if you want to go down the rabbit-hole)
- The implications and reasoning behind PIC are best left for EECS 370 and EECS 482

Creating your own libraries

Dynamic libraries

1. Compile the objects

- `gcc -c -fPIC -o somecode.o somecode.c`
- `-fPIC`: compile as **p**osition **i**ndependent **c**ode
- (there's also `-fpic`... if you want to go down the rabbit-hole)
- The implications and reasoning behind PIC are best left for EECS 370 and EECS 482

2. Link your the objects

- `gcc -shared -fPIC -o libmylib.so somecode.o morecode.o yaycode.o`
- `-shared`: "produce a shared object"

Creating your own libraries

Dynamic libraries

1. Compile the objects

- `gcc -c -fPIC -o somecode.o somecode.c`
- `-fPIC`: compile as **p**osition **i**ndependent **c**ode
- (there's also `-fpic`... if you want to go down the rabbit-hole)
- The implications and reasoning behind PIC are best left for EECS 370 and EECS 482

2. Link your the objects

- `gcc -shared -fPIC -o libmylib.so somecode.o morecode.o yaycode.o`
- `-shared`: "produce a shared object"

3. It gets deeper... <http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html>

- Versioning (*soname* fun)
- Maintaining binary compatibility

Creating your own libraries

Dynamic libraries

1. Compile the objects

- `gcc -c -fPIC -o somecode.o somecode.c`
- `-fPIC`: compile as **p**osition **i**ndependent **c**ode
- (there's also `-fpic`... if you want to go down the rabbit-hole)
- The implications and reasoning behind PIC are best left for EECS 370 and EECS 482

2. Link your the objects

- `gcc -shared -fPIC -o libmylib.so somecode.o morecode.o yaycode.o`
- `-shared`: "produce a shared object"

3. It gets deeper... <http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html>

- Versioning (*soname* fun)
- Maintaining binary compatibility

4. Even deeper... <https://www.akkadia.org/drepper/dsohowto.pdf>

- Really great read
- Recommended by my interviewer during the interview for an internship

Hacking time

Questions?