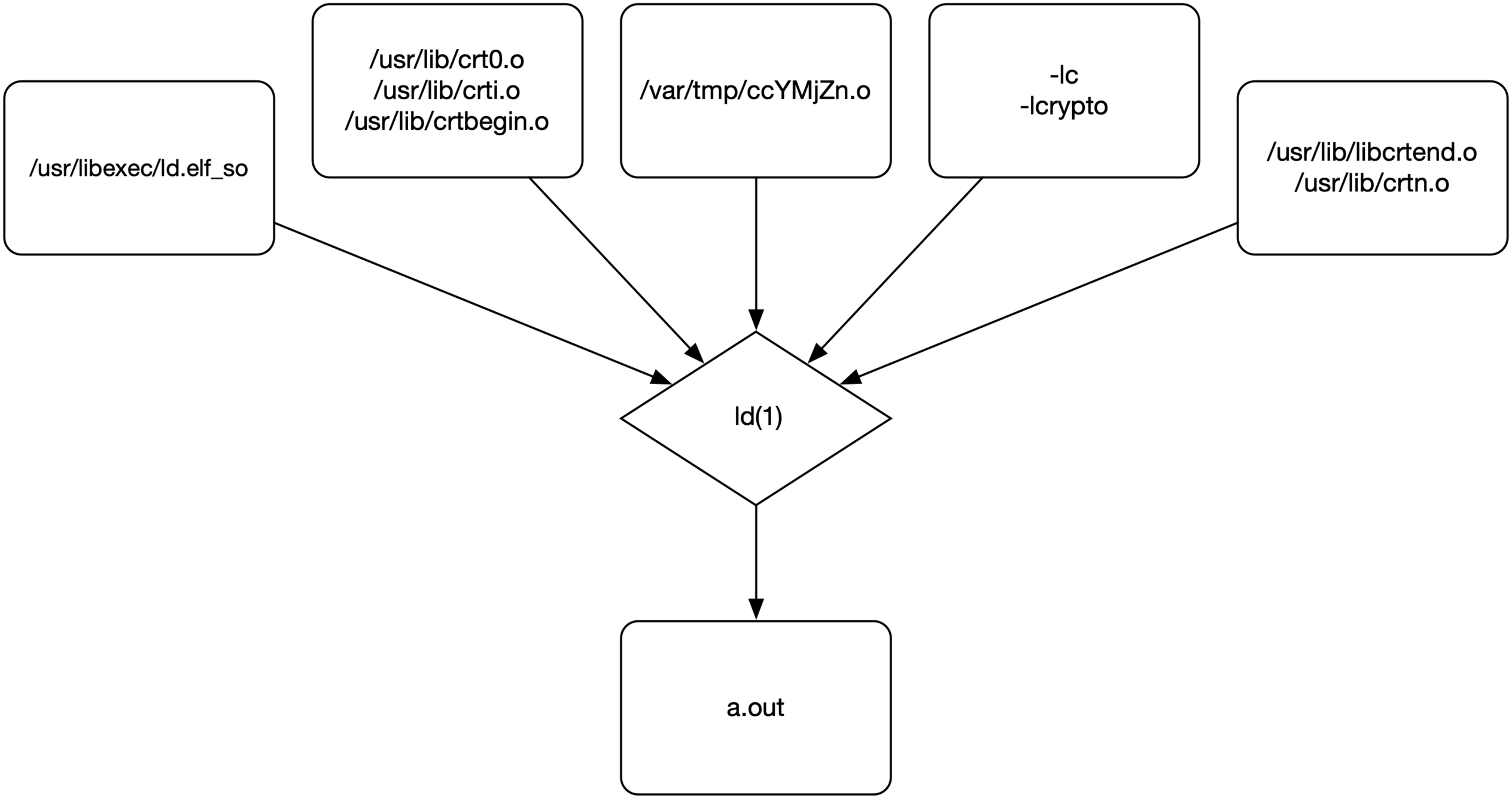
Advanced Programming in the UNIX Environment

Week 11, Segment 3:
Shared Libraries

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/


/usr/libexec/ld.elf_so
/usr/lib/crt0.o
/usr/lib/crti.o
/usr/lib/crtbegin.o
/var/tmp/ccYMjZn.o
-lc
-lcrypto
/usr/lib/libcrtend.o
/usr/lib/crtn.o
ld(1)
a.out

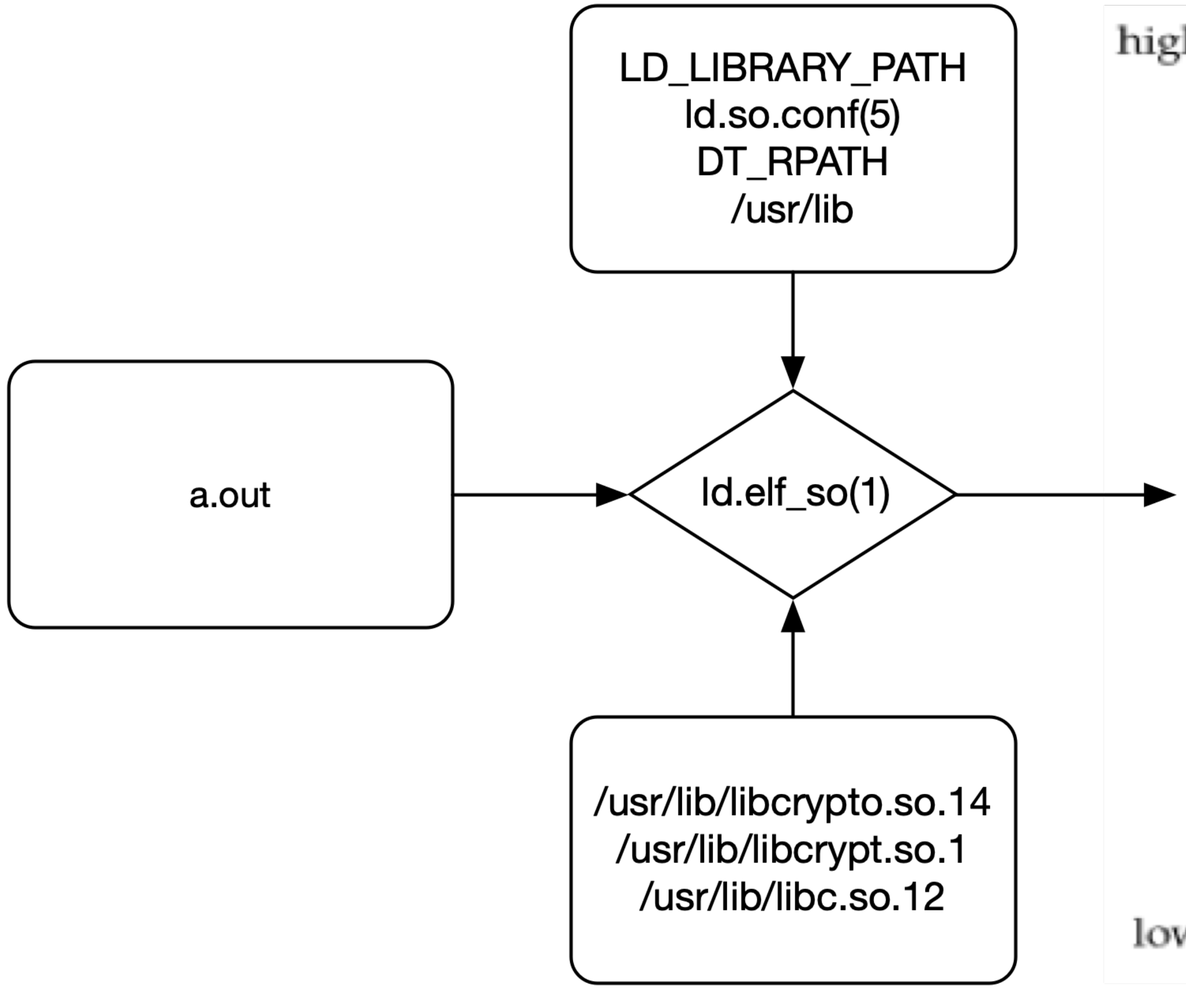
Shared Libraries

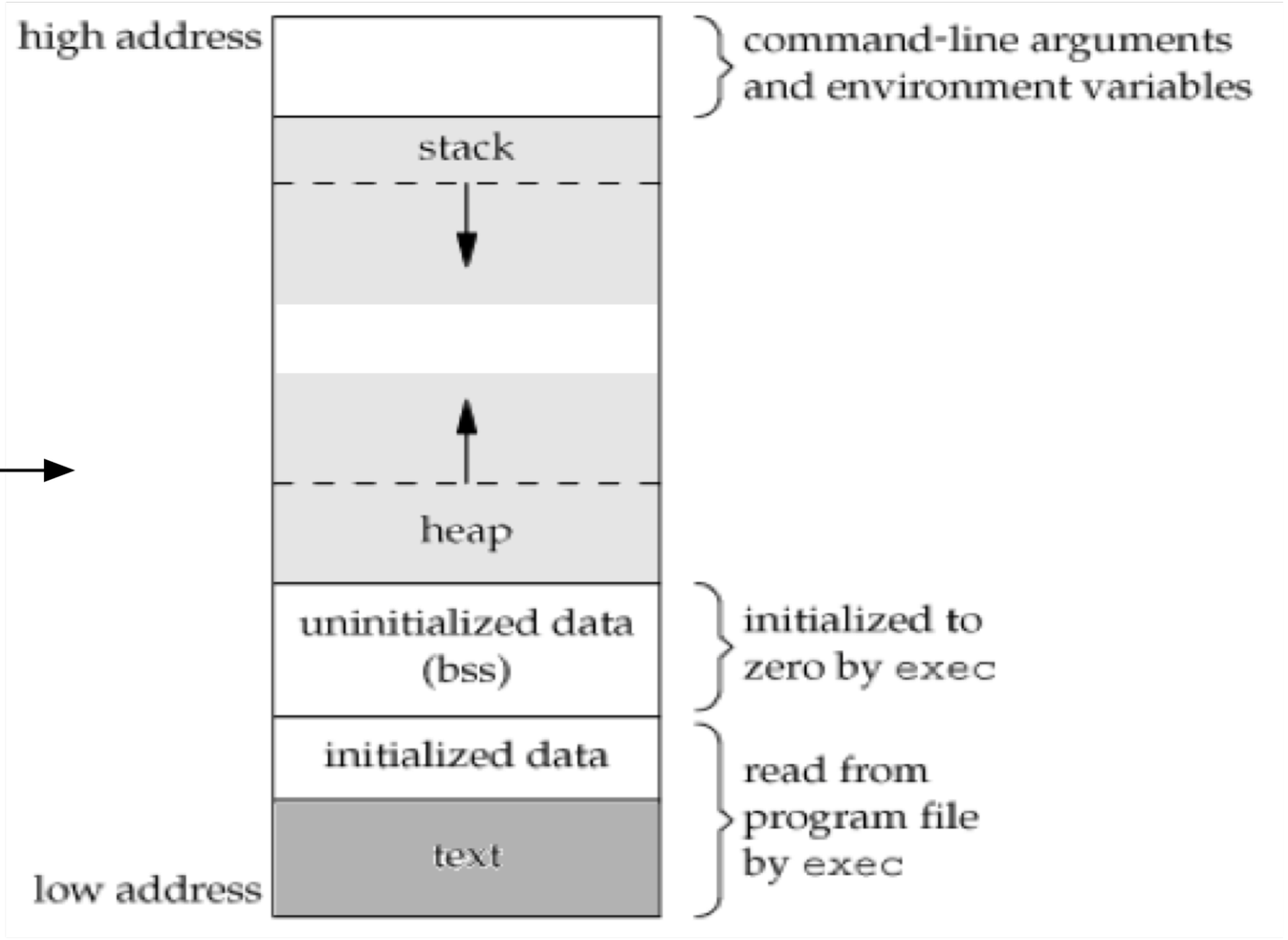
What is a shared library, anyway?

- contains a set of callable C functions (*i.e.*, implementation of function prototypes defined in `.h` header files)
- code is position-independent (*i.e.*, code can be executed anywhere in memory)
- libraries may be *static* or *dynamic*
- dynamically shared libraries can be loaded/unloaded at execution time or at will

Terminal — 80×24

```
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.12]
 0x0000000000000001 (NEEDED)             Shared library: [libcrypt.so.1]
 0x000000000000000c (INIT)               0x4004c0
 0x000000000000000d (FINI)               0x4008f0
 0x0000000000000004 (HASH)               0x4001d0
 0x0000000000000005 (STRTAB)             0x400388
 0x0000000000000006 (SYMTAB)             0x400220
 0x000000000000000a (STRSZ)              117 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x600ba0
 0x0000000000000002 (PLTRELSZ)           168 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x400418
 0x0000000000000007 (RELA)               0x400400
 0x0000000000000008 (RELASZ)             24 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x0000000000000000 (NULL)               0x0
jschauma@apue$ ./a.out
Usage: ./a.out string
jschauma@apue$ ./a.out foo
$1$$n1rTiFE0nRifwV/43bVon/
jschauma@apue$
```


LD_LIBRARY_PATH
ld.so.conf(5)
DT_RPATH
/usr/lib
a.out
ld.elf_so(1)
/usr/lib/libcrypto.so.14
/usr/lib/libcrypt.so.1
/usr/lib/libc.so.12


high address
command-line arguments
and environment variables
stack
heap
uninitialized data
(bss)
initialized to
zero by exec
initialized data
text
read from
program file
by exec
low address

Shared Libraries

How do shared libraries work?

- at *link time*, the linker resolves undefined symbols
- contents of object files and *static libraries* are pulled into the executable at link time
- contents of *dynamic libraries* are used to resolve symbols at **link time**, but loaded at **execution time** by the *dynamic linker*
- contents of *dynamic libraries* may be loaded **at any** time via explicit calls to the dynamic linking loader interface functions


Terminal — 80×24
000000000041f33b T tolower
000000000041f34e T tolower_l
000000000041f311 T toupper
000000000041f324 T toupper_l
00000000004782d0 W usleep
000000000406502 T valloc
00000000041419f T vfprintf
00000000004141e1 W vfprintf_l
000000000040e6b0 W vsnprintf
000000000040e577 W vsnprintf_l
000000000476750 W vsnprintf_ss
00000000414666 T wcrtomb
00000000004145b8 T wcrtomb_l
00000000041472e T wcsrtombs
00000000414672 T wcsrtombs_l
00000000004147b8 T wctob
00000000041477c T wctob_l
000000000041f9a0 W write
000000000478420 W writev
jschauma@apue$ nm a.out.dyn | wc -l
36
jschauma@apue$ nm a.out.static | wc -l
1075
jschauma@apue$

Statically Linked Shared Libraries

Static libraries:

- created using `ar(1)`
- usually end in `.a`
- effectively a single file containing other (object) files
- linking statically pulls in all the code from the archives into the executable

```
jschauma@apue$ sudo chmod u+s a.out
jschauma@apue$ ls -l a.out
-rwsr-xr-x  1 root  users  8432 Nov 14 22:49 a.out
jschauma@apue$ ./a.out
ldtest0 => Hello world!
ldtest1 => Hello world!
ldtest2 => Hello world!
jschauma@apue$ echo $LD_LIBRARY_PATH
./lib2
jschauma@apue$ sudo chmod u-s ./a.out
jschauma@apue$ ./a.out
Muahaha, I can do anything now!
Including... unlink(whatever)
And to have nobody notice, I'll also do what's expected.
ldtest0 => Hello world!
Muahaha, I can do anything now!
Including... unlink(whatever)
And to have nobody notice, I'll also do what's expected.
alternate ldtest1 implementation => Hello world!
Muahaha, I can do anything now!
Including... unlink(whatever)
And to have nobody notice, I'll also do what's expected.
ldtest2 => Hello world!
jschauma@apue$
```

Dynamically Linked Shared Libraries

Dynamic libraries:

- require object files to be compiled into Position Independent Code (PIC)
- usually end in .so
- frequently have multiple levels of symlinks providing backwards compatibility / ABI definitions
- symbols are resolved at link time, but require the libraries to be found at execution time
- system- and user-specific configuration may influence resolution


Terminal — 80×24
0000000000601280 B __sF
0000000004007a0 T __start
0000000000601262 D _edata
0000000000601488 B _end
0000000000400c60 T _fini
0000000004006c0 T _init
U _libc_init
0000000004007a0 T _start
U abort
U atexit
U dlerror
U dlopen
U dlsym
0000000000601480 B environ
U err
U exit
U fprintf
0000000000400c02 T main
0000000000400b5a T printCrypt
U puts
jschauma@apue$ nm a.out | grep crypt
jschauma@apue$ ./a.out foo
$1$$n1rTiFE0nRifwV/43bVon/
jschauma@apue$

Summary and Exercises

- Static libraries let you build statically linked executables containing all the code needed to run the program.
- Dynamic libraries let you define which code should be pulled in at execution time.
- The behavior of dynamically linked executables can be influenced by changing the dynamic library without requiring the executable to be re-compiled or re-linked.

- We saw the use of the `LD_LIBRARY_PATH` environment variable; how does it compare to the `LD_PRELOAD` variable? What valid use cases are there for either, and how could either lead to security problems?
- Some systems / link-loaders support the `LD_DEBUG` environment variable; play around with the different values.
- For more details, see `link(5)` and play with `readelf(1)`, `objdump(1)`, and `nm(1)`.

Links

- Implement a trivial library yourself: https://stevens.netmeister.org/631/libgreet-exercise.html
- https://en.wikipedia.org/wiki/Position-independent_code
- Linkers and Loaders: https://is.gd/flULDM
- libelf by Example: https://is.gd/v45CcV
- Dynamic linker tricks: https://is.gd/VGDTTD