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/
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
<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Name/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libc.so.12]</td>
</tr>
<tr>
<td>0x0000000000000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libcrypt.so.1]</td>
</tr>
<tr>
<td>0x000000000000000c</td>
<td>(INIT)</td>
<td>0x4004c0</td>
</tr>
<tr>
<td>0x000000000000000d</td>
<td>(FINI)</td>
<td>0x4008f0</td>
</tr>
<tr>
<td>0x0000000000000004</td>
<td>(HASH)</td>
<td>0x4001d0</td>
</tr>
<tr>
<td>0x0000000000000005</td>
<td>(STRTAB)</td>
<td>0x400388</td>
</tr>
<tr>
<td>0x0000000000000006</td>
<td>(SYMTAB)</td>
<td>0x400220</td>
</tr>
<tr>
<td>0x000000000000000a</td>
<td>(STRSZ)</td>
<td>117 (bytes)</td>
</tr>
<tr>
<td>0x000000000000000b</td>
<td>(SYMEXT)</td>
<td>24  (bytes)</td>
</tr>
<tr>
<td>0x0000000000000015</td>
<td>(DEBUG)</td>
<td>0x0</td>
</tr>
<tr>
<td>0x0000000000000003</td>
<td>(PLT/GOT)</td>
<td>0x600ba0</td>
</tr>
<tr>
<td>0x0000000000000002</td>
<td>(PLTRELSZ)</td>
<td>168 (bytes)</td>
</tr>
<tr>
<td>0x0000000000000014</td>
<td>(PLTREL)</td>
<td>RELA</td>
</tr>
<tr>
<td>0x0000000000000017</td>
<td>(JMPREL)</td>
<td>0x400418</td>
</tr>
<tr>
<td>0x0000000000000007</td>
<td>(RELA)</td>
<td>0x400400</td>
</tr>
<tr>
<td>0x0000000000000008</td>
<td>(RELSZ)</td>
<td>24  (bytes)</td>
</tr>
<tr>
<td>0x0000000000000009</td>
<td>(RELAENT)</td>
<td>24  (bytes)</td>
</tr>
<tr>
<td>0x0000000000000000</td>
<td>(NULL)</td>
<td>0x0</td>
</tr>
</tbody>
</table>

jshauma@apue$ ./a.out
Usage: ./a.out string
jshauma@apue$ ./a.out foo
$1$$n1rTlFE0nRifwV/43bVon/
jshauma@apue$
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
0000000000028630 T tolower
000000000002863e T tolower_l
000000000002863c T toupper
0000000000028644 T toupper_l
000000000002864c W usleep
0000000000028660 W valloc
000000000002866c T vfprintf
0000000000028674 T vfprintf_l
0000000000028678 W vsnprintf
000000000002867e W vsnprintf_l
000000000002868c W vsnprintf_ss
0000000000028690 T wcrtomb
0000000000028698 T wcrtomb_l
00000000000286a0 T wcsrtombs
00000000000286a8 T wcsrtombs_l
00000000000286b0 T wctob
00000000000286b8 T wctob_l
00000000000286c0 W write
00000000000286c8 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
sudo chmod u+s a.out
ls -l a.out
-rw-r-xr-x 1 root users 8432 Nov 14 22:49 a.out
./a.out
ldtest0 => Hello world!
ldtest1 => Hello world!
ldtest2 => Hello world!

echo $LD_LIBRARY_PATH
./lib2

sudo chmod u-s ./a.out
./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.
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
B __sF
T __start
D _edata
B __end
T _fini
T _init
U _libc_init
T _start
U abort
U atexit
U dlerror
U dlopen
U dlsym
B environ
U err
U exit
U fprintf
T main
T printCrypt
U puts

jschauma@apue$ nm a.out | grep crypt
jschauma@apue$ ./a.out foo
$1$sn1rTiFE0nRifwV/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
- Linkers and Loaders: https://is.gd/flULDM
- libelf by Example: https://is.gd/v45CcV
- Dynamic linker tricks: https://is.gd/VGDTTD