CS61 Section Notes

Week 8 (Fall 2011)

Linking

`gcc` is a *compiler driver* that invokes the actual compiler, linker, etc. as needed on behalf of the user. Consequently, there is a lot of mysterious stuff going on in the background that many programmers aren't aware of. Fortunately, you're taking CS61!

1. The Unix *Executable and Linkable Format* (ELF) is a common and representative object file format. Take a few minutes to brainstorm; what are the components of a relocatable ELF object?

2. Now, consider the following C code...

`main.c`

```
void swap();

int buf[2] = {1, 2};

int main() {
    swap();
    return 0;
}
```

`swap.c`

```
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;

void swap() {
    int temp;

    bufp1 = &buf[1];
    temp = *bufp0;
    *bufp0 = *bufp1;
    *bufp1 = temp;
}
```

For each of the following symbols in `swap.o.symtab`, fill out the symbol type (local, global, or extern) and the module (`main.o` or `swap.o`) and ELF section where it is defined.

Symbol	Type	Module	Section
`buf`			
`bufp0`			
`bufp1`			
`swap`			
`temp`			

3. Consider the following C code:

main.c	swap.c
`int *x = 0x8000000;`	`int x;`
`int y;`	`static int y;`
`char z = ‘a’;`	`static double z;`
`float f = 3.14;`	`double f = 3.1415`
`float w;`	`int w;`
`static int q;`	`static int q;`
`...`	`...`

In trying to link the object files of main and swap, to which version of each variable (x, y, z, f, w, and q), will the linker resolve (the swap version, or the main version)? Are there any that cannot be resolved?

Libraries

When two or more object files are linked together, the linker must...

- *Resolve symbols:* The linker must associate each symbol *reference* with a single *definition*, just as you did in the previous problem.
- *Relocate:* Once symbols have been resolved, the linker must update each symbol reference to point to the actual location of the symbol definition in memory. These placeholders are listed in the .rel section of each ELF object file.

The linker then combines the .text, .data, etc. of each object file to make a new object file. In a sense, the new object file contains all the contents of the old object files.

Sometimes, you want to be able to *package* many objects together (such as a bunch of related printing functions), but only *copy* the objects that you actually *reference*. In this case, what you want is a *library:* a collection of conditionally-linked objects.

4. Because libraries depend on references to link, the objects that reference them must be processed by the compiler driver before the library is processed. Let a -> b denote that a depends on b; that is, b defines a symbol that a references. For each question (a) through (c), in what order must the following be processed?

   a. p.o -> libx.a
   b. p.o -> libx.a -> liby.a
   c. p.o -> libx.a -> liby.a **and** liby.a -> libx.a -> p.o

Dynamic Linking

Up to this point we have discussed only *static linking*, which takes place at compile time and involves copying over the linked objects into a new executable. In contrast, *dynamic linking* against a *shared library* of code can take place at load or even run time.

5. Take a few minutes to brainstorm: what are the pros and cons of static versus dynamic linking?