CS 241 — Lecture 8

Bartosz Antczak *Instructor: Kevin Lanctot* *January 26, 2017*

Recall from Last Lecture

Assemblers translate assembly language to machine code.
Loaders track and adjust labels and load a program into memory. Loaders allow a program to be loaded anywhere into RAM.

8.1 Linkers

Linkers, by definition, break up a program into files of smaller sub-programs.

8.1.1 How to Link

Linking multiple files requires two parts:

The External Symbol Reference (ESR) Format

An `.import <symbol>` statement that tells the assembler that `symbol` occurs in another file (e.g., `.import notify_nsa` means that the symbol `notify_nsa` is defined in another file). Its structure looks like this:

word1:	0x11	(a constant that signifies that what follows is an ESR)
word2:	address	(where the symbol is used)
word3:	length	(how long the symbol is in bytes (say n))
word3:	1st char of symbol (in ASCII)	
	...	
word $n+3$:	last char of symbol in ASCII	

The External Symbol Definition (ESD) Format

An `.export <symbol>` statement that tells the assembler that other files may use this symbol. This statement means that the defined symbol has *global* scope. Its structure is identical to that of the ESR, except the constant used to signify that what follows is an ESD is *0x05*, rather than 0x11.
The ESR and ESD are stored in the third part of the MERL file.

8.1.2 Linker Pseudocode

The goal of a linker is to handle multiple files and external symbols. The linker follows four steps:

1. **Concatenate the programs** (Visual example shown on next page)

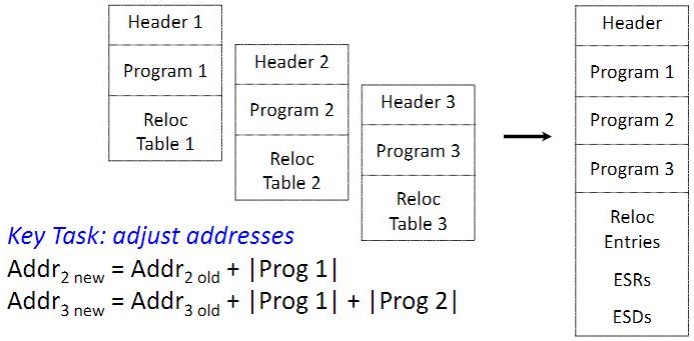

Figure 8.1: Concatenating three files. Courtesy of Prof. Lanctot's slides.

2. **Combine and Adjust ESD's**
   For the ESD's for program 2, 3, and so on, we must shift them down by the size of all the programs before it (i.e., program 2's ESD must be shifted down by the size of program 1)

3. **Use new ESD's to update old ESR's** For each old ESR, ff a new ESD exists, update the value at the location + the offset; otherwise adjust the new ESR with the new offset (e.g., $\text{ESR}_{2\,\text{new}} = \text{ESR}_{2\,\text{old}} + |\text{Prog 1}|$).

4. **Relocate addresses** Relocate addresses by shifting them based on the offset (e.g., $\text{Addr}_{2\,\text{new}} = \text{Addr}_{2\,\text{old}} + |\text{Prog 1}|$).

Dynamic Linking

What we defined previously is *static linking* (i.e., all of the files are linked before the program is laoded). Another approach is to use **dynamic linking**, which relocates symbols while a program is running. For instance, if you want to import a new dictionary in Microsoft Word, say one in another language, you would use dynamic linking to load that dictionary file into the running program.