Jackson Myers
03/01/2023
Computer Architecture
Dr. Oh

1. A linker merges different pieces of code into one executable. The linker works by resolving symbols and their respective types in files that will be linked. It will use these symbols to determine where to store data and how to organize it so that the program can be properly executed from disk. This process is called symbol relocation. All of this allows the linker to create an executable from multiple files. This allows programs to be modular, meaning programmers can work on different files from the same program, at the same time. This also means that external libraries can be used in your program, which can save space. Possibly the largest benefit beyond modularity is that you only need to recompile files that have changed, then you can re-link the files and everything should work.

2. A relocatable object is a file that contains data that can be combined with other relocatable object files. The relocatable object contains similar data that an executable would in regards to ELF, fields such as (.text, .data, .bss, etc.). An executable object also contains all of the data (.text, .data, .rodata, etc). However, an executable is already "linked" and can properly load a program's data into memory and execute it.

   Differences: relocatable objects are not executable, they are meant to be "combined" by the linker so that they can become executable. Executables are files that are stored in disk, that already contain the linked relocatable objects. In the ELF, it contains data from the relocatable objects that were used to make it, except it can actually be executed by the computer.

3. Benefits of static and dynamic libraries:

   Static:
   a. Functions can be organized into separate files, which is more time efficient.
   b. Libraries are available at compile-time, so performance is superior to dynamically linked libraries.

   Dynamic:
   a. Can be dynamically loaded at run-time.
   b. Libraries can be shared in memory by different processes.

4. SYMBOL TYPES AND SECTIONS

| | | |
|---|---|---|
| buf: | global, | .data |
| value: | external global, | .bss |
| bufp0: | local, | .bss |
| bufp1: | global, | .bss |
| decrease: | local, | .text |
| cnt: | local, | .data |
| swap: | global, | .text |

5.

5.1: The problem is that we have two conflicting strong symbols: the variable int func = 2; and the function func(){}. Since both of these symbols are strong, the linker cannot resolve them. To fix this, the best way would be to change the name of either the variable or the symbol.

5.2: The problem now is that we have a linker warning because the symbol “a” is conflicting in both of the files. The linker will choose the int sized definition from a.c because it’s strong, and b.c uses ‘a’ as a weak double. This could break b.c’s functionality and create unpredictable output from the program because the defined size of the variable is incorrect for b.c.