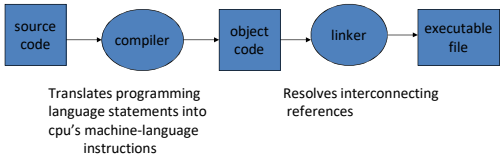
CS35L-5

Week 8 Lec1

Building an executable file

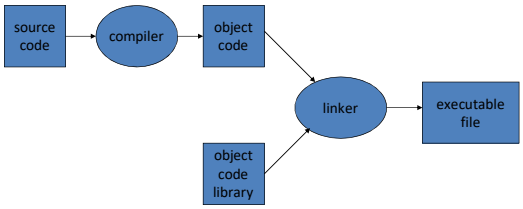


Translates programming language statements into cpu's machine-language instructions

Resolves interconnecting references

Linking and Loading

- Program life cycle:
  write -> compile -> link -> load -> execute
- Linker collects procedures and links them together - multiple object files into one executable
- Loading: This refers to copying a program image from hard disk to the main memory in order to put the program in a ready-to-run state.



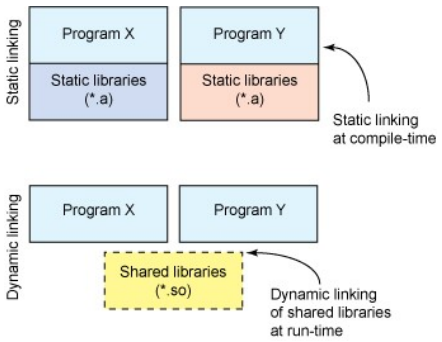
A previously compiled collection of standard program functions

Static Linking

- Carried out only once to produce an executable file
- If static libraries are called, the linker will copy all the modules referenced by the program to the executable
- Static libraries are typically denoted by the .a file extension and created using the ar (archiver) program

Dynamic Linking

- Allows a process to add, remove, replace or relocate object modules during its execution.
- If shared libraries are called:
  - Only copy a little reference information when the executable file is created
  - Complete the linking during loading time or running time
- Dynamic libraries are typically denoted by the .so file extension
  - .dll on Windows



Dynamic linking

- Unix systems: Code is typically compiled as a *dynamic shared object* (DSO)
- Dynamic vs. static linking resulting size

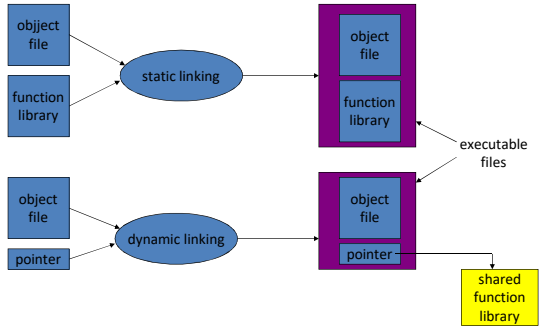
```
$ gcc -static hello.c -o hello-static
$ gcc hello.c -o hello-dynamic
$ ls -l hello
     80 hello.c
  13724 hello-dynamic
    383 hello.s
1688756 hello-static
```

- If you are the sysadmin, which do you prefer?

Advantages of dynamic linking

- The executable is typically smaller since not entire external program is in the executable.
- When the library is changed, the code that references it does not usually need to be recompiled.
- The executable accesses the .so at run time; therefore, multiple programs can access the same .so at the same time
  - Memory footprint amortized across all programs using the same .so
  - Load time reduced. (Static - takes constant load time)

Smaller is more efficient



Disadvantages of dynamic linking

- Performance hit
  - Need to load shared objects (at least once)
  - Need to resolve addresses (once or every time)
- What if the necessary dynamic library is missing?
- What if we have the library, but it is the wrong version?

How are libraries dynamically loaded?

Table 1. The DI API

Function	Description
dlopen	Makes an object file accessible to a program
dlsym	Obtains the address of a symbol within a `dlopen`ed object file
dlerror	Returns a string error of the last error that occurred
dlclose	Closes an object file

In all,

- Static Libraries: installed into executable before program can be run
- Shared Libraries: loaded at program start-up (if not already) and shared between programs
- Dynamically Loaded Libraries: loaded and used at any time while a program is running

Lab 8

- Write and build simple cos(sqrt(3.0)) program in C
  - Use `ldd` to investigate which dynamic libraries your hello world program loads
  - Use `strace` to investigate which system calls your hello world program makes
- Use "`ls /usr/bin | awk 'NR%101==SID%101'`" to find ~25 linux commands to use `ldd` on
  - Record output for each one in your log and investigate any errors you might see
  - From all dynamic libraries you find, create a sorted list
    - Remember to remove the duplicates!