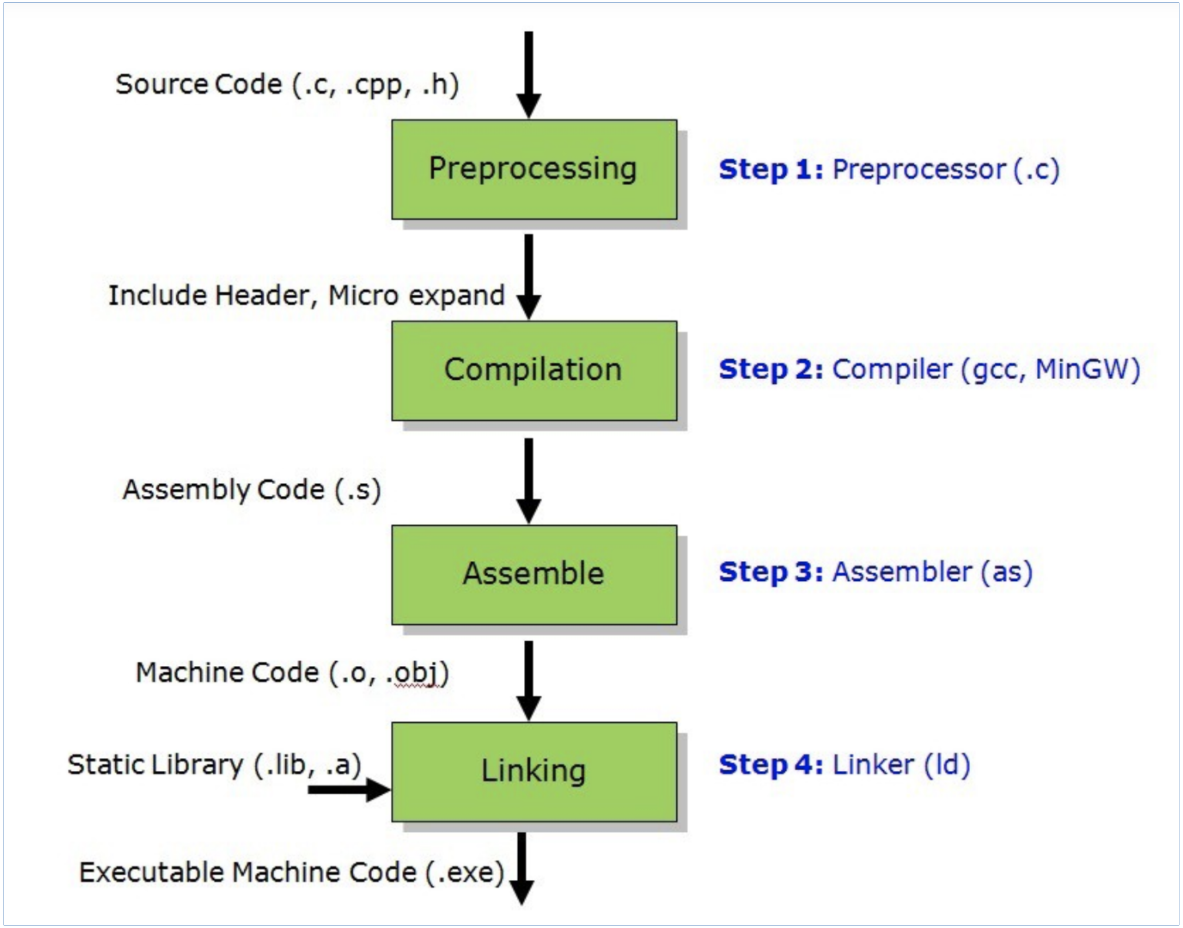
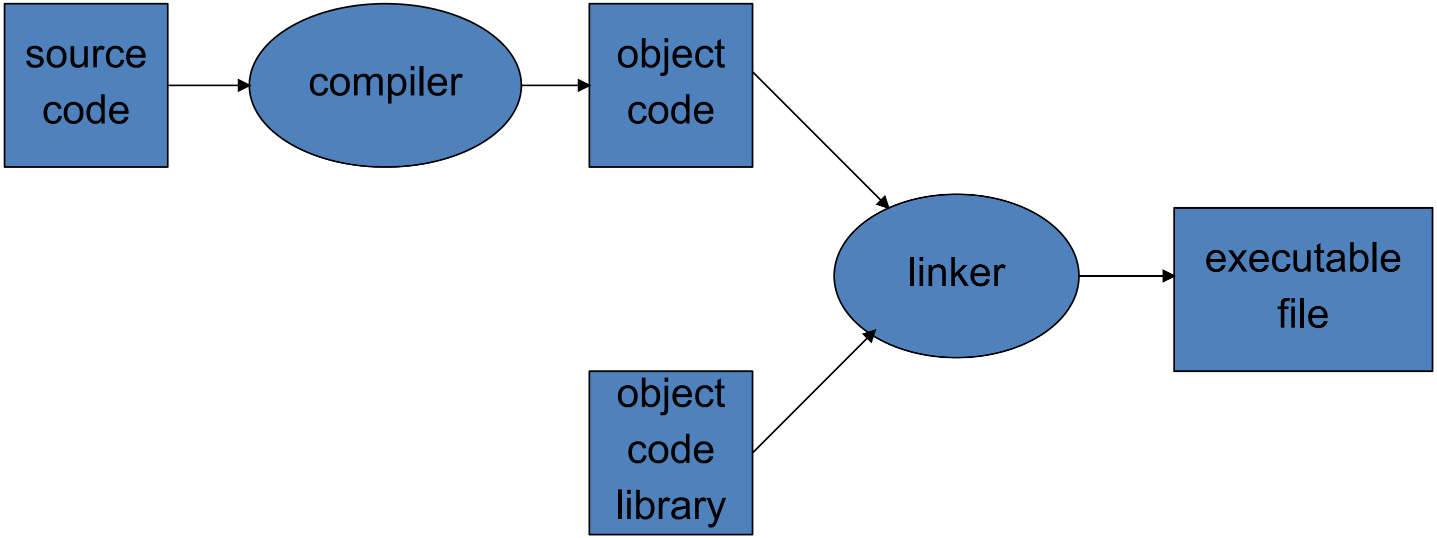
CS35L Software Construction Laboratory

Lab 1: Nandan Parikh

Week 7; Lecture 1


Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (.c)
Include Header, Micro expand
Compilation
Step 2: Compiler (gcc, MinGW)
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

http://binaryupdates.com/introduction-of-c/



source code

compiler

object code

linker

executable file

object code library

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

Dynamic Linking

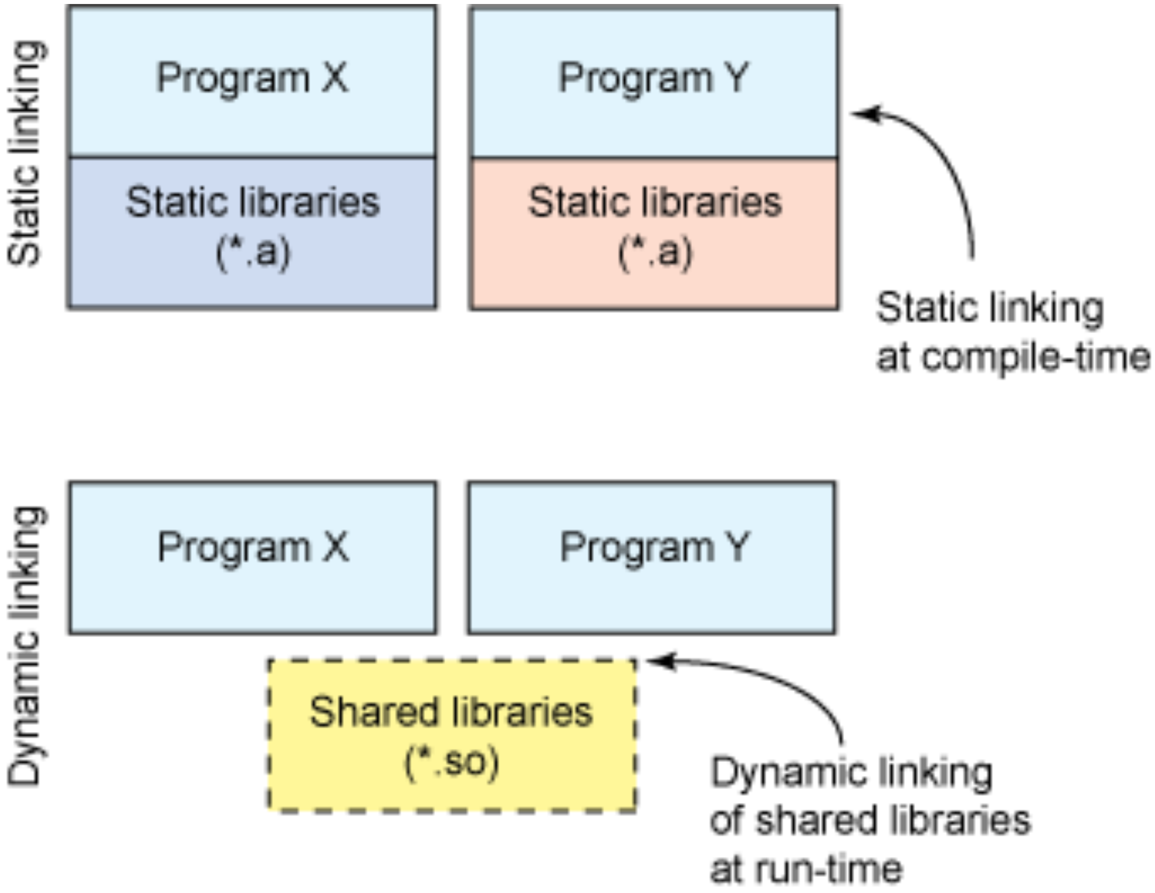
- Allows a process to add, remove, replace or

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

Linking and Loading

- Linker collects procedures and links together the object modules into one executable program
- Why isn't everything written as just one big program, saving the necessity of linking?
  - Efficiency: if just one function is changed in a 100K line program, why recompile the whole program? Just recompile the one function and relink.
  - Multiple-language programs
  - Other reasons?


Static linking
Program X
Static libraries
(*.a)
Program Y
Static libraries
(*.a)
Static linking
at compile-time
Dynamic linking
Program X
Program Y
Shared libraries
(*.so)
Dynamic linking
of shared libraries
at run-time

Dynamic linking

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

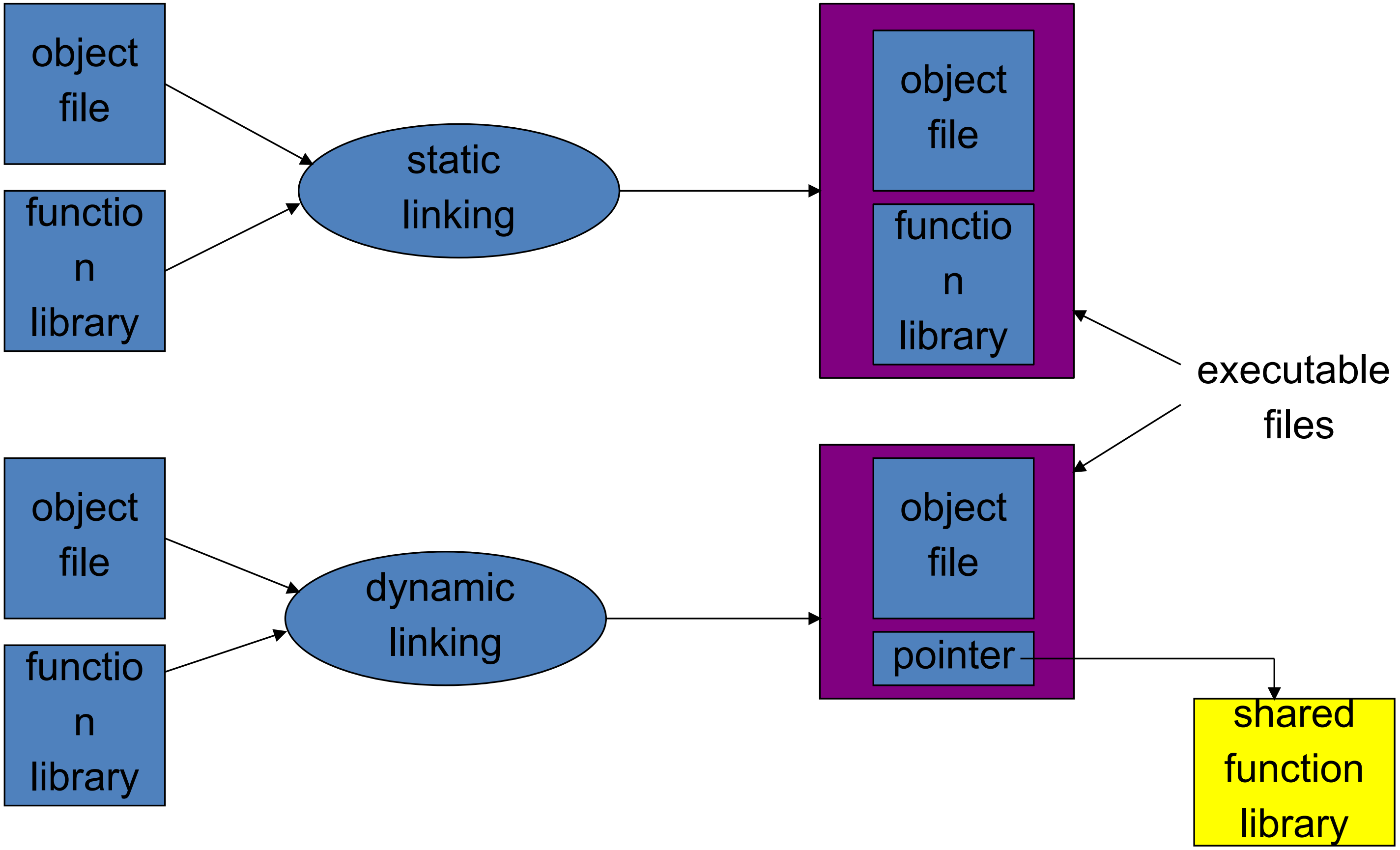
  ```
  $ gcc -static hello.c -o hello-static
  $ gcc hello.c -o hello-dynamic
  $ ls -l hello
       80 hello.c
    13724 hello-dynamic
  1688756 hello-static
  ```

- Pros and cons?

Advantages of dynamic linking

- The executable is typically smaller
- 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

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)
  - Remember back to the system call assignment…
- What if the necessary dynamic library is missing?
- What if we have the library, but it is the wrong version?

Useful Links for creating static/dynamic libraries

- http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html

- http://tldp.org/HOWTO/Program-Library-HOWTO/index.html

- https://www.ibm.com/developerworks/library/l-dynamic-libraries/

For getting started with concepts :

- https://medium.com/@tyastropheus/s-is-for-static-the-abcs-of-the-c-static-library-cdc4109c30a6

Lab 7

- Build the code simpgmp.c
  - Use ldd to investigate which dynamic libraries your program loads
  - Use strace to investigate which system calls your program makes
- Use “ls /usr/bin | awk ‘NR%101==nnnnnnnnn%101’” to find ~12 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 omit the duplicates!