CISC 3320 MW3

Linkers and Loaders

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

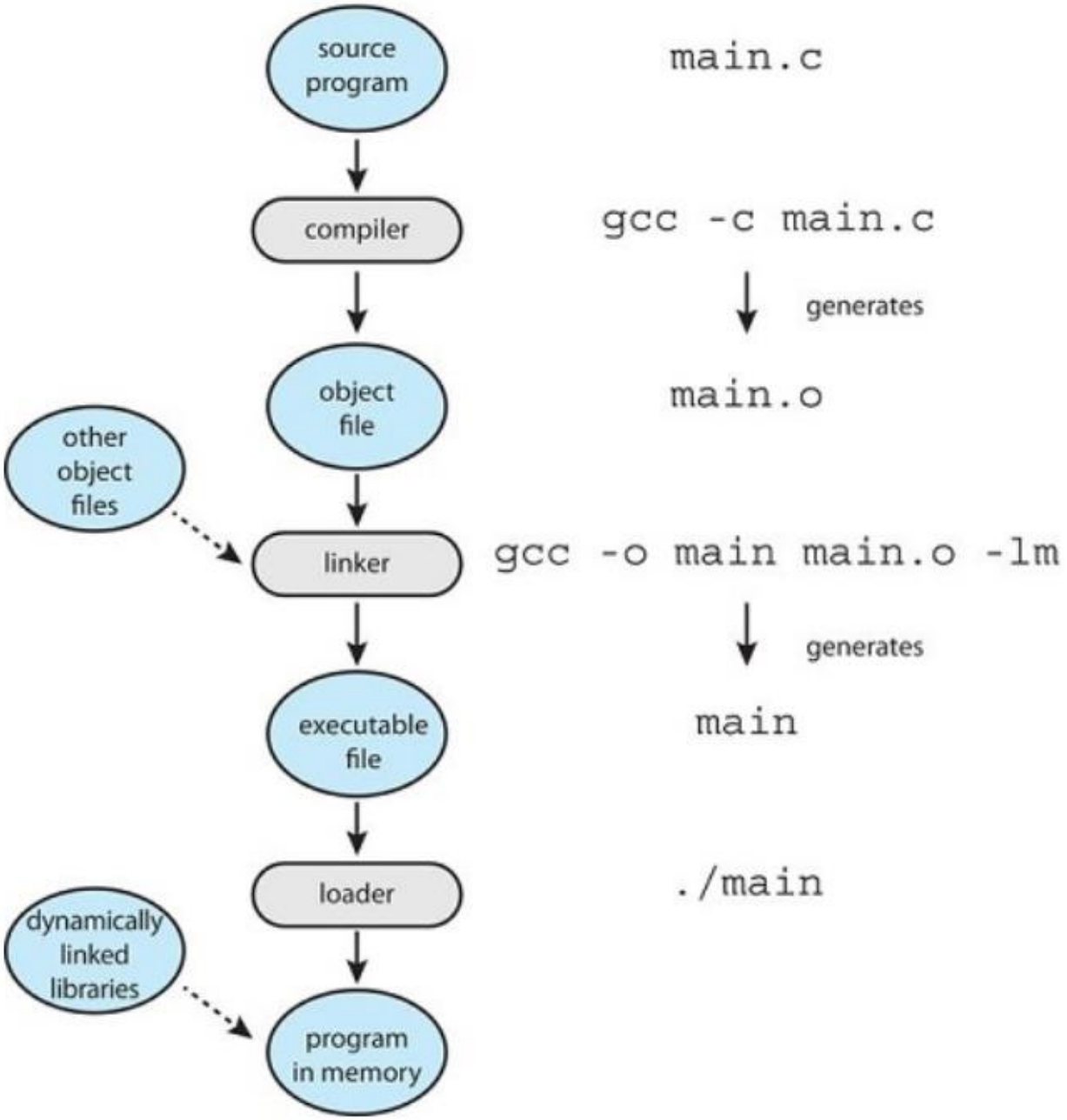
- These slides are a revision of the slides by the authors of the textbook

Outline

- Linkers and linking
- Loaders and loading
- Object and executable files

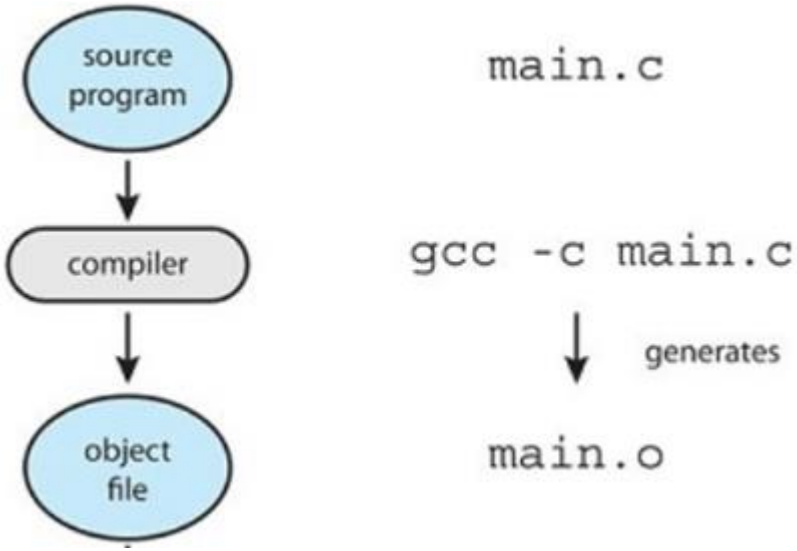
Authoring and Running a Program

- A programmer writes a program in a programming language (the source code)
- The program resides on disk as a binary executable file translated from the source code of the program
  - e.g., a.out or prog.exe
- To run the program on a CPU
  - the program must be brought into memory, and
  - create a process for it
- A multi-step process


source program
main.c
compiler
gcc -c main.c
generates
object file
main.o
other object files
linker
gcc -o main main.o -lm
generates
executable file
main
loader
./main
dynamically linked libraries
program in memory

Compilation

- Source files are compiled into object files
- The object files are designed to be loade into any physical memory location, a format known as an relocatable object file.



Relocatable Object File

- Object code: formatted machine code, but typically non-executable
- Many formats
  - The Executable and Linkable Format (ELF)
  - The Common Object File Format (COFF)

Examining an Object File

- In Linux/UNIX,

  $ file main.o

  main.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped

  $ nm main.o

- Also use objdump, readelf, elfutils, hexedit
- You may need

  # apt-get install hexedit

  # apt-get install elfutils

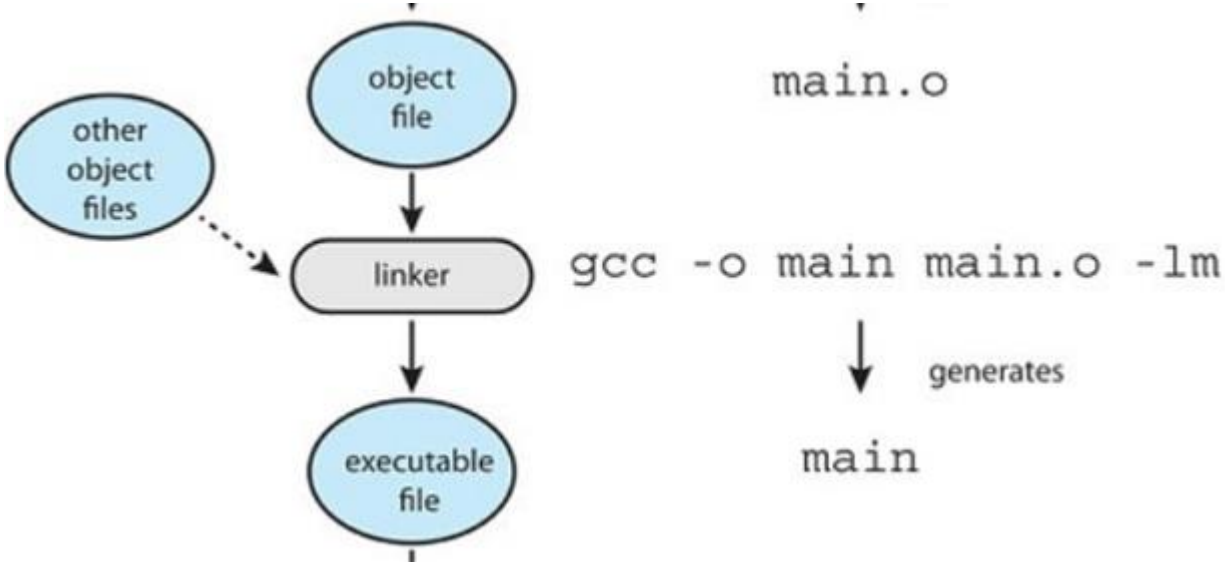
Linking

- During the linking phase, other object files or libraries may be included as well
  - Example:

    ```
    $ g++ -o main -lm main.o sumsine.o
    ```

  - A program consists of one or more object files.
  - Linker combines relocatable object files into a single binary executable file.

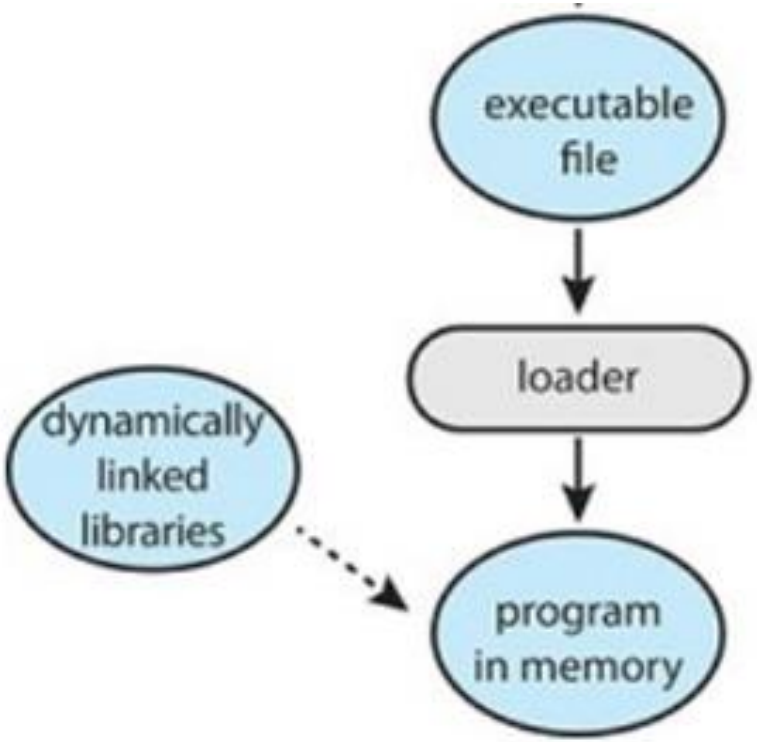
Linking: Example



Loader

- A loader is used to load the binary executable file into memory, where it is eligible to run on a CPU core.
- An activity associated with linking and loading is relocation
- It assigns final addresses to the program parts and adjusts code and data in the program to match those addresses
- So that, for example, the code can call library functions and access its variables as it executes.

Loading: Example




```
main

./main
```

Running Loader to Load a Program

- The loader may be invoked differently in different operating systems or operating systems shells
- Examples:
  - Enter the name of the execute file on the command line and hit the ENTER key.
  - Double-clicking on the icon associated with the executable file invokes the loader using a similar mechanism
- In Linux/UNIX,
  - The shell first creates a new process to run the program using the fork() system call.
  - The shell then invokes the loader with the exec() system call, passing exec() the name of the executable file.
  - The loader then loads the specified program into memory using the address space of the newly created process.

Library and Dynamically Link Library

- Library: a collection object code/files combined in a single formatted file
- Static library
  - Object code in the library are linked into the executable file during the linking process, becomes a part of the executable, and loaded into memory from the executable
- Dynamic link library

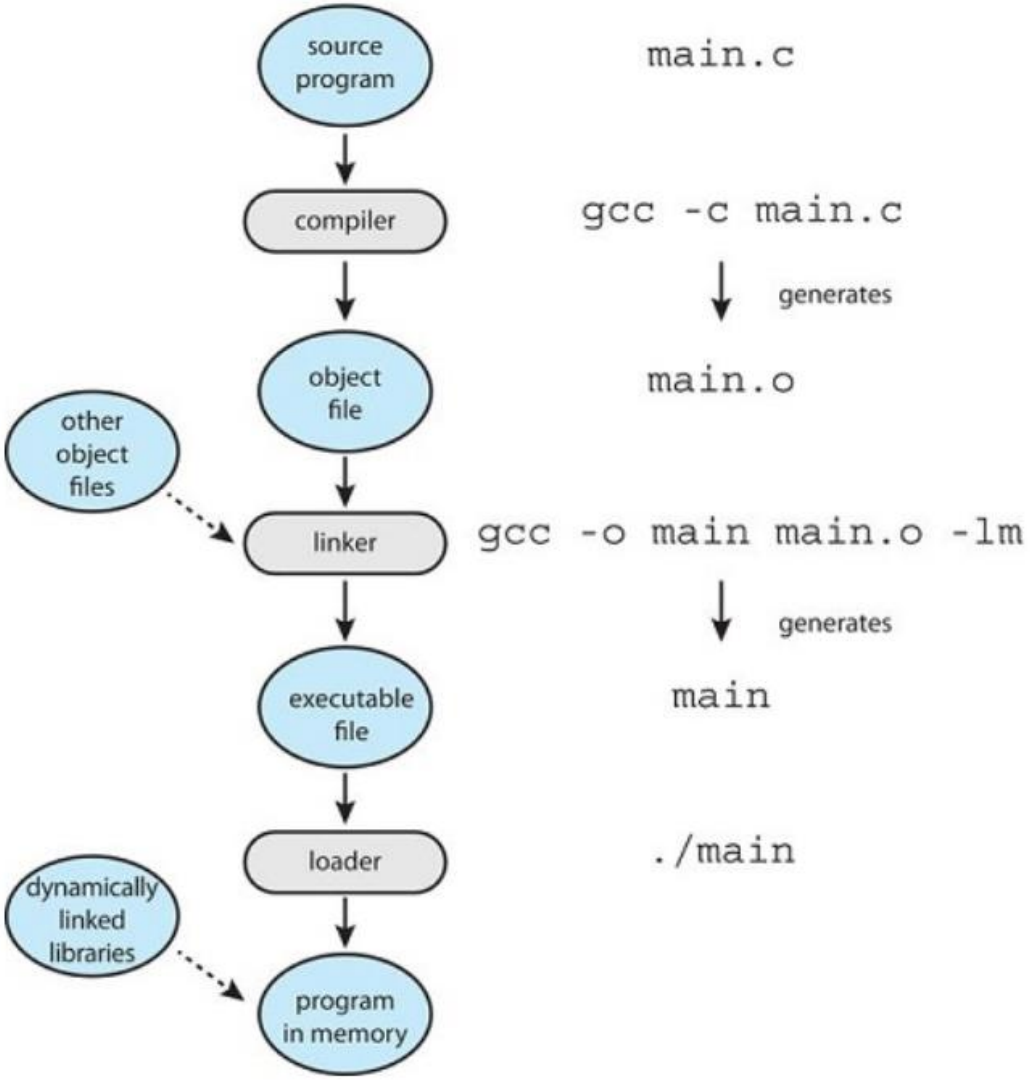
Dynamic Link Library

- Object code in the library are linked during the loading process, and the object code is not a part of the executable.
- Examples
  - Windows: DLLs (dynamically linked libraries)
  - Linux/UNIX: SOs (dynamically linked shared object libraries)
- Benefit: the library is conditionally linked and is loaded as required during program run time, which avoids linking and loading libraries that may end up not being used into an executable file.
  - For example, the math library is a Linux dynamically linked shared object library, as such is not linked into the executable file main.
  - In this case, the linker inserts relocation information that allows it to be dynamically linked and loaded as the program is loaded.
  - It is possible for multiple processes to share dynamically linked libraries, resulting in a significant savings in memory use.

Object File and Executable File

- Executable files, like object files, typically have standard formats
- Executable files and object files include
  - the compiled machine code, and
  - a symbol table containing metadata about functions and variables that are referenced in the program.
- Example formats
  - Linux/UNIX: the ELF format (for Executable and Linkable Format). There are separate ELF formats for relocatable and executable files.
    - The ELF file for executable files has the program's entry point, the address of the first instruction to be executed when the program runs.
  - Windows: the Portable Executable (PE) format
  - Mac OS X: the Mach-O format.

Questions?


source program
main.c
compiler
gcc -c main.c
generates
object file
main.o
other object files
linker
gcc -o main main.o -lm
generates
executable file
main
loader
./main
dynamically linked libraries
program in memory

Applications are System Specific

- Why?
  - Each operating system provides a unique set of system calls. System calls are part of the set of services provided by operating systems for use by applications.
- But, we do run some applications cross platforms
  - Chrome, Firefox, etc

Portable Applications

- How?
- Via interpreted program languages
- Via a virtual machine
- Via a standard language or API and compilation

Interpreted Programming Languages

- Via interpreted programming languages
  - The application can be written in an interpreted language (such as Python or Ruby) that has an interpreter available for multiple operating systems.
  - The interpreter reads each line of the source program, executes equivalent instructions on the native instruction set, and calls native operating system calls.
  - Performance suffers relative to that for native applications, and the interpreter provides only a subset of each operating system's features, possibly limiting the feature sets of the associated applications.

Virtual Machine

- The application can be written in a language that includes a virtual machine containing the running application.
- The virtual machine is part of the language's full Run-Time Environment (RTE).
- Example: Java and JRE
  - Java has an RTE that includes a loader, byte-code verifier, and other components that load the Java application into the Java virtual machine.
  - This RTE has been ported, or developed, for many operating systems, from mainframes to smartphones, and in theory any Java app can run within the RTE wherever it is available.
  - Systems of this kind have disadvantages similar to those of interpreters, discussed above.

Standard API and Compilation

- The application developer can use a standard language or API in which the compiler generates binaries in a machine- and operating-system-specific language.
- The application must be ported to each operating system on which it will run.
- This porting can be quite time consuming and must be done for each new version of the application, with subsequent testing and debugging.
- Example of a Standard API: the POSIX API and its set of standards for maintaining source-code compatibility between different variants of UNIX-like operating systems.

Questions?

- Applications are often system specific
- Approaches to write portable applications