Static & Shared Libraries

Shriram Raja

Ph.D. Student, Computer Science

Why Libraries?

- In large projects with several source files, you might want to consolidate commonly used functions so that you need not duplicate their definitions
- Common source files can be used but not all functions might be required for all files
- Linking a large file with unnecessary functions increases the size of the executable
- Having them as separate files will require you to specify all of them at compile time
- Hence -> static libraries: just provide the name of the library to the linker and it will link only the required object files
- Linux: static libraries are archive files which contain multiple obj files.

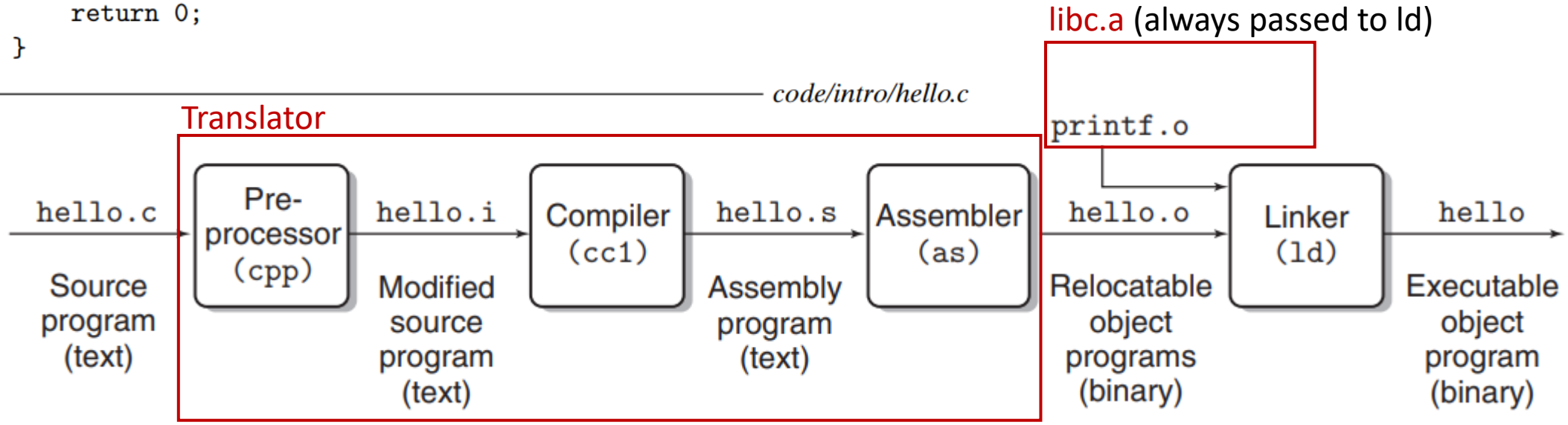
Compilation Pipeline

code/intro/hello.c

```
1   #include <stdio.h>
2
3   int main()
4   {
5       printf("hello, world\n");
6       return 0;
7   }
```

code/intro/hello.c

```
linux> gcc -o hello hello.c
```



BOSTON UNIVERSITY

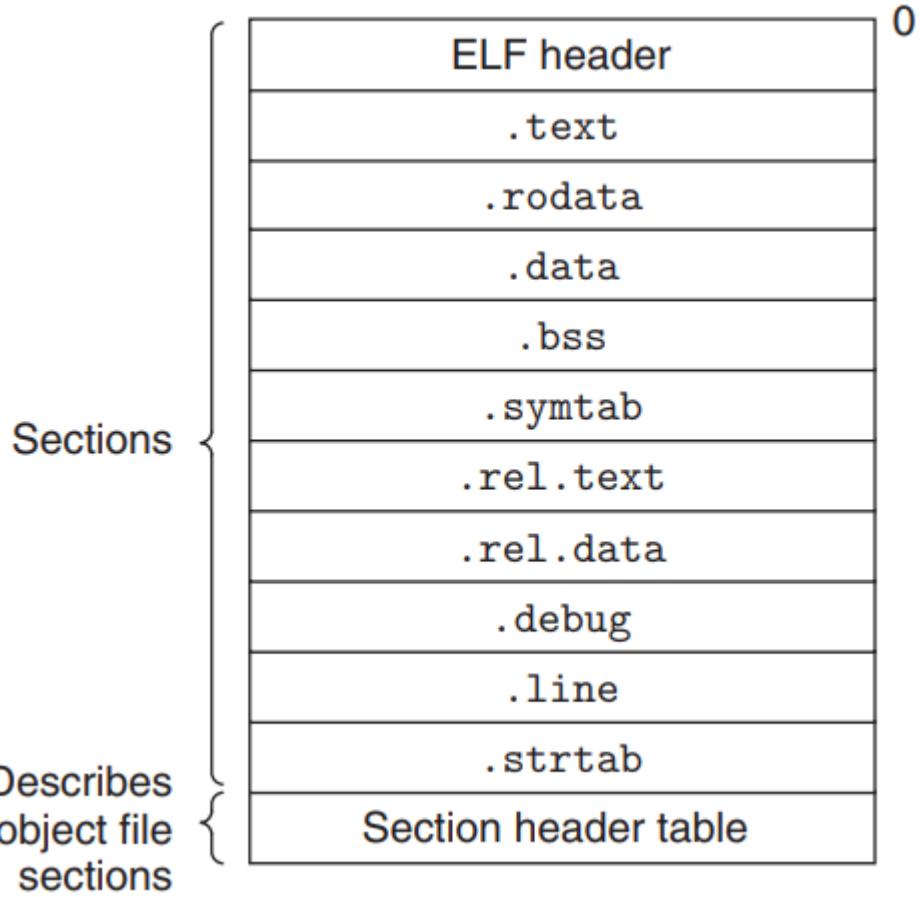
Object Files

They are called relocatable because their address starts at 0; they are relocated to actual addresses by the Linker

Type	Relocatable	Shared	Executable
Description	Can be combined with other relocatable obj files to generate an executable	Can be loaded to memory and linked dynamically	Can be copied into memory and executed
Generated by	Compilers & Assemblers		Linkers

Object Files

- Object file organization can vary from system to system
- First Unix systems: a.out format
- Windows: Portable Executable format (PE)
- Mac OS: Mach-O format
- Modern Linux: Executable and Linkable Format (ELF)
- Typical ELF relocatable obj file



Object Files

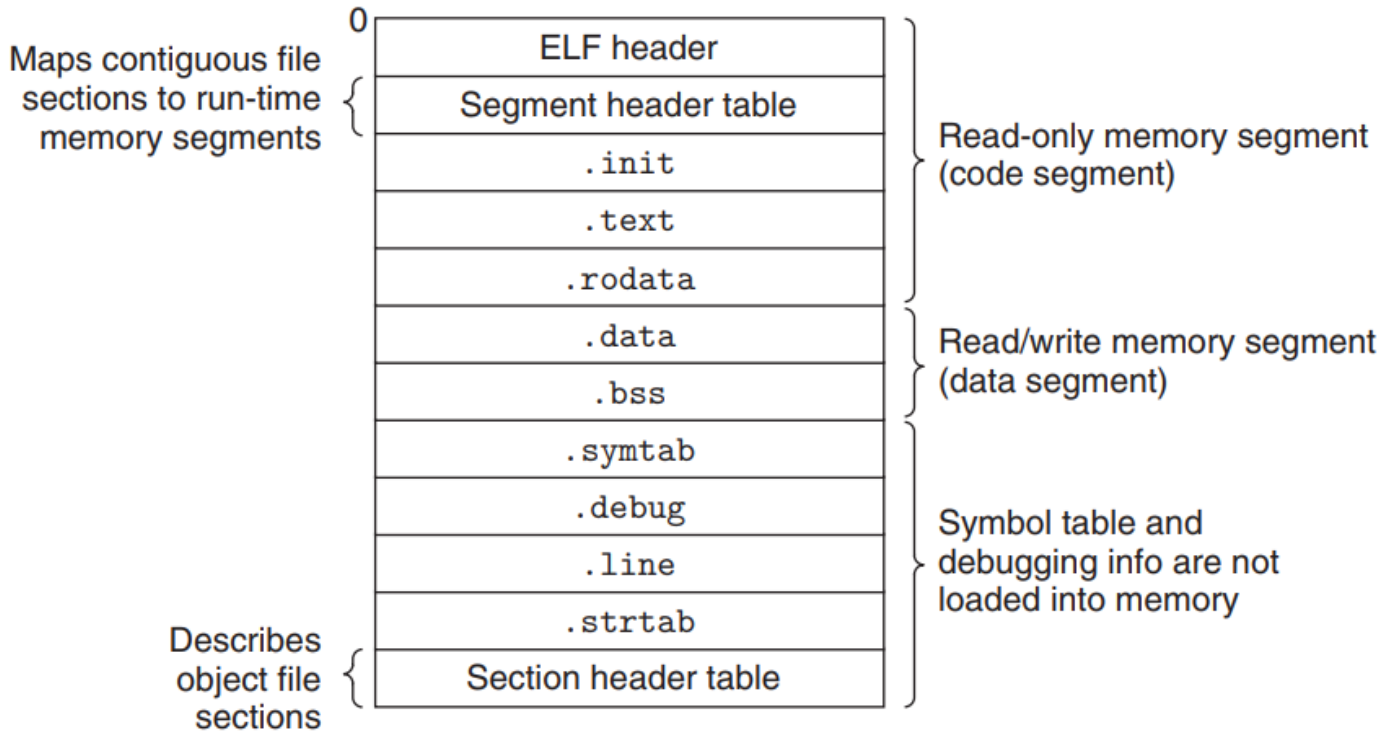
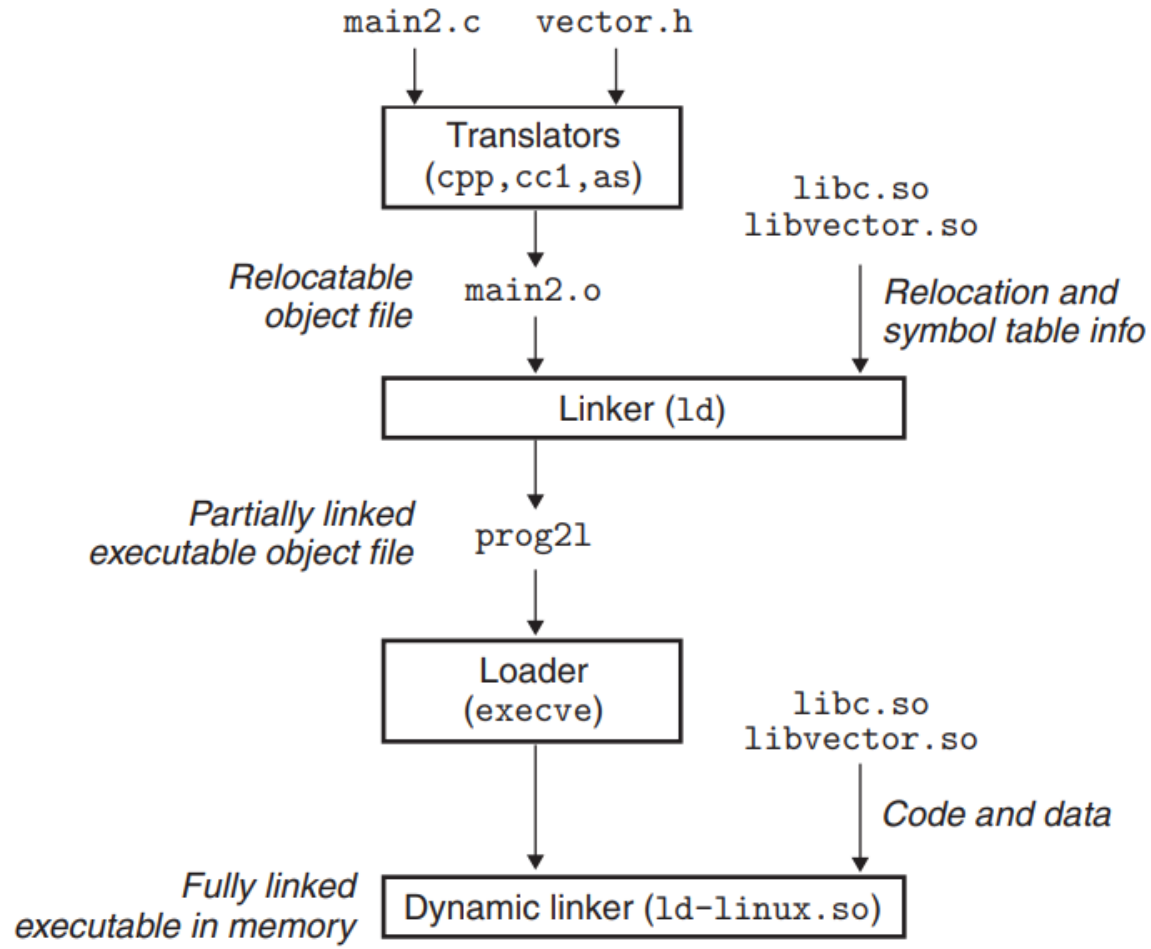


Figure 7.13 **Typical ELF executable object file.**

Shared Libraries

- Static Libraries have issues as well:
  - When static libraries are updated, the applications must be relinked with the updated version
  - When most functions in the system use the same obj files from the same static library, duplicating the code leads to wastage of memory
- Hence -> shared libraries: part of the linking is done at compile time, and the rest is done later using a dynamic linker

Dynamic Linking



Dynamic Linking

- Applications can also request dynamic libraries:
  - dlopen(): opens and links a shared library
  - dlsym(): takes a handle to the open shared library and a symbol name (variable/function) and returns the address of the symbol (Returns NULL if it wasn't found)

Compiling Static and Shared Libraries

- The ‘ar’ command is used to invoke the archiver which concatenates the given object files into an archive which can be used as a static library

```
ar rc libutil.a util_file.o util_net.o util_math.o
```

- Compiling shared objects requires slightly more work:

```
cc -fPIC -c util_file.c
cc -fPIC -c util_net.c
cc -fPIC -c util_math.c
cc -shared libutil.so util_file.o util_net.o util_math.o
```

- We can use a Makefile to streamline our compilation process and deal with these dependencies

Makefile

```
target-name : dependencies
    action
```

```
blah: blah.o
  cc blah.o -o blah # Runs third

blah.o: blah.c
  cc -c blah.c -o blah.o # Runs second

# Typically blah.c would already exist, but I want to limit any additional required files
blah.c:
  echo "int main() { return 0; }" > blah.c # Runs first
```

References

1. Computer Systems: A Programmer's Perspective (Third Edition), Randal E. Bryant & David R. O'Hallaron

2. Makefile: Getting Started, Tutorial, Automatic Variables