Linking & Loading

Dr. D. M. Akbar Hussain
Department of Electronic Systems

1

Linking

Typically, program building process involves four steps and utilizes different 'tools' such as a;

- Preprocessor
- Compiler
- Assembler
- Linker

Finally you have a single executable file. The stages that occur in the following order regardless of the operating system/compiler .

Dr. D. M. Akbar Hussain
Department of Electronic Systems

2

Linking

1.**Preprocessing** is the first pass of any C compilation. It processes include-files, conditional compilation instructions and macros.

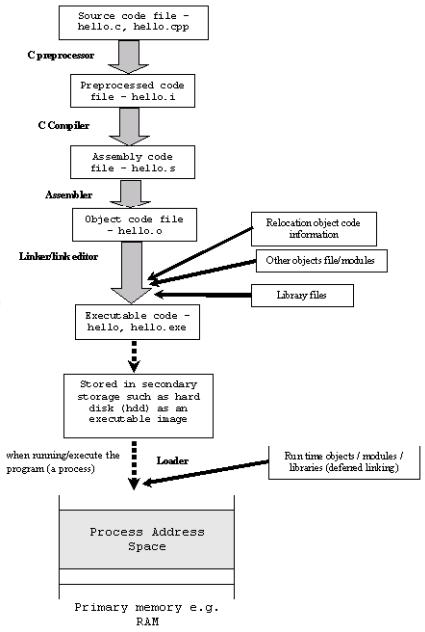
2.**Compilation** is the second pass. It takes the output of the preprocessor, and the source code, and generates assembler source code.

Linking

3.**Assembly** is the third stage of compilation. It takes the assembly source code and produces an assembly listing with offsets. The assembler output is stored in an object file.

4.**Linking** is the final stage of compilation. It takes one or more object files or libraries as input and combines them to produce a single (usually executable) file. In doing so, it resolves references to external symbols, assigns final addresses to procedures/functions and variables, and revises code and data to reflect new addresses (a process called relocation).

Linking & Loading



Dr. D. M. Akbar Hussain
Department of Electronic Systems

5

File extensions

File extension	Description
file_name.c	C source code which must be preprocessed.
file_name.i	C source code which should not be preprocessed.
file_name.ii	C++ source code which should not be preprocessed.
file_name.h	C header file (not to be compiled or linked).
file_name.cc file_name.cp file_name.cxx file_name.cpp file_name.c++ file_name.C	C++ source code which must be preprocessed. For file_name.cxx, the xx must both be literally character x and file_name.C, is capital c.
file_name.s	Assembler code.
file_name.S	Assembler code which must be preprocessed.
file_name.o	Object file by default, the object file name for a source file is made by replacing the extension .c, .i, .s etc with .o

Dr. D. M. Akbar Hussain
Department of Electronic Systems

6

Component of a Running Program

Typically, a running program has 4 components:

1. A Code Segment
2. A Stack Segment
3. Data Segment
4. Set of Registers

Linker

The Linker construct an executable code from more than one sources (object files including library object files).

It does that in obvious ways, making copies of these segments and concatenating them into an executable file.

Two main issues:

- **Addresses inside the code and data segments.**
- **Each object file may contain addresses in other object files.**

Relocation Info

For example: First object file a.o is 1000 byte, now second segment deriving object code b.o will start at location with machine address 1000. Which means all the addresses for b.o should be increased by 1000.

Therefore, linker must know the code and data segment addresses, this information is called **Relocation Information.**

There are two ways to provide such info: **Bit Map or Linked List.**

External Addresses

Second Issue: Individual object codes may contain references to other object addresses or to library object files.

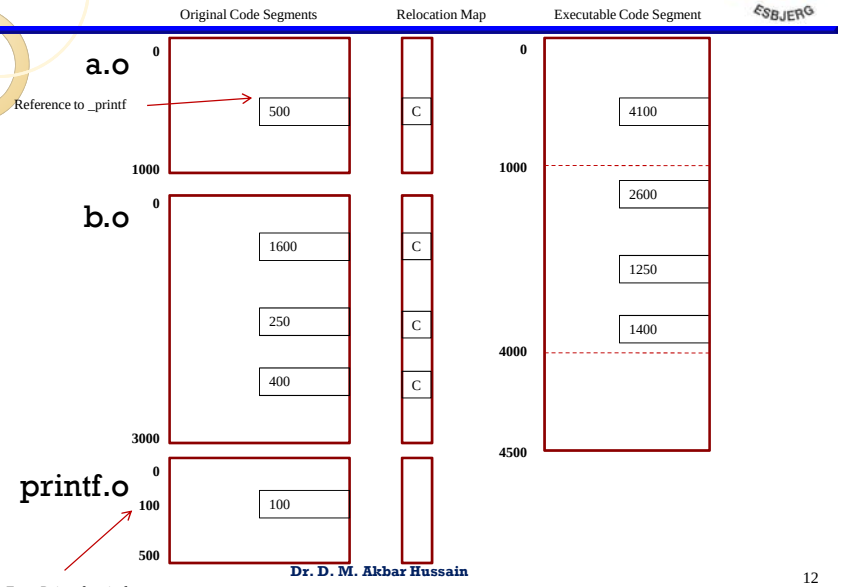
These type of references are marked with an external symbol: **"external"**

Resolving

For example in object file "a.o" printf is at location 500 (in code segment), it is an explicit reference/information, now if library object file "printf.o" has the body starting at address 100 (in the code segment), it is an explicit information in the external symbol table that it features the entry point _printf at address 100.

The job of the linker is to resolve this.

Executable Code



Object File

After the source code has been assembled, it will produce an Object files (e.g. .o, .obj) and then linked, producing an executable files.

An object and executable come in several formats such as ELF (Executable and Linking Format) and COFF (Common Object-File Format). For example, ELF is used on Linux systems, while COFF is used on Windows systems.

Other object file formats are listed in the following Table.

Object File Formats

Object File Format	Description
a.out	The a.out format is the original file format for Unix. It consists of three sections: text, data, and bss, which are for program code, initialized data, and uninitialized data, respectively. This format is so simple that it doesn't have any reserved place for debugging information. The only debugging format for a.out is stabs, which is encoded as a set of normal symbols with distinctive attributes.
COFF	The COFF (Common Object File Format) format was introduced with System V Release 3 (SVR3) Unix. COFF files may have multiple sections, each prefixed by a header. The number of sections is limited. The COFF specification includes support for debugging but the debugging information was limited. There is no file extension for this format.
ECOFF	A variant of COFF. ECOFF is an Extended COFF originally introduced for Mips and Alpha workstations.
XCOFF	The IBM RS/6000 running AIX uses an object file format called XCOFF (eXtended COFF). The COFF sections, symbols, and line numbers are used, but debugging symbols are dbx-style stabs whose strings are located in the .debug section (rather than the string table). The default name for an XCOFF executable file is a.out.
PE	Windows 9x and NT use the PE (Portable Executable) format for their executables. PE is basically COFF with additional headers. The extension normally .exe.
ELF	The ELF (Executable and Linking Format) format came with System V Release 4 (SVR4) Unix. ELF is similar to COFF in being organized into a number of sections, but it removes many of COFF's limitations. ELF used on most modern Unix systems, including GNU/Linux, Solaris and Irix. Also used on many embedded systems.
SOM/ESOM	SOM (System Object Module) and ESOM (Extended SOM) is HP's object file and debug format (not to be confused with IBM's SOM, which is a cross-language Application Binary Interface - ABI).

Sections

When we examine the content of these object files there are areas called sections. Sections can hold executable code, data, dynamic linking information, debugging data, symbol tables, relocation information, comments, string tables, and notes.

Some sections are loaded into the process image and some provide information needed in the building of a process image while still others are used only in linking object files.

There are several sections that are common to all executable formats (may be named differently, depending on the compiler/linker) as listed below:

Section Names

Section	Description
text	This section contains the executable instruction codes and is shared among every process running the same binary. This section usually has READ and EXECUTE permissions only. This section is the one most affected by optimization.
bss	BSS stands for 'Block Started by Symbol'. It holds un-initialized global and static variables. Since the BSS only holds variables that don't have any values yet, it doesn't actually need to store the image of these variables. The size that BSS will require at runtime is recorded in the object file, but the BSS (unlike the data section) doesn't take up any actual space in the object file.
data	Contains the initialized global and static variables and their values. It is usually the largest part of the executable. It usually has READ/WRITE permissions.
rdata	Also known as .rodata (read-only data) section. This contains constants and string literals.
reloc	Stores the information required for relocating the image while loading.
Symbol table	A symbol is basically a name and an address. Symbol table holds information needed to locate and relocate a program's symbolic definitions and references. A symbol table index is a subscript into this array. Index 0 both designates the first entry in the table and serves as the undefined symbol index. The symbol table contains an array of symbol entries.
Relocation records	Relocation is the process of connecting symbolic references with symbolic definitions. For example, when a program calls a function, the associated call instruction must transfer control to the proper destination address at execution. Re-locatable files must have relocation entries' which are necessary because they contain information that describes how to modify their section contents, thus allowing executable and shared object files to hold the right information for a process's program image. Simply said relocation records are information used by the linker to adjust section contents.