System Programming

For BCA 3rd Semester

Unit 4

Compiled
By
Sakhi Bandyopadhyay
Dept. of Computer Science & BCA,
Kharagpur College,
Kharagpur 721305

Linker

Linker is a program in a system which helps to link a object modules of program into a single object file. It performs the process of linking. Linker are also called link editors. Linking is process of collecting and maintaining piece of code and data into a single file. Linker also link a particular module into system library. It takes object modules from assembler as input and forms an executable file as output for loader.

Linking is performed at both compile time, when the source code is translated into machine code and load time, when the program is loaded into memory by the loader. Linking is performed at the last step in compiling a program.

```
Source code -> compiler -> Assembler -> Object code -> Linker -> Executable
file -> Loader
```

Static Linking

- In static linking, the liner links all modules of a program before its execution begins; it produces a binary program that does not contain any unresolved external references.
- If statically linked programs use the same module from a library, each program will get a private copy of the module.
- If many programs that use the module are in execution at the same time, many copies of the module might be present in memory.

Dynamic Linking

- Dynamic linking is performed during execution of a binary program.
- The linker is invoked when an unresolved external reference and resumes execution of the program.
- This arrangement has several benefits concerning use, sharing and updating of library modules.

- If the module referenced by a program has already been linked to another program that is in execution, a copy of the module would exist in memory. The same copy of the module could be lined to his program as well, thus saving memory.
- To facilitate dynamic linking, each program is first processed by the static linker.

Comparison between Linker and Loader:

BASIS FOR COMPARISON	LINKER	LOADER
Basic	It generates the executable module of a source program.	It loads the executable module to the main memory.
Input	It takes as input, the object code generated by an assembler.	It takes executable module generated by a linker.
Function	It combines all the object modules of a source code to generate an executable module.	It allocates the addresses to an executable module in main memory for execution.
Type/Approach	Linkage Editor, Dynamic linker.	Absolute loading, Relocatable loading and Dynamic Run-time loading.

LOADER

LOADER

Loader is utility program which takes object code as input prepares it for execution and loads the executable code in to the memory. Thus loader is actually responsible for initiating the execution process.

Functions of loader

The loader is responsible for the activities such as allocation, linking, relocation and loading

1) It allocates the space for program in the memory, by calculating the size of the program. This activity is called allocation.
2) It resolves the symbolic references (code/data) between the object modules by assigning all the user subroutine and library subroutine addresses. This activity is called linking.
3) There are some address dependent locations in the program, such address constants must be adjusted according to allocated space, such activity done by loader is called relocation.
4) Finally it places all the machine instructions and data of corresponding programs and subroutines into the memory. Thus program now becomes ready for execution.

Types of loader

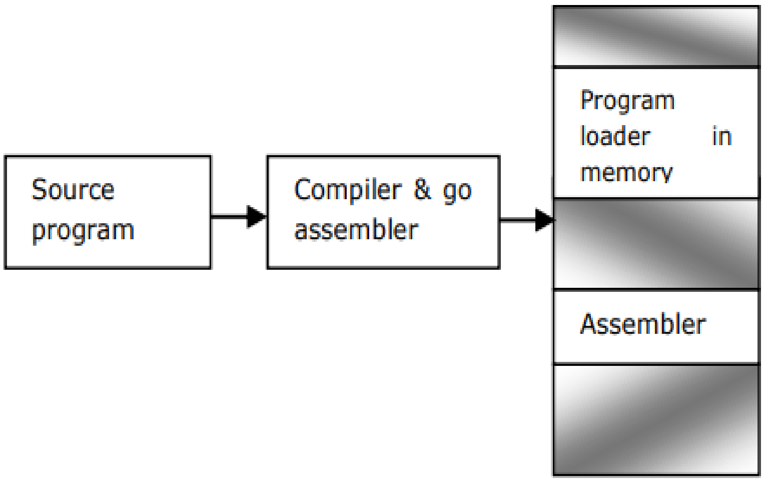
1. Compile and go to loader
2. General loader
3. absolute loader
4. direct linking loader (DLL)

Compile-and-Go Loaders

- Assembler is loaded in one part of memory and assembled program directly into their assigned memory location.
- After the loading process is complete, the assembler transfers the control to the starting instruction of the loaded program.

Advantages

- The user need not be concerned with the separate steps of compilation, assembling, linking, loading, and executing.
- Execution speed is generally much superior to interpreted systems.
- They are simple and easier to implement.



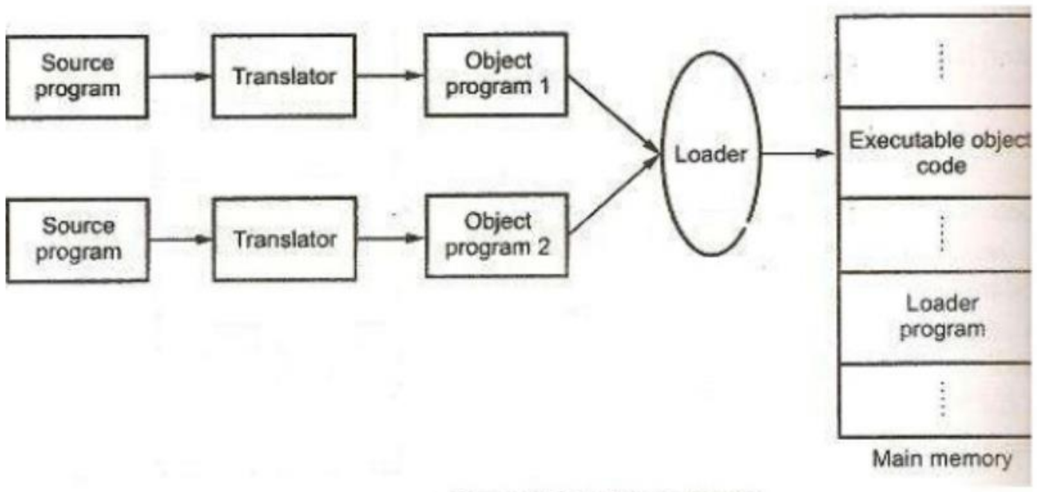
Disadvantages

- There is wastage in memory space due to the presence of the assembler.
- The code must be reprocessed every time it is run.

General Loader Schemes

- The general loading scheme improves the compile/assemble-and-go scheme by allowing different source programs (or modules of the same program) to be translated separately into their respective object programs.
- The object code (modules) is stored in the secondary storage area; and then, they are loaded.

- The loader usually combines the object codes and executes them by loading them into the memory, including the space where the assembler had been in the assemble-and-go scheme.
- Rather than the entire assembler sitting in the memory, a small utility component called loader does the job.
- Note that the loader program is comparatively much smaller than the assembler, hence making more space available to the user for their programs.



General loader scheme

Advantages of the general loading scheme:

- Saves memory and makes it available for the user program as loaders are smaller in size than assemblers. The loader replaces the assembler.
- Reassembly of the program is no more needed for later execution of the program. The object file/deck is available and can be loaded and executed directly at the desired location.
- This scheme allows use of subroutines in several different languages because the object files processed by the loader utility will all be in machine language.

Disadvantages of the general loading scheme:

- The loader is more complicated and needs to manage multiple object files.
- Secondary storage is required to store object files, and they cannot be directly placed into the memory by assemblers.

Absolute Loaders

- An absolute loader loads a binary program in memory for execution.
- The binary program is stored in a file contains the following:
    - A Header record showing the load origin, length and load time execution start address of the program.

  - A sequence of binary image records containing the program's code. Each binary image record contains a part of the program's code in the form of a sequence of bytes, the load address of the first byte of this code and a count of the number of bytes of code.
- The absolute loader notes the load origin and the length of the program mentioned in the header record.
- It then enters a loop that reads a binary image record and moves the code contained in it to the memory area starting on the address mentioned in the binary image record.
- At the end, it transfers control to the execution start address of the program.

Advantages of the absolute loading scheme:

- Simple to implement and efficient in execution.
- Saves the memory (core) because the size of the loader is smaller than that of the assembler.
- Allows use of multi-source programs written in different languages. In such cases, the given language assembler converts the source program into the language, and a common object file is then prepared by address resolution.
- The loader is simpler and just obeys the instruction regarding where to place the object code in the main memory.

Disadvantages of the absolute loading scheme:

- The programmer must know and clearly specify to the translator (the assembler) the address in the memory for inner-linking and loading of the programs. Care should be taken so that the addresses do not overlap.
- For programs with multiple subroutines, the programmer must remember the absolute address of each subroutine and use it explicitly in other subroutines to perform linking.
- If the subroutine is modified, the program has to be assembled again from first to last.

Design of direct linking loader (DLL)

The direct linking loader has the advantage of allowing the programmer multiple procedure segments and multiple data segments and allows complete freedom in referencing data or instructions contained in other segments. This provides flexible intersegment referencing and accessing ability ,while at the same time allowing independent translation of programs.

The translator must give the loader the following information with each procedure or data segment.

1. The length of segment

2. A list of all symbols in the segment that may be referenced by other segment and their relative location with in the segment

3. A list of all symbols not defined in the segment but referenced in the segment

4. Information as to where address constants are located in the segment and a description of how to revise their values

5. The machine code translation of the source program and the relative address assigned.

There are four sections to the object desk

1. External symbol directory cards(ESD)

2. Instructions and data cards called text of a program(TXT)

3. Relocation and linkage directory cards (RLD)

4. End card(END)

The ESD cards contain the information necessary to build the external directory or symbol table. External symbols are symbols that can be referred beyond the subroutine level.