Course Code: CSL602	Course Name: SPCC LAB
Class : TE-CO B-3	Date: 24/03/2021
Roll no : 18CO63	Name : SHAIKH TAUSEEF MUSHTAQUE ALI

Experiment :05

Aim: To Study & understand different schemes of loader.

Theory:

Input given to the loader is binary program which is an output from the linker. The user's source program decks are converted to the object program decks (machine language) by assemblers and compilers.

The loader is a program which accepts the object program decks, prepares these programs for execution, and initiates the execution.

The loader performs following four functions:

1. Allocation: Allocate space in memory for the programs.
2. Linking: Resolve symbolic references between object decks.
3. Relocation: Adjust all address dependant locations to corresponding to the allocated space.
4. Loading: Physically place the machine instructions and data into memory.

Loader Schemes:

There is variety of schemes for accomplishing the four functions of a loader.

1) " **Compile-and-Go" Loaders:**

One method of performing the loader function to have the assembler run in one part of memory & place the assembled machine instructions and data, as they are assembled, directly on to their assigned memory locations.

Such a scheme is commonly called "compile-and-go" or "assemble-and-go".

It is relatively easy to implement but having so many disadvantages.

1. Memory wastage: As assembler occupies some memory there is simply wastage, as this amount of memory is unavailable to the object program.
2. Retranslation: It is necessary to retranslate (assemble) the user's program deck for every execution.
3. Language: It is very difficult to handle source programs that are in different languages.

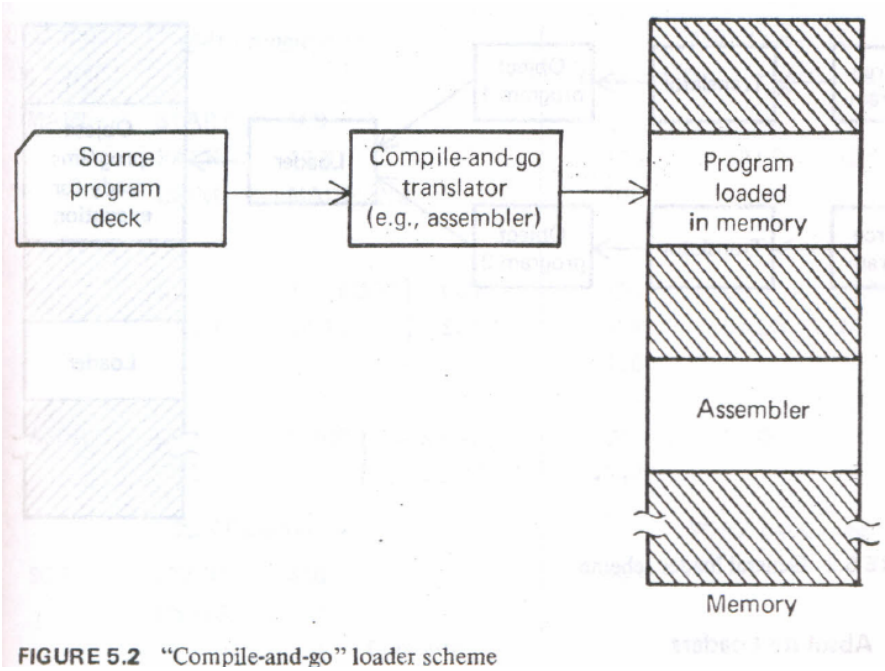

FIGURE 5.2 "Compile-and-go" loader scheme

2) General Loader Scheme

This scheme avoids the disadvantages of preceding "compile-and-go" scheme.

As the size of the loader is assumed to be smaller than the assembler, more memory is available to the user

Reassembly is not required to run the program at later date as it loads only the object deck.

Finally the problem of source program in different language is automatically handled because the object deck/ object program is always in a single language i.e. machine code.

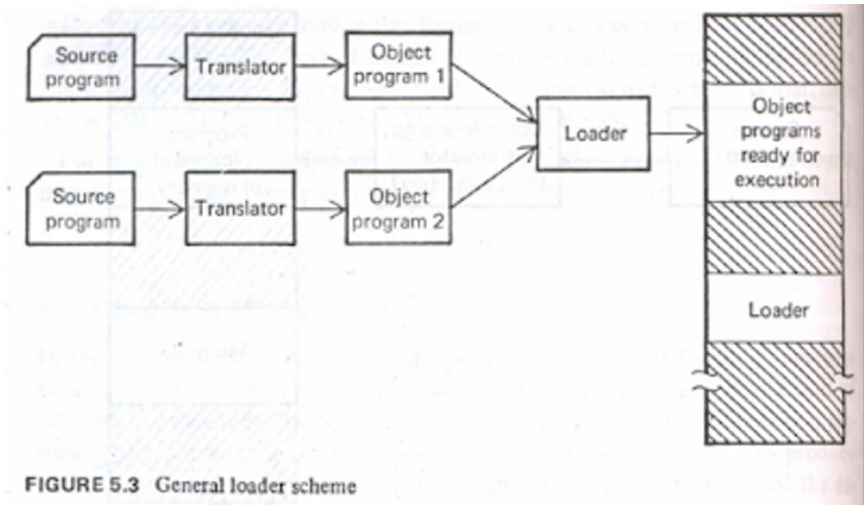

FIGURE 5.3 General loader scheme

3) Absolute Loaders:

It is the simplest type of loader scheme which fits in to general model of previous type called as absolute loader.

This scheme outputs the machine language translation of the source program and the data is punched on to the cards.

This scheme makes more core available to the user since the assembler is not in the memory at load time.

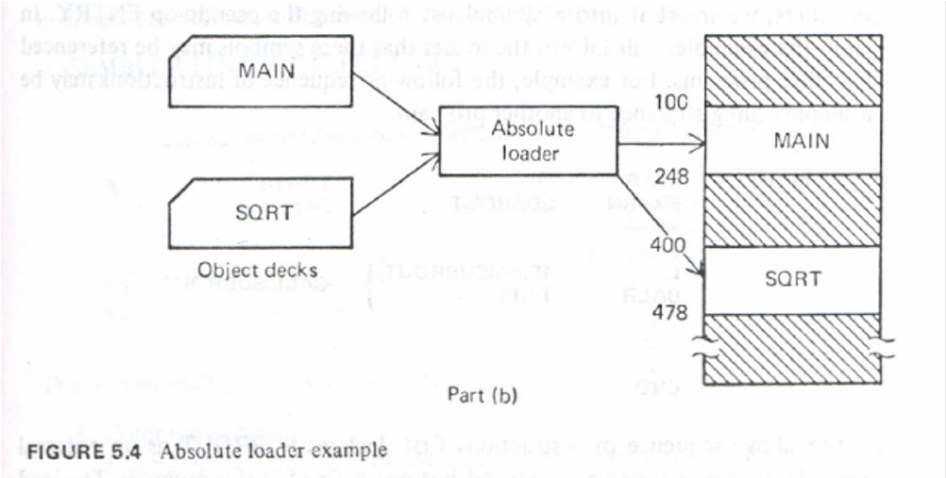

FIGURE 5.4 Absolute loader example

Absolute loaders are simple to implement but do have several disadvantages:

1. The programmer must specify the assembler the address in core where the program is to be loaded.

2. If there are multiple subroutines, the programmer must remember the address of each and use that absolute address explicitly in his other subroutines to perform subroutine linkage.

4) Relocating Loader:

To avoid the possible reassembling of all subroutines when a single subroutine is changed, and to perform the tasks of allocation and linking for the programmer, the general class of relocating loaders was introduced.

An example of relocating loader scheme is Binary symbolic subroutine (BSS).

BSS allows many procedure segment, yet only one data segment i.e. common segment.

The assembler assembles each procedure segment independently and passes on to the loader the text and information as to relocation and intersegment references.

The output of relocating assembler using a BSS scheme is the object program and also information about all other programs it references. Also there is information about

relocation as locations in this program that need to be changed if it is to be loaded in to the core.

5) Direct Linking Loaders:

It is the most popular loading scheme used today. It is a general relocatable loader.

It has the advantage of allowing the programmer multiple procedures and multiple data segments and giving him the complete freedom in referencing data or instructions contained in the other programs.

This provides flexible intersegment referencing and accessing ability while at the same time allowing independent translations of programs.

6) Dynamic Loading:

In all above schemes we have seen that all of the subroutines needed are loaded in to the core at the same time. But if the total amount of core required by all these subroutines exceeds the amount available there is a trouble.

This always happens with large programs or small computers. To solve this problem there are some hardware techniques such as paging and segmentation.

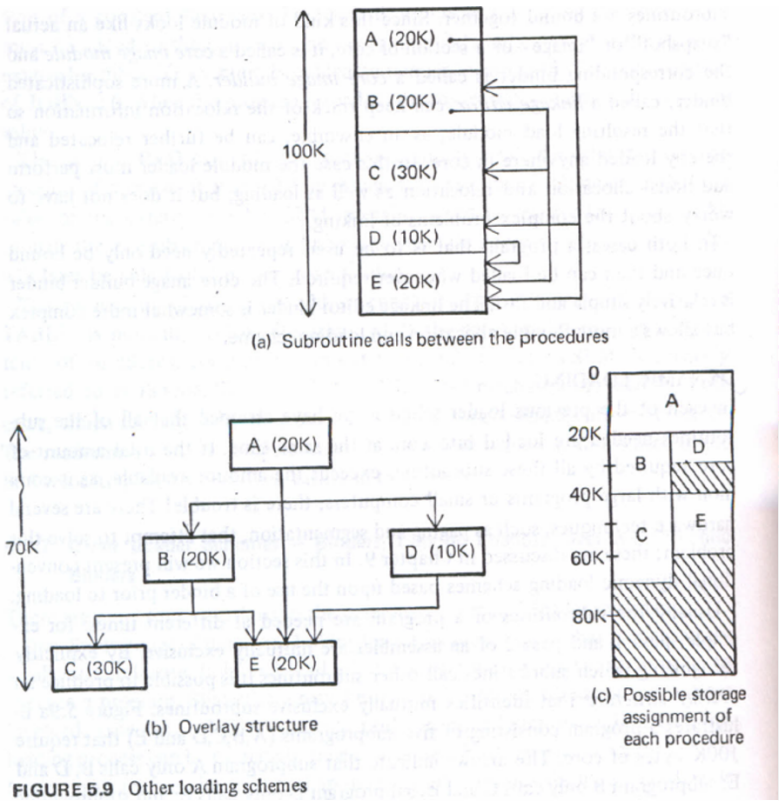

FIGURE 5.9 Other loading schemes

Conclusion:

With the help of this Experiment we get information of different schemes of loader.