**Interpreter**

In computer science, an **interpreter** is a  computer program that directly executes, i.e. *performs*, instructions written in a [programming](https://en.wikipedia.org/wiki/Programming_language) or scripting language, without previously compiling them into a machine language program. An interpreter generally uses one of the following strategies for program execution:

1. Parse the source code and perform its behavior directly.

2. Translate source code into some efficient intermediate representation and immediately execute this.

3. Explicitly execute stored precompiled code made by a compiler which is part of the interpreter system.

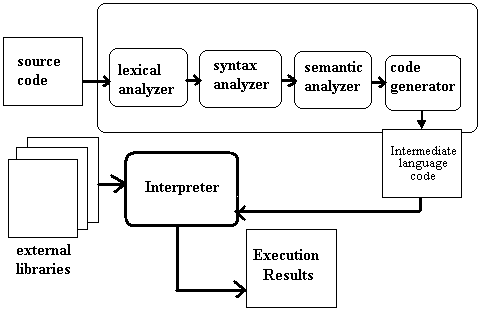
**Problems of Interpreter**

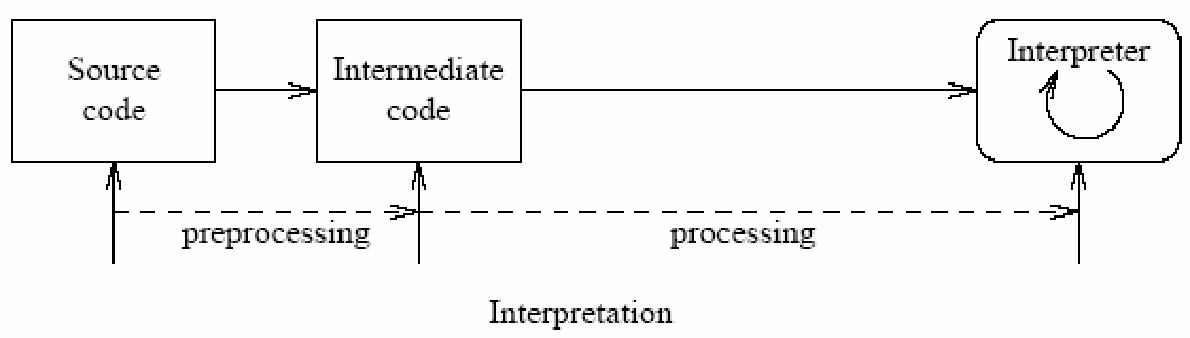
1. Interpreter generally takes much time to interpret a long program.

2. Often interpreted programs run slower than compiled programs.

3. Interpreter interprets a program line by line. So, when a bug is found in the program , at that very moment, interpreter stops interpreting and doesn’t interpret the line next lines of that program.

**Diagrams of a Interpretation**





**Linker**

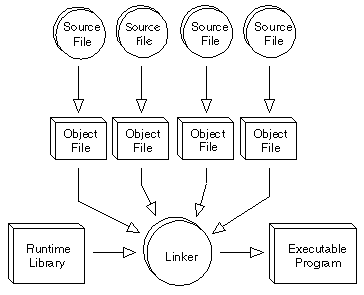
In computer science, a linker is a computer program that takes one or more object files generated by a compiler and combines them into one, executable program.

Computer programs are usually made up of multiple modules that span separate object files, each being a compiled computer program. The program as a whole refers to these separately-compiled object files using symbols. The linker combines these separate files into a single, unified program; resolving the symbolic references as it goes along.

Dynamic linking is a similar process, available on many operating systems, which postpones the resolution of some symbols until the program is executed. When the program is run, these dynamic link libraries are loaded as well. Dynamic linking does not require a linker.

The linker bundled with most Linux systems is called ID; see our ID documentation page for more information.

**Diagram of Linker**



**Loader**

In computing, a **loader** is the part of an operating system that is responsible for loading programs and libraries. It is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution. Loading a program involves reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running. Once loading is complete, the operating system starts the program by passing control to the loaded program code.

**Responsibilities of Loader**

In **Unix**, the loader is the handler for the system call **execve().** The Unix loader's tasks include:

- Validation (permissions, memory requirements etc.);

- Copying the program image from the disk into main memory;

- Copying the command-line arguments on the stack;

- Initializing registers (e.g., the stack pointer);

- Jumping to the program entry point (\_start).

In **Microsoft Windows** **7** and above, the loader is the LdrInitializeThunk function contained in ntdll.dll, that does the following:

-Initialization of structures in the DLL itself (i.e. critical sections, module lists);

-validation of executable to load;

- Creation of a heap (via the function RtlCreateHeap);

- Allocation of environment variable block and PATH block;

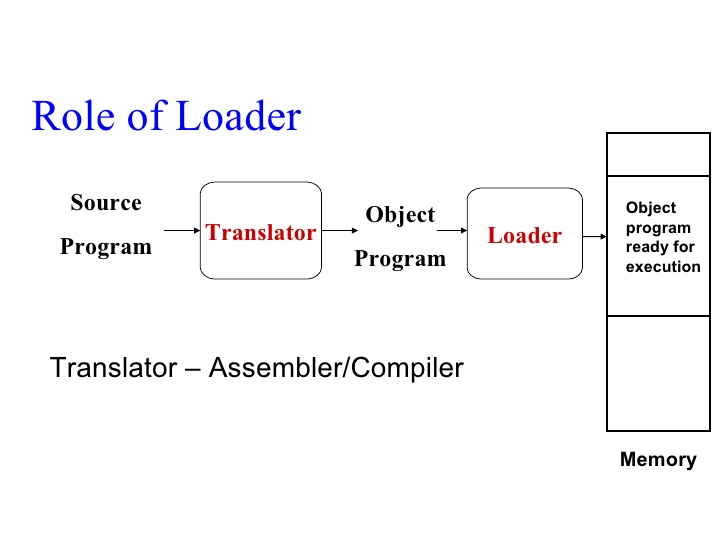
- Addition of executable and NTDLL to the module list (a doubly-linked list);

- Loading of KERNEL32.DLL to obtain several important functions, for instance BaseThreadInitThunk;

- Loading of executable's imports (i.e. dynamic-link libraries) recursively (check the imports' imports, their imports and so on);

- In debug mode, raising of system breakpoint;

**Diagram of Loader**



**Computer Software**

Sometimes abbreviated as SW and S/W, software is a collection of instructions that enable the user to interact with a computer, its hardware, or perform tasks. Computer software is coded by programmers using special languages. The code is stored inside the computer's memory or on external storage devices, such as CDs, DVDs, memory cards and USB flash drives. The word software was first used in the late 1960s to show the difference from computer hardware, which are the parts of a machine that can be seen and touched. Software is the instructions that the computer follows.

**Types of software**

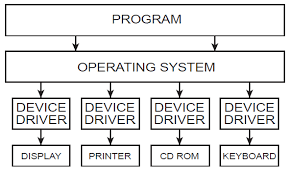
Computer software can be put into categories based on common function, type, or field of use. Following are the four types of software :

- System Software

- Application Software

- Open source Software and

- Proprietary Software

**System Software**

System software is a type of computer program that is designed to run a computer’s hardware and application program.

The two main types of system software are the operating system and the software installed with the operating system, often called utility software. The operating system and utility software typically depend on each other to function properly.

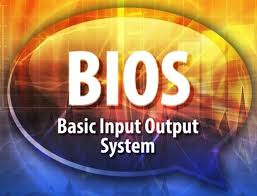
Some system software is used directly by users and other system software works in the background. System software

can allow users to interact directly with hardware functionality, like the Device Manager and many of the utilities found in the Control Panel.

**Examples of system software**

1. Utility software

2. System servers

3. Operating system (OS)

4. Windows/graphical user interface (GUI) systems

5. Microsoft Windows

6. Linux

7. Unix

8. Mac OSX

9. DOS

10. BIOS Software11. HD Sector Boot Software

12. Device Driver Software i.e Graphics Driver etc