# LECTURE – 2

# SYSTEM PROGRAMMING & SYSTEM ADMINISTRATION

## SECTION -A

**REFERENCES**: SYSTEM PROGRAMMING BY JOHN J. DONOVAN (TMH EDITION)

&

GOOGLE SEARCH ENGINE

# INTRODUCTION

- **Subroutines**
- **Macros**
- **Linkers**
- **Loaders**

# SUBROUTINE DEFINITION:

- A subroutine is a body of computer instructions designed to be used by other routines to accomplish a task.

In computer science, a **subroutine** (also called **procedure**, **method**, **function**, or **routine**) is a portion of code within a larger program that performs a specific task and is relatively independent of the remaining code.

# TWO TYPES OF SUBROUTINES ARE THERE :

- **Closed & Open Subroutines.**
- An open subroutine or macro definition is one whose code is inserted into the main program (flow continues).
- Thus if same open subroutine were called four times, it would appear in four different places in the calling program.
- Closed Subroutine:

  A closed subroutine can also be stored outside the main routine, and control transfers to the subroutine.

  Associated with the closed subroutine are the two tasks the main program must performs:

  # transfer of control

  # transfer of data.

- The task of adjusting programs so they maybe placed in arbitrary memory locations is called relocation.
- Relocating loaders perform four functions:

1. Allocate space in memory for the programs (allocation).
2. Resolve symbolic references between objects decks (linking).
3. Adjust all address dependent locations, such as address constants, to correspond to the allocated space.
4. Physically place the machine instructions and data into memory.

- The period of execution of user’s program is called execution time
- The period of translating a user’s source program is called assembly or compile time.
- Load time refers to the period of loading and preparing an object program for execution.

# Macros

- To relieve programmers of the need to repeat identical parts of their program, operating systems provide a macro processing facility, which permits the programmer to define an abbreviation for a part of his program and to use the abbreviation in his program.
- The macro processor treats the identical parts of the program defined by the abbreviation as “macro definition” and saves the definition.
- The macro processor substitutes the definition for all occurrences of the abbreviation (macro call) in the program.

- In addition to helping programmers abbreviate their programs, macro facilities have been used as general text handlers and for specializing operating systems to individual computer installations.
- In specializing operating systems ( system generation), the entire operating system is written as a series of macro definitions.
- To specialize the operating system, a series of macro calls are written.
- These are processed by the macro processor by substituting the appropriate definitions, thereby producing all the programs for an operating system.

# **Implicit & Explicit execution** in terms of Interpreters

**Explicit execution in case of Interpreters:** :- java program :-Helloworld.java.

- We'll compile it with java compiler i.e. (javac.exe file will run… as it's an executable file…)
- **javac Helloworld.java**
- Source code is converted into bytecode ( java.exe file will run…)
- **java Helloworld.java**
- This byte code is interpreted by java-runtime environment and this is explicitly done by Interpreter… (we need some kind of environment like JRE explicitly to execute our java programs….)

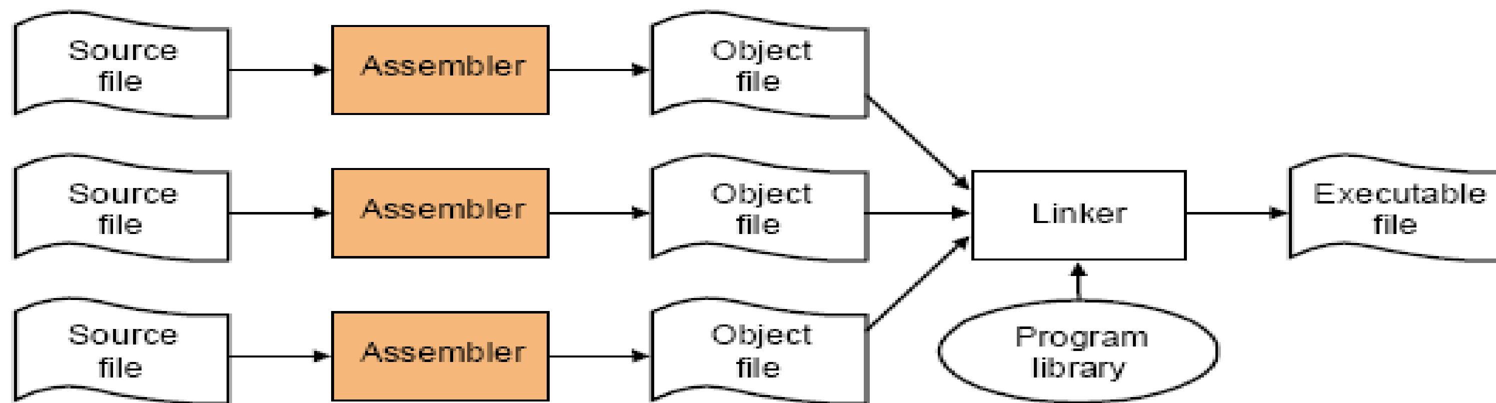
**Implicit execution in case of compilers:-**

- C programs are implicitly executed by compiler as they are simply compiled and executed
- There's no need of interpreters or some other environment to execute the instructions…that's why we cal in implicit execution..

# Linker

- Also called *link editor* and *binder,* a linker is a program that combines object modules to form an executable program.
- Many programming languages allow you to write different pieces of code, called *modules*, separately.
- This simplifies the programming task because you can break a large program into small, more manageable pieces.
- Eventually, though, you need to put all the modules together. This is the job of the linker.

Compilers and assemblers create object files containing the generated binary code and data for a source file.




A ``program library'' is simply a file containing compiled code (and data) that is to be incorporated later into a program; program libraries allow programs to be faster to recompile, and easier to update.

# Basic doubts of students

- Static and dynamic linking
- linkers in detail
- Implicit and explicit execution
- Program generators examples.
- Translation Hierarchy_where to place interpreters in the diagram.

( In the diagram of translation hierarchy we are considering an example of C program.. And C programs are only compiled and executed there's no requirement of interpreters in C programming…they are needed in case of Java programming)

## Static linking

- A compiler can generate static or dynamic code depending upon how you proceed with the linking process.
- If you create static object code (we call it static because they exist through out the program execution), the output files are larger but they can be used as independent binary files .
- This means that you can copy an executable file to another system and It does not depend on shared libraries when it is executed.
- **(Shared libraries are libraries that are loaded by programs when they start. When a shared library is installed properly, all programs that start afterwards automatically use the new shared library.** )
- **Dynamic object Code**: It is often useful to create objects dynamically as well. The main reason you'd want to create an object dynamically, rather than define it statically, is that you don't know in advance that you'll need the object at all - or, more typically, you don't know exactly how many instances of the object you'll need.

## Dynamic linking

- if you chose dynamic linking, the final executable code is much smaller but it depends heavily upon shared libraries. If you copy the final executable program to another system, you have to make sure that the shared libraries are also present on the system where your application is executed.

# APPLICATIONS

- **Evaluation order**

  Macro systems have a range of uses. Being able to choose the order of evaluation (see lazy evaluation and non-strict functions) enables the creation of new syntactic constructs (e.g control structures) indistinguishable from those built into the language. For instance, in a Lisp dialect that has cond but lacks if, it is possible to define the latter in terms of the former using macros. For example, Scheme has both continuations and hygienic macros, which enables a programmer to design their own control abstractions, such as looping and early exit constructs, without the need to build them into the language.

- **Data sub-languages and domain-specific languages**

  Next, macros make it possible to define data languages that are immediately compiled into code, which means that constructs such as state machines can be implemented in a way that is both natural and efficient.

- **Binding constructs**

  Macros can also be used to introduce new binding constructs. The most well-known example is the transformation of let into the application of a function to a set of arguments.

# SCOPE OF RESEARCH

The linker actually enables separate compilation. As shown in Figure , an executable can be made up of a number of source files which can be compiled and assembled into their object files respectively, independently.