* System Programming and Operating System (SPOS) - Assignment Number - 3

Name :- Kaustubh Shrikant Kabra.
Class :- Third Year Engineering.
Div :- A Roll Number :- 38
Batch :- T2
Department :- Computer Department
College :- AISSMS's IOIT.

★ Compare different types of loader scheme. Explain static and dynamic linking. Explain lexical analysis, syntax analysis and semantic analysis with an example.

→ There are 6 types of loader schemes:

i) Compile and Go loader -
Assembles lies in the memory and loader itself does the process of assembling and loading machine instruction and data at specified memory locations

ii) General loader schemes -
Source program is converted into obj: program by some translator. Loader accepts these obj modules and puts machine instruction and data in an executable form at thier assigned memory.

iii) Absolute loader -
It take O/P of assembles and loads it into memory without relocation. The O/P of the assembler can be stored on any machine readable forms of storage.

Page:

iv) Subroutine linking loader-
The way in which a machine makes it possible to call and return from subroutine is referred to as subroutine linkage method.

v) Relocating Loader-
It loads a program in specific area of memory, relocates it so that it can execute correctly.

vi) Direct linking loader-
It is type of relocatable loader most common type of loader. It cannot have direct access to source code.

Static Linking-
The linker links all modules of a program before its execution starts it create a binary program that does not includes any external references which is not resolved.

Dynamic linking-
Dynamic linking is performed at the time of execution of a binary library. Only one copy of shared library is kept in memory.
The load time might be reduced if the shared library code is already pressure in memory.

Java Class Loader -

Java class loader is an abstract class. It belongs to a java.lang package. It loads classes from different resourses. Java class loader is used to load the classes at run time. In other words, JVM performs the linking process at runtime. In this way, uniqueness is maintained in the runtime environment. It is essential to execute a Java program.

Java Class Loader is based on three principales:-

- Delegation principle
- Visibility principle
- Uniqueness principle.

Scanned by TapScanner

Date :
Page :

Types of Class Loader -

In Java, every class loader has a predefined location from where they load class files. There are following types of class loader in Java:

1) Bootstrap Class Loader:
It load standard JDK class files from rt.jar and other core classes. It is a parent of all class loaders. It doesn't have any parent.

2) Extension Class Loader:
It delegates class loading request to its parents. If the loading of a class is unsuccessful, it loads classes from jre/lib/ext directory.

3) System Class Loader:
It loads application specific classes from the Class Path environment variable. It is a child of extension class loader.

How Class Loader works in Java :-

When JVM request for a class, it invokes a loadclass() method of the java.lang.classloader class by passing the fully classified name of the class. The load class() method calls for find loaderclass() method to check that the class has been already loaded or not. It is required to avoid loading the class multiple times.

If the class is already loaded, it delegates the request to parent class loader to load the class. If the ClassLoader is not finding the class, it invokes the ~~class~~ findclass() method to look for classes in the file system.

Visibility principle states that child ClassLoader can see the class loaded by the parent ClassLoader, but vice versa is not true.

According to uniqueness principle, a class loaded by the parent should not be loaded by child class loader again.

In short, class loader follows the following rule:

- It checks if the class is already loaded.
- If the class is not loaded, ask parent class loader to load the class.
- If parent class loader cannot load class, attempt to load it in this class loader.

When classes are loaded -

There are only two cases:

1) When the new byte code is excuted.

2) When the byte code makes a static reference to a class.

For example :-> System. out