ASSIGNMENT B1

TITLE : Dynamic Link Loading

Problem Statement:
Write a program to create Dynamic Link Library for any mathematical operation and write an application program to test it.
(Java Native interface/ Use VB or VC++)

Objectives:

- To understand Dynamic Link Library concepts.
- To implement Dynamic Link Library.
- To study about Visual Basic.

Outcomes:
I will be able to understand and implement Dynamic Link Library and be able to use Visual Basic.

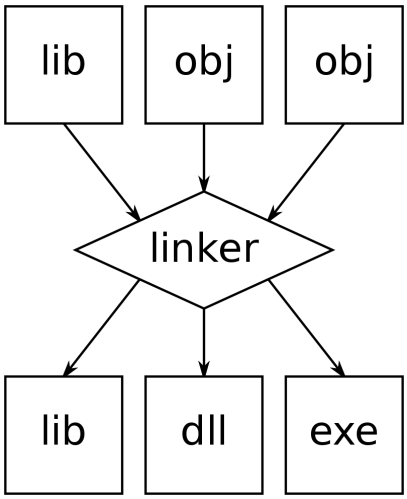
Software and Hardware Requirements:

- Working PC.
- 64 bit Fedora OS
- Eclipse IDE and JAVA
- I3 processor

Theory

- **LINKING:**

  Linking is the process of collecting and combining various pieces of code and data into a single file that can be loaded (copied) into memory and executed. Linking can be performed at compile time, when the source code is translated into machine code, at load time, when the program is loaded into memory and executed by the loader, and even at run time, by application programs. On early computer systems, linking was performed manually. On modern systems, linking is performed automatically by programs called linkers.


lib
obj
obj
linker
lib
dll
exe

- **Relocation:**

  Relocation is the process of assigning load addresses for position-dependent code and data of a program and adjusting the code and data to reflect the assigned addresses. Prior to the advent of multiprocess systems, and still in many embedded systems, the addresses for objects were absolute starting at a known location, often zero. Since multiprocessing systems dynamically link and switch between programs it became necessary to be able to relocate objects using position-independent code. A linker usually performs relocation in conjunction with **symbol resolution**, the process of searching files and libraries to replace symbolic references or names of libraries with actual usable addresses in memory before running a program.

- **Self Relocation:**

  Self-relocation is similar to the relocation process employed by the linker-loader when a program is copied from external storage into main memory; the difference is that it is the loaded program itself rather than the loader in the operating system or shell that performs the relocation.

  One form of self-relocation occurs when a program copies the code of its instructions from one sequence of locations to another sequence of locations within the main memory of a single computer, and then transfers processor control from the instructions found at the source locations of memory to the instructions found at the destination locations of memory. As such, the data operated upon by the algorithm of the program is the sequence of bytes which define the program.

  Self-relocation typically happens at load-time (after the operating system has loaded the software and passed control to it, but still before its initialization has finished), sometimes also when changing the program's configuration at a later stage during runtime.

- **Static Linking**

  When we click the .exe (executable) file of the program and it starts running, all the necessary contents of the binary file have been loaded into the process's virtual address space. However, most programs also need to run functions from the system libraries, and these library functions also need to be loaded.

  In the simplest case, the necessary library functions are embedded directly in the program's executable binary file. Such a program is statically linked to its libraries, and statically linked executable codes can commence running as soon as they are loaded.

- **Disadvantage of static linking:**

  Every program generated must contain copies of exactly the same common system library functions. In terms of both physical memory and disk-space usage, it is much more efficient to load the system libraries into memory only once. Dynamic linking allows this single loading to happen.

- **Dynamic Linking**

  Every dynamically linked program contains a small, statically linked function that is called when the program starts. This static function only maps the link library into memory and runs the code that the function contains. The link library determines what are all the dynamic libraries which the program requires along with the names of the variables and functions needed from those libraries by reading the information contained in sections of the library.

  After which it maps the libraries into the middle of virtual memory and resolves the references to the symbols contained in those libraries. We don't know where in the memory these shared libraries are actually mapped: They are compiled into position-independent code (PIC), that can run at any address in memory.

- **Advantage:**

  Memory requirements of the program are reduced. A DLL is loaded into memory only once, whereas more than one application may use a single DLL at the moment, thus saving memory space. Application support and maintenance costs are also lowered.

- **Dynamic Link Library**

  A dynamic-link library (DLL) is a module that contains functions and data that can be used by another module (application or DLL).

  A DLL can define two kinds of functions: exported and internal. The exported functions are intended to be called by other modules, as well as from within the DLL where they are defined. Internal functions are typically intended to be called only from within the DLL where they are defined. Although a DLL can export data, its data is generally used only by its functions. However, there is nothing to prevent another module from reading or writing that address.

  DLLs provide a way to modularize applications so that their functionality can be updated and reused more easily. DLLs also help reduce memory overhead when several applications use the same functionality at the same time, because although each application receives its own copy of the DLL data, the applications share the DLL code.

A DLL file is often given a ".dll" file name suffix. DLL files are dynamically linked with the program that uses them during program execution rather than being compiled into the main program.

The advantage of DLL files is that space is saved in random access memory (RAM) because the files don't get loaded into RAM together with the main program. When a DLL file is needed, it is loaded and run. For example, as long as a user is editing a document in Microsoft Word, the printer DLL file does not need to be loaded into RAM. If the user decides to print the document, the Word application causes the printer DLL file to be loaded and run.

A program is separated into modules when using a DLL. With modularized components, a program can be sold by module, have faster load times and be updated without altering other parts of the program. DLLs help operating systems and programs run faster, use memory efficiently and take up less disk space.

Conclusion:

Thus we have successfully implemented a Dynamic Link Library for a mathematical operation and tested it on a Java interface.