Experiment - 1 Case study on various system programs

Name : **Roll-no :**
Batch : B3

1. Difference between System Program and application program

- System program aims to produce software which provides services to the computer hardware.
- Application programming is used to build application software which includes software like notepad, Word Pad, calculator, web browser, Microsoft excel and many more. It interacts with system software which in turn interacts and makes physical hardware functional.

System Program	Application Serif
1) It is the collection of components and art or designing of a given program.	1) It is the set of programs that view computer as a tool for solving a particular problem.
2) Programming is done using assembly language which interacts with hardware.	2) Application programming is used to build application software which includes software like C,Java,Python
3) System software that executes the application software.	3) Application software is a software that are been used by the end user.
4) System programming is used to write low level instructions	4)Application Programming is used to write High level instructions
5)Examples:- Loader, Linker, Compiler	5) Examples:- Library management System, calculator, web browser

2. Difference between System Programming and application programming

System Programming and Maintenance

System programmers are responsible for the maintenance of the operating system and its infrastructure. This can range from installing new products, integrating third-party solutions, and even developing their own software to customize the platform. It is here that a system programmer may operate occasionally as or with an application programmer. They may need to work to hook products in directly to their system, or even create their own system-level solutions.

A system programmer needs to be knowledgeable about the entirety of the system's infrastructure and its resources. They will need to ensure that the system is operating as productively and efficiently as possible. They will need to be able to plan for the future needs of the business and to decommission and replace solutions as necessary. For the most part, the role of the system programmer will lie in keeping the system updated and monitoring it for any potential issues.

Application Programming and Development

An applications programmer is charged with the development and programming of a software solution. Applications programmers may create proprietary frameworks for an organization or may be tasked with developing software solutions from the ground up. In recent years, many applications programmers have been developing for cloud-based infrastructures and mobile environments.When hired internally at a business, an applications programmer will generally be called upon to provide solutions for the company. This begins with determining the needs of the business. An applications programmer will often interview users directly to identify the best direction for the software to take. They will then prototype and test the solution until it's able to fulfill the company's goals.

Integrating System Programming and Application Programming

A system programmer will often work with the applications developed by an application programmer. Though the application programmer will be in charge of developing the solution, the system programmer will need to integrate the solution into the system. Both the application programmer and the system programmer will likely have similar backgrounds regarding their programming knowledge and experience, but system programmers will likely do less programming and more resource management. In many cases (though not always), a system programmer is required to have a wider array of experience and training.

3. Difference between Compiler and Interpreter

A compiler is a piece of code that translates the high level language into machine language. When a user writes a code in a high level language such as Java and wants it to execute, a specific compiler which is designed for Java is used before it will be executed. The compiler scans the entire program first and then translates it into machine code which will be executed by the computer processor and the corresponding tasks will be performed.

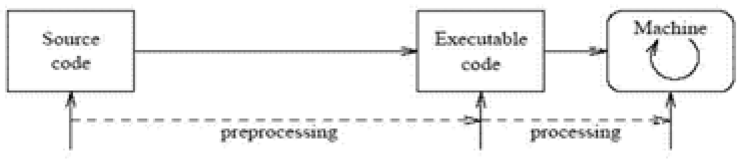

Fig 1 : Compilation

Interpreters are not much different than compilers. They also convert the high level language into machine readable binary equivalents. Each time when an interpreter gets a high level language code to be executed, it converts the code into an intermediate code before converting it into the machine code. Each part of the code is interpreted and then execute separately in a sequence and an error is found in a part of the code it will stop the interpretation of the code without translating the next set of the codes.

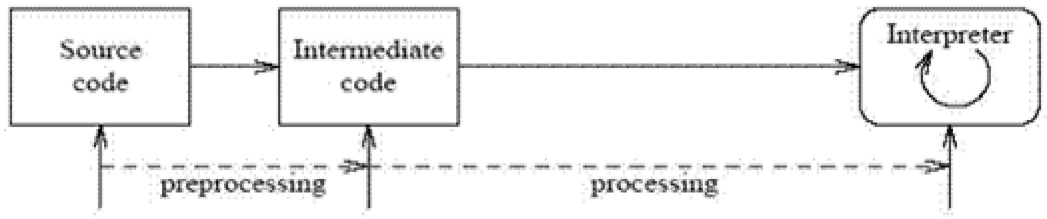

Fig 2 : Interpretation

Compiler	Interpreter
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

4. Difference between Assembler and Compiler

The compiler works on a basic principle. It simply reads the whole program written on it. Then it converts the whole program to the machine/computer language. For this process, the compiler takes a lot of time to read the whole program or to analyze the whole program

Assembler is also a compiler. But in the assembler, the source code is written in the Assembly language. This assembly language is a simple language. And this language is understandable by the humans as well. The work of an assembler is to convert the assembly language to the machine language. The assembler works on the principle of the one to one mapping translation.

Compiler	Interpreter
Generates the assembly language code or directly the executable code.	Generates the relocatable machine code.
The input here is, Preprocessed source code.	The input here is, Assembly language code.
The compilation phases are lexical analyzer, syntax analyzer, semantic analyzer, intermediate code generation, code optimization, code generation.	Assembler makes two passes over the given input.
The assembly code generated by the compiler is a mnemonic version of machine code.	The relocatable machine code generated by an assembler is represented by binary code.
High level programming like C uses compiler	Assembly Language programming like Turbo Assembler uses assembler

5. Difference between Assembler and interpreter

Assembers : Assembler are used to convert assembly language code into machine code. Example turbo assembler

Interpreter : An interpreter is a computer program which executes a statement directly (at runtime). Example python

Assembler	Interpreter
An assembler translates an assembly language program into its equivalent machine language program.	A interpreter translates a high-level language program into its equivalent machine language program.
Assembler is software or a tool that translates Assembly language to machine code. So, an assembler is a type of a compiler and the source code is written in Assembly language.	An interpreter is a computer program or a tool that executes programming instructions. An interpreter may either execute the source code directly or converts the source to an intermediate code and execute it directly or execute precompiled code produced by a compiler

Assemblers produce an object code, which might have to be linked using linker programs in order to run on a machine	most interpreters can complete the execution of a program by themselves.
Assemblers are used in the ocassions for producing code that runs very efficiently for occasions in which performance is very important (for e.g. graphics engines, washing machines, etc.)	interpreters are used when we need high portability. For example, the same Java bytecode can be run on different platforms by using the appropriate interpreter (JVM).
An Assembler reads assembly language (Object Code)	Interpreters read a higher level language and execute the code directly.

6. Brief overview about assembler, linker, loader, compiler, interpreter, device driver

- **Compiler**
  A Compiler is a language translator that takes as input a source program in some HLL and converts it into a lower-level language (i.e. machine or assembly language).
  So, an HLL program is first compiled to generate an object file with machine-level instructions (i.e. compile time) and then instructions in object file are executed (i.e. run time).

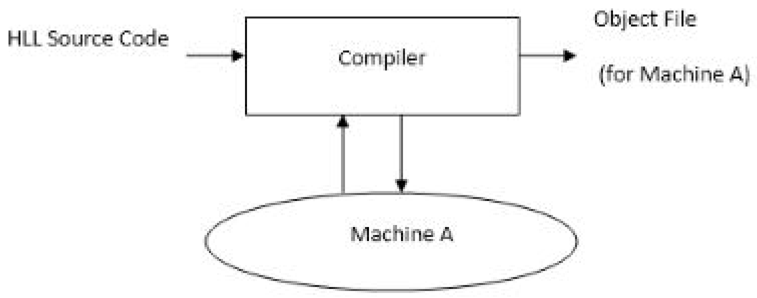

Fig 3 : Compiler

- **Linker**
  A Linker (or a Linkage Editor) takes the object file, loads and compiles the external sub-routines from the library and resolves their external references in the main-program.
  A Compiler generates an object file after compiling the source code. But this object file cannot be executed immediately after it gets generated. This is because the main program may use separate subroutines in its code (locally defined in the program or available globally as language subroutines). The external sub-routines have not been compiled with the main program and therefore their addresses are not known in the program.

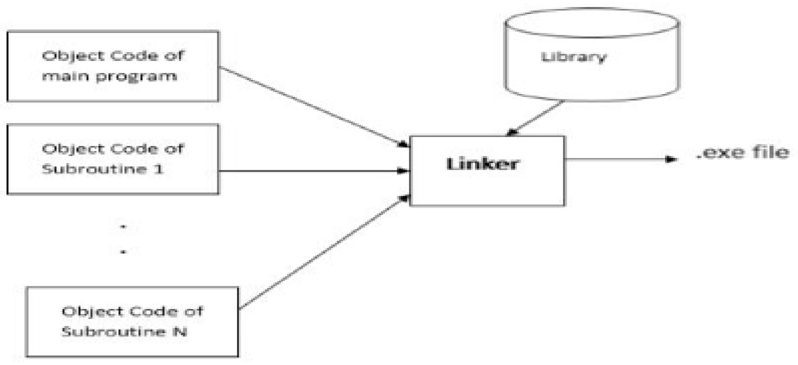
Object Code of main program
Object Code of Subroutine 1
Object Code of Subroutine N
Library
Linker
.exe file

Fig 4 : Linker

- **Loader**

  Programmers usually define the program to be placed at some pre-defined location in the memory. But this loading address given by the programmer may never be used, as it has not been coordinated with the OS

  A Loader does the job of coordinating with the OS to get the initial loading address for the program, prepares the program for execution (i.e. generates an .exe file) and loads it at that address.

  Also, during the course of its execution, a program may be relocated to a different area of main memory by the OS (when memory is needed for other programs). This may render address-sensitive instructions useless (For example, load / store from a specific address in the memory).

  An important job of Loader is to modify these address-sensitive instructions, so that they run correctly after relocation.

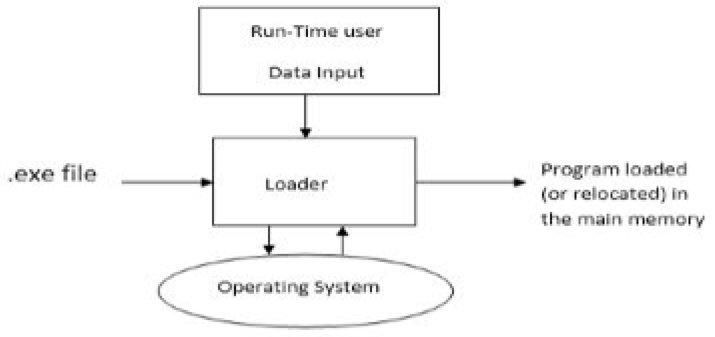

Fig 5 : Loader

- **Interpreter**

  An Interpreter is similar to a compiler, but one big difference is that it executes each line of source code as soon as its equivalent machine code is generated. (This approach is different from a compiler, which compiles the entire source code into an object file that is executed separately).

  If there are any errors during interpretation, they are notified immediately to the programmer and remaining source code lines are not processed. The main advantage with Interpreters is that, since it immediately notifies an error to the programmer, debugging becomes a lot easier.

- **Assembler**
  Assembler is a language translator which takes as input a program in assembly language of machine A and generates its equivalent machine code for machine A, and the assembler itself is written in assembly language of Machine A.

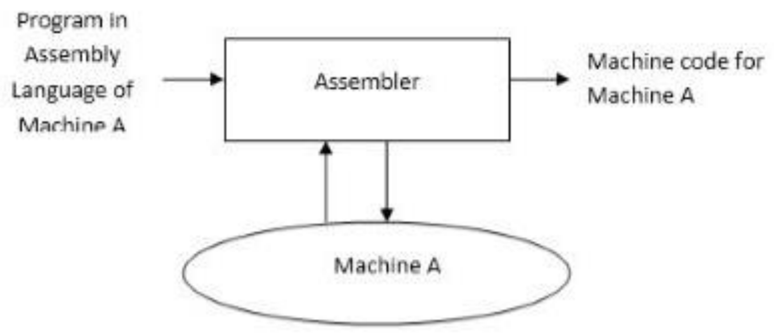
Program in Assembly Language of Machine A
Assembler
Machine code for Machine A
Machine A

Fig 6 : Assembler

- **Device driver**
  A device driver is special software that permits the operating system to interact with a specific device, such as a printer, a mouse, or a hard drive. A device driver enables the operating system to recognize the device and to use a standard inter-face (provided as an application programming interface, or API) to access and control devices.
  Think of device drivers like translators between a program we're using and a device that that program wants to utilize somehow. The software and the hardware were created by different people or companies and speak two completely different languages, so a translator (the driver) allows them to communicate.
  In other words, a software program can provide information to a driver to explain what it wants a piece of hardware to do, information the device driver understands and then can fulfil with the hardware.
  Thanks to device drivers, most software programs don't need to know how to work directly with hardware, and a driver doesn't need to include a full application experience for users to interact with. Instead, the program and driver simply need to know how to interface with each other.