Yasmin Ansari
TE – CMPN (A)
Roll No: 71

PRACTICAL NO –1

AIM: Case Study on system programming.

System programmingis the activity of programmingcomputersystem software. The primary distinguishing characteristic of systems programming when compared to application programming is that application programming aims to produce software which provides services to the user directly (e.g. word processor), whereas systems programming aims to produce software and software platforms which provide services to other software, are performance constrained, or usually both (e.g. operating systems, computational science applications, game engines and AAA video games, industrial automation, and software as a service applications)

Comparison between System program and Application program:

Subject	Application Program	System Program
Definition	Application software is computer software designed to help the user to perform specific tasks.	System software is computer software designed to operate the computer hardware and to provide a platform for running application software.
Purpose	It is specific purpose software.	It is general-purpose software.
Classification	Package Program, Customized Program	Time Sharing, Resource Sharing, Client Server Batch Processing Operating System Real time Operating System Multi-processing Operating System Multi-programming Operating System Distributed Operating System
Environment	Application Software performs in a environment which created by System/Operating System	System Software Create his own environment to run itself and run other application.
Execution Time	It executes as and when required.	It executes all the time in computer.
Essentiality	Application is not essential for a computer.	System software is essential for a computer
Number	The number of application software is much more than system software.	The number of system software is less than application software.

1) Compiler:

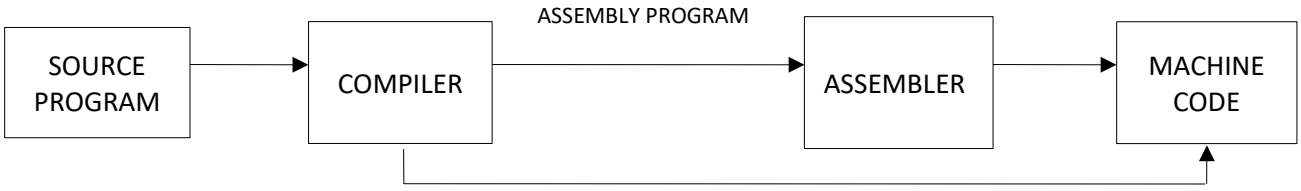
A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as object code. The most common reason for converting source code is to create an executable program.

The name "compiler" is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language or machine code). If the compiled program can run on a computer whose CPU or operating system is different from the one on which the compiler runs, the compiler is known as a cross-compiler. More generally, compilers are a specific type of translator.

Structure of a compiler

Compilers bridge source programs in high-level languages with the underlying hardware. A compiler verifies code syntax, generates efficient object code, performs run-time organization, and formats the output according to assembler and linker conventions. A compiler consists of:

- The front end: Verifies syntax and semantics, and generates an intermediate representation or IR of the source code for processing by the middle-end. Performs type checking by collecting type information. Generates errors and warning, if any, in a useful way. Aspects of the front end include lexical analysis, syntax analysis, and semantic analysis.
- The middle end: Performs optimizations, including removal of useless or unreachable code, discovery and propagation of constant values, relocation of computation to a less frequently executed place (e.g., out of a loop), or specialization of computation based on the context. Generates another IR for the backend.
- The back end: Generates the assembly code, performing register allocation in process. (Assigns processor registers for the program variables where possible.) Optimizes target code utilization of the hardware by figuring out how to keep parallel execution units busy, filling delay slots. Although most algorithms for optimization are in NP, heuristic techniques are well-developed.



2)Assembler:

An assembler program creates object code by translating combinations of mnemonics and syntax for operations and addressing modes into their numerical equivalents. This representation typically includes an operation code ("opcode") as well as other control bits. The assembler also calculates constant expressions and resolves symbolic names for memory locations and other

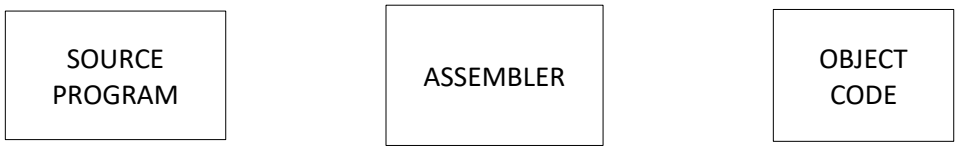
entities. The use of symbolic references is a key feature of assemblers, saving tedious calculations and manual address updates after program modifications.

High-level assemblers

More sophisticated high-level assemblers provide language abstractions such as:

- High-level procedure/function declarations and invocations
- Advanced control structures
- High-level abstract data types, including structures/records, unions, classes, and sets
- Sophisticated macro processing (although available on ordinary assemblers since the late 1950s for IBM 700 series and since the 1960s for IBM/360, amongst other machines)
- Object-oriented programming features such as classes, objects, abstraction, polymorphism, and inheritance

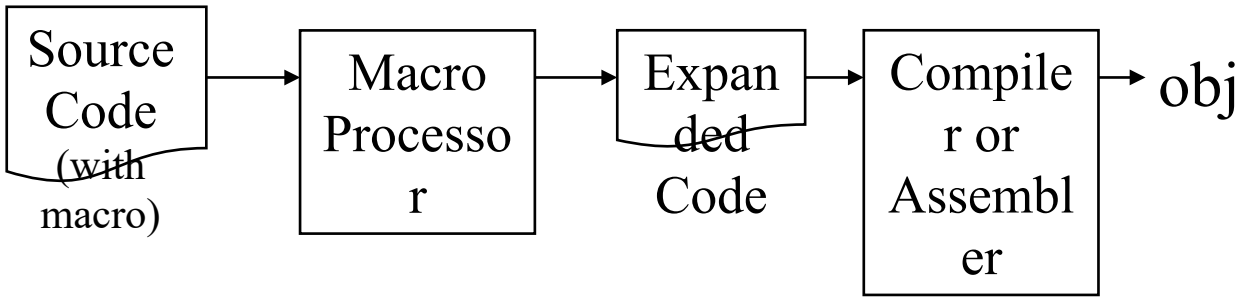
Multiple sets of mnemonics or assembly-language syntax may exist for a single instruction set, typically instantiated in different assembler programs. In these cases, the most popular one is usually that supplied by the manufacturer and used in its documentation.



3)Macroprocessor:

A general-purpose macro processor or general purposepreprocessor is a macro processor that is not tied to or integrated with a particular language or piece of software.A macro processor is a program that copies a stream of text from one place to another, making a systematic set of replacements as it does so. Macro processors are often embedded in other programs, such as assemblers and compilers. Sometimes they are standalone programs that can be used to process any kind of text.

Macro processors have been used for language expansion (defining new language constructs that can be expressed in terms of existing language components), for systematic text replacements that require decision making, and for text reformatting (e.g. conditional extraction of material from an HTML file).



4)Linker:

In computing, a linkeris a computer program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another object file.

A simpler version that writes its output directly to memory is called the loader, though loading is typically considered a separate process.

Computer programs typically comprise several parts or modules; these parts/modules need not all be contained within a single object file, and in such cases refer to each other by means of symbols. Typically, an object file can contain three kinds of symbols:

- defined "external" symbols, sometimes called "public" or "entry" symbols, which allow it to be called by other modules,
- undefined "external" symbols, which reference other modules where these symbols are defined, and
- local symbols, used internally within the object file to facilitate relocation.

For most compilers, each object file is the result of compiling one input source code file. When a program comprises multiple object files, the linker combines these files into a unified executable program, resolving the symbols as it goes along.

Dynamic linking

Many operating system environments allow dynamic linking, that is the postponing of the resolving of some undefined symbols until a program is run. That means that the executable code still contains undefined symbols, plus a list of objects or libraries that will provide definitions for these. Loading the program will load these objects/libraries as well, and perform a final linking. Dynamic linking needs no linker.

This approach offers two advantages:

- Often-used libraries (for example the standard system libraries) need to be stored in only one location, not duplicated in every single binary.
- If a bug in a library function is corrected by replacing the library, all programs using it dynamically will benefit from the correction after restarting them. Programs that included this function by static linking would have to be re-linked first.

There are also disadvantages:

- Known on the Windows platform as "DLL Hell", an incompatible updated library will break executables that depended on the behavior of the previous version of the library if the newer version is not properly backwards compatible.

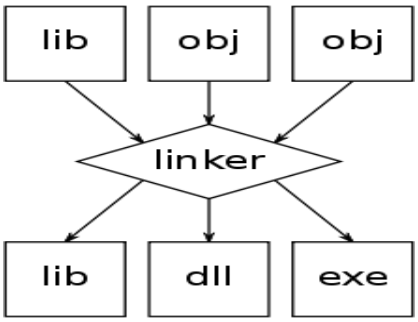
- A program, together with the libraries it uses, might be certified (e.g. as to correctness, documentation requirements, or performance) as a package

Static linking

Static linking is the result of the linker copying all library routines used in the program into the executable image. This may require more disk space and memory than dynamic linking, but is more portable, since it does not require the presence of the library on the system where it is run.

Linkage editor

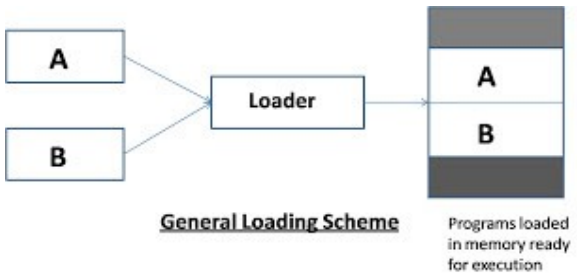
In IBMSystem/360mainframe environments such as OS/360, including z/OS for the z/Architecture mainframes, this type of program is known as a linkage editor. However, a linkage editor, as the name implies, has the additional capability of allowing the addition, replacement, and/or deletion of individual program sections.



5)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.

In the case of operating systems that support virtual memory, the loader may not actually copy the contents of executable files into memory, but rather may simply declare to the virtual memory subsystem that there is a mapping between a region of memory allocated to contain the running program's code and the contents of the associated executable file. This may mean parts of a program's code are not actually copied into memory until they are actually used, and unused code may never be loaded into memory at all.



6)Operating System:

An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs. The operating system is a component of the system software in a computer system. Application programs usually require an operating system to function.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.Examples of modern operating systems include Apple OS X, Linux and its variants, and Microsoft Windows.

Types of operating systems

a)Single and multi-tasking

A single-tasking system can only run one program at a time, while a multi-tasking operating system allows more than one program to be running in concurrency. This is achieved by time-sharing, dividing the available processor time between multiple processes which are each interrupted repeatedly in time-slices by a task scheduling subsystem of the operating system. Multi-tasking may be characterized in preemptive and co-operative types

b)Singleand multi-user

Single-user operating systems have no facilities to distinguish users, but may allow multiple programs to run in tandem.A multi-user operating system extends the basic concept of multi-tasking with facilities that identify processes and resources, such as disk space, belonging to multiple users, and the system permits multiple users to interact with the system at the same time. Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources to multiple users.

c)Distributed

A distributed operating system manages a group of distinct computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they form a distributed system.

d)Templated

In an OS, distributed and cloud computing context, templating refers to creating a single virtual machine image as a guest operating system, then saving it as a tool for multiple running virtual machines (Gagne, 2012, p. 716). The technique is used both in virtualization and cloud computing management, and is common in large server warehouses.

e)Embedded

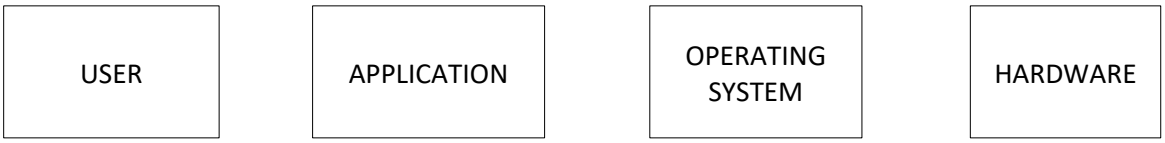
Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design. Windows CE and Minix 3 are some examples of embedded operating systems.

f)Real-time

A real-time operating system is an operating system that guarantees to process events or data within a certain short amount of time. A real-time operating system may be single- or multi-tasking, but when multitasking, it uses specialized scheduling algorithms so that a deterministic nature of behavior is achieved. An event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts.

g)Library

A library operating system is one in which the services that a typical operating system provides, such as networking, are provided in the form of libraries. These libraries are composed with the application and configuration code to construct unikernels — which are specialised, single address space, machine images that can be deployed to cloud or embedded environments.



7)INTERPRETER:

In computer science, an interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming 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 codemade by a compiler which is part of the interpreter system.

While interpretation and compilation are the two main means by which programming languages are implemented, they are not mutually exclusive, as most interpreting systems also perform some translation work, just like compilers. The terms "interpreted language" or "compiled language" signify that the canonical implementation of that language is an interpreter or a compiler, respectively. A high level language is ideally an abstraction independent of particular implementations

Comparison between compiler and interpreter and assembler:

Assembler:
Assembler is a computer program which is used to translate program written in Assembly Language in to machine language. The translated program is called as object program. Assembler checks each instruction for its correctness and generates diagnostic messages, if there are mistakes in the program. Various steps of assembling are:

1. Input source program in Assembly Language through an input device.
2. Use Assembler to produce object program in machine language.
3. Execute the program.

Compiler:
A compiler is a program that translates a program written in HLL to executable machine language. The process of transferring HKK source program in to object code is a lengthy and complex process as compared to assembling. Compliers have diagnostic capabilities and prompt the programmer with appropriate error message while compiling a HLL program. The corrections are to be incorporated in the program, whenever needed, and the program has to be recompiled. The process is repeated until the program is mistake free and translated to an object code. Thus the job of a complier includes the following:
1.To translate HLL source program to machine codes.
2. To trace variables in the program
3. To include linkage for subroutines.
4. To allocate memory for storage of program and variables.
5. To generate error messages, if there are errors in the program.

Interpreter:
The basic purpose of interpreter is same as that of complier. In compiler, the program is translated completely and directly executable version is generated. Whereas interpreter translates each instruction, executes it and then the next instruction is translated and this goes on until end of the program. In this case, object code is not stored and reused. Every time the program is executed, the interpreter translates each instruction freshly. It also has program diagnostic capabilities. However, it has some disadvantages as below:
1. Instructions repeated in program must be translated each time they are executed.
2. Because the source program is translated fresh every time it is used, it is slow process or execution takes more time. Approx. 20 times slower than complier.