system software

System software is a type of computer program that is designed to run a computer's hardware and application programs. If we think of the computer system as a layered model, the system software is the interface between the hardware and user applications. The operating system is the best-known example of system software. The OS manages all the other programs in a computer.

System software is used to manage the computer itself. It runs in the background, maintaining the computer's basic functions so users can run higher-level application software to perform certain tasks. Essentially, system software provides a platform for application software to be run on top of.

Important features of system software

Computer manufacturers usually develop the system software as an integral part of the computer. The primary responsibility of this software is to create an interface between the computer hardware they manufacture and the end user.

System software generally includes the following features:

1. **High speed.** System software must be as efficient as possible to provide an effective platform for higher-level software in the computer system.
2. **Hard to manipulate.** It often requires the use of a programming language, which is more difficult to use than a more intuitive user interface (UI).
3. **Written in a low-level computer language.** System software must be written in a computer language the central processing unit (CPU) and other computer hardware can read.
4. **Close to the system.** It connects directly to the hardware that enables the computer to run.
5. **Versatile.** System software must communicate with both the specialized hardware it runs on and the higher-level application

software that is usually hardware-agnostic and often has no direct connection to the hardware it runs on. System software also must support other programs that depend on it as they evolve and change.

What is application software?

Application software is a type of computer program that performs a specific personal, educational, and business function. Each program is designed to assist the user with a particular process, which may be related to productivity, creativity, and/or communication.

Functions of Application Software

Application software programs are created to facilitate a variety of functions, including but not limited to:

- managing information
- manipulating data
- constructing visuals
- coordinating resources
- calculating figures

Examples of Application Software

The most common application software programs are used by millions every day and include:

- Microsoft suite of products (Office, Excel, Word, PowerPoint, Outlook, etc.)
- Internet browsers like Firefox, Safari, and Chrome
- Mobile pieces of software such as Pandora (for music appreciation), Skype (for real-time online communication), and Slack (for team collaboration)

Operating System

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management −

- Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management −

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

Other Important Activities

Following are some of the important activities that an Operating System performs −

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** − Recording delays between request for a service and response from the system.
- **Job accounting** − Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** − Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other softwares and users** − Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

UTILITY PROGRAMS

A utility program is system software that helps users to analyse, configure, monitor, or help maintain their computers. Most operating systems include a set of basic utilities for users, and additional utilities that could be downloaded if needed. Examples of utilities include:

- back-up software that helps a user create back-up copies of the files on their computer
- a device manager that helps a user install new hardware such as a mouse, USB, etc.

disk cleaners that helps a user to free up space on a storage device

- file managers that allow users to manage the files that are stored on their computers
- system (Task Managers) monitors that summarise a computer’s performance for the user.

Without these utilities it would be a lot harder for users to manage and keep their computers running optimally.

DEVICE DRIVER PROGRAMS

A device driver is software that contains a set of instructions that command a computer’s operating system on how to communicate with the hardware so that it can function properly. Device drivers allow communication between the operating system and all the devices, such as the mouse, keyboard, printer, etc.

The field of IT is forever changing, so it would be impossible to create an operating system that knows how each device functions or works (especially devices those that have not yet been invented). It is for this reason that each hardware manufacturer is responsible for developing drivers for their own manufactured devices.

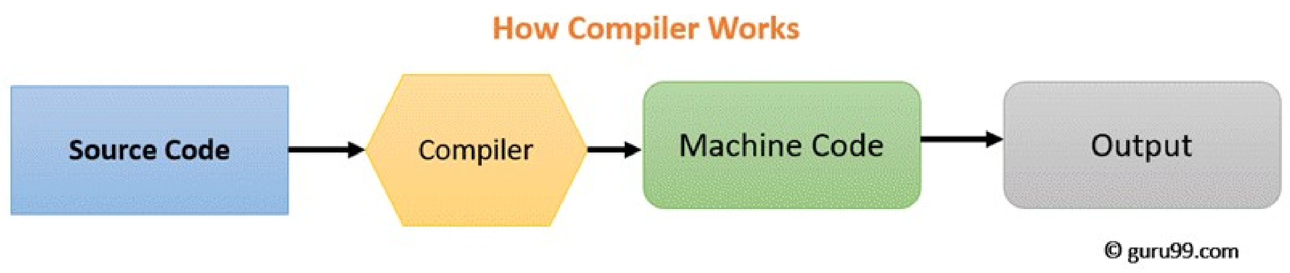
What is Compiler?

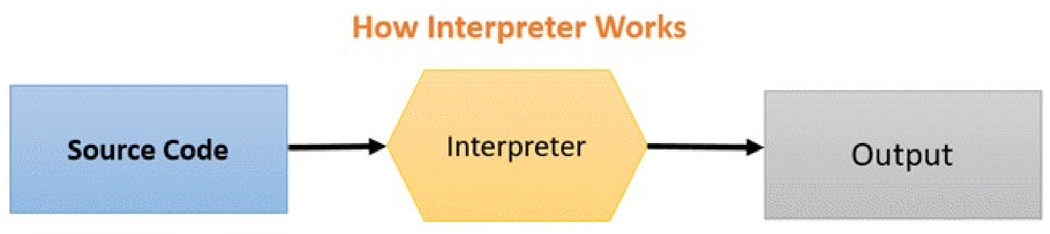
A compiler is a computer program that transforms code written in a high-level programming language into the machine code. It is a program which translates the human-readable code to a language a computer processor understands (binary 1 and 0 bits). The computer processes the machine code to perform the corresponding tasks.

A compiler should comply with the syntax rule of that programming language in which it is written. However, the compiler is only a program and cannot fix errors found in that program. So, if you make a mistake, you need to make changes in the syntax of your program. Otherwise, it will not compile.

What is Interpreter?

An interpreter is a computer program, which coverts each high-level program statement into the machine code. This includes source code, pre-compiled code, and scripts. Both compiler and interpreters do the same job which is converting higher level programming language to machine code. However, a compiler will convert the code into machine code (create an exe) before program run. Interpreters convert code into machine code when the program is run.





KEY DIFFERENCE

- Compiler transforms code written in a high-level programming language into the machine code, at once, before program runs, whereas an Interpreter coverts each high-level program statement, one by one, into the machine code, during program run.
- Compiled code runs faster while interpreted code runs slower.
- Compiler displays all errors after compilation, on the other hand, the Interpreter displays errors of each line one by one.
- Compiler is based on translation linking-loading model, whereas Interpreter is based on Interpretation Method.
- Compiler takes an entire program whereas the Interpreter takes a single line of code.

Difference Between Compiler and Interpreter

Basis of difference	Compiler	Interpreter
Programming Steps	• Create the program. • Compile will parse or analyses all of the language statements for its correctness. If incorrect, throws an error • If no error, the compiler will convert source code to machine code. • It links different code files into a runnable	• Create the Program • No linking of files or machine code generation • Source statements executed line by line DURING Execution

Basis of differenc e	**Compiler**	**Interpreter**
	program(know as exe) • Run the Program	
Advantage	The program code is already translated into machine code. Thus, it code execution time is less.	Interpreters are easier to use, especially for beginners.
Disadvantage	You can't change the program without going back to the source code.	Interpreted programs can run on computers that have the corresponding interpreter.
Machine code	Store machine language as machine code on the disk	Not saving machine code at all.
Running time	Compiled code run faster	Interpreted code run slower
Model	It is based on language translationlinking-loading model.	It is based on Interpretation Method.
Program generation	Generates output program (in the form of exe) which can be run independently from the original program.	Do not generate output program. So they evaluate the source program at every time during execution.
Execution	Program execution is separate from the compilation. It performed only after the entire output program is compiled.	Program Execution is a part ofInterpretation process, so it is performed line by line.
Memory requirement	Target program executeindependently and do not require the compiler in the memory.	The interpreter exists in the memory during interpretation.
Best suited for	Bounded to the specific target machine and cannot be ported. C and C++ are a most popular a programming language which uses compilation model.	For web environments, where load times are important. Due to all the exhaustive analysis is done, compiles take relatively larger time to compile even small code that may not be run multiple times. In such cases, interpreters are better.
Code Optimization	The compiler sees the entire code upfront. Hence, they perform lots of optimizations that make code run faster	Interpreters see code line by line, and thus optimizations are not as robust as compilers
Dynamic Typing	Difficult to implement as compilers cannot predict what happens at turn time.	Interpreted languages support Dynamic Typing
Usage	It is best suited for the Production Environment	It is best suited for the program and developmentenvironment.
Error execution	Compiler displays all errors and warning at the compilation time.	The interpreter reads a single statement and shows the error if any. You must correct the

Basis of difference	Compiler	Interpreter
	Therefore, you can't run the program without fixing errors	error to interpret next line.
Input	It takes an entire program	It takes a single line of code.
Output	Compliers generates intermediate machnie code.	Interpreter never generate any intermediate machnie code.
Errors	Display all errors after, compilation, all at the same time.	Displays all errors of each line one by one.
Pertaining Programming languages	C,C++,C#, Scala, Java all use complier.	PHP, Perl, Ruby uses an interpreter.

Role of Compiler

- Compliers reads the source code, outputs executable code
- Translates software written in a higher-level language into instructions that computer can understand. It converts the text that a programmer writes into a format the CPU can understand.
- The process of compilation is relatively complicated. It spends a lot of time analyzing and processing the program.
- The executable result is some form of machine-specific binary code.

Role of Interpreter

- The interpreter converts the source code line-by-line during RUN Time.
- Interpret completely translates a program written in a high-level language into machine level language.
- Interpreter allows evaluation and modification of the program while it is executing.
- Relatively less time spent for analyzing and processing the program
- Program execution is relatively slow compared to compiler

HIGH-LEVEL LANGUAGES

High-level languages, like C, C++, JAVA, etc., are very near to English. It makes programming process easy. However, it must be translated into machine language before execution. This translation process is either conducted by either a compiler or an interpreter. Also known as source code.

MACHINE CODE

Machine languages are very close to the hardware. Every computer has its machine language. A machine language programs are made up of series of binary pattern. (Eg. 110110) It represents the simple operations which should be performed by the computer. Machine language programs are executable so that they can be run directly.

OBJECT CODE

On compilation of source code, the machine code generated for different processors like Intel, AMD, an ARM is different. tTo make code portable, the source code is first converted to Object Code. It is an intermediary code (similar to machine code) that no processor will understand. At run time, the object code is converted to the machine code of the underlying platform.

What Is a Software License?

A software license is a contract between the entity that created and supplied an application, underlying source code, or related product and its end user. The license is a text document designed to protect the intellectual property of the software developer and to limit any claims against them that may arise from its use.

A software license also provides legally binding definitions for the distribution and use of the software. End-user rights, such as installation, warranties, and liabilities, are also often spelled out in the software license, including protection of the developer's intellectual property.

Most software falls under one of two categories that have distinct differences in how they are viewed under copyright law:

- **Proprietary** – also referred to as "closed source"
- **Free and open-source software (FOSS)** – referred to as "open source"

FOSS software licenses – give rights to the customer that include modification and reuse of the software code, providing the actual source code with the software product(s). This open-source type of licensing affords the user authority to modify the software functions and freedom to inspect the software code.

Proprietary software licenses – provide no such authority for code modification or reuse and normally provide software with operational code only, and no source code. A proprietary software license often includes terms that prohibit “reverse engineering” of the object code with the intention of obtaining source code by the licensee.

In both cases, the software license will most often specify limitations of liability from use of the software product, any mutual responsibilities such as support, and any warranties or disclaimer of warranty.

Where software is not covered by any license, it is normally categorized as:

- **Public domain software –** freely available for use and not copyright protected
- **Private unlicensed software –** such as business applications that still falls under copyright protection

Open source and proprietary software licensing may also specify additional restrictions and terms:

- Transfer of ownership to the buyer or retention of ownership by the seller
- Any authorization for copying, selling, or distributing the software
- Definition of whether the license constitutes purchase or leasing of the software

How Does Software Licensing Work?

New users of a software will normally enter into an end-user license agreement (EULA) that constitutes a legal definition of the relationship between the licensor (provider) and licensee (user or business). The EULA is a contract that establishes the rights of the purchaser for installing and using the software.

Every EULA contains a clause that stipulates when its conditions are activated by an end user. This may be the moment the user opens the product packaging or, for example, when the user clicks on a button agreeing to accept the EULA’s terms to access it.

Cloud-based applications such as Software as a Service (SaaS) will often include license details in EULAs including:

- Monthly or annual charges per user
- Duration of the agreement
- Terms of cancellation of the agreement
- Recovery of any charges if canceled during the agreement

An additional use of software licensing is in cases where a software developer or firm grants authority for selling or distributing the software under the second party’s brand. The developer retains ownership, but the re-branding company is permitted to resell the software product. This method of licensing is called “white labeling.”

What Are the Types of Software Licenses?

There are five main software license categories or types used to cover different kinds of software and various business arrangements. These encompass a wide spectrum of licensing scenarios, from free software (public domain) to paid commercial software (proprietary).

Between these two extremes, there are also three categories (GNU/LGPL, permissive, and copyleft) that apply to various forms of open-source projects. Failure to follow the terms and conditions of an open-source license can lead to revealing trade secrets or even legal action from the project's developers.

5 Types of Software Licenses You Need to Know About

1. Public Domain License

When software is defined as being in the public domain, anyone is free to use and modify the software without restrictions. This is a "permissive" license that allows adopting the code into applications or projects and reusing the software as desired.

For many reasons, businesses must exercise caution when adopting public domain software in projects or other important applications:

- Public domain software may not always adhere to best coding practices or may not be up to standards of secure software that the application requires.
- Software that does not fall under specific licensing terms is not always public domain code. Be sure the software is truly public domain before copying, reusing, or distributing it.

2. GNU/LGPL – GNU Lesser General Public License (LGPL)

Under an LGPL license, developers have rights to link to open source libraries within their own software. Resulting code can be licensed under any other type of license – even proprietary – when projects are compiled or linked to include an LGPL-licensed library.

The caveat is that if any part of the library is copied into the code or modified, the terms of the original LGPL license will apply to the developed code that used the library.

3. Permissive

This type of license is one of the most common and popular among open-source software licenses. Under a permissive license – also referred to as "Apache" or "BSD style" – there are few restrictions or requirements for the distribution or modifications of the software. Another variation of a permissive software license is the "MIT" license.

Variants in permissive licenses include differences in requirements for preserving license notices and copyrights for the software, as well as how the software may be used (commercial or private), trademark requirements, and other stipulations.

4. Copyleft

This license's terms are restrictive – known as reciprocal licenses. Under the terms of a copyleft license, the licensed code may be modified or distributed as part of a software project if the new code is distributed under the same software license.

This means that if the code included in the software product was specified to be for "personal use only," the new product being distributed must carry that same designation/restriction.

Since the original software included with the new project allowed modifications and distribution, this may not be the best license for software developers because the resulting code must also carry the copyleft license type – including the availability of the source code.

5. Proprietary

These software licenses make the software ineligible for copying, modifying, or distribution. This is the most restrictive type of software license, protecting the developer or owner from unauthorized use of the software.

What Is a Software License Agreement?

A software license agreement is a legal document that stipulates several key conditions between a software company or developer and a user to allow use of the software.

These conditions are designed to protect the developer's intellectual property rights and to limit claims against them for potential damage resulting from use of their software. In some cases, pricing and terms of payment may also be included, though this is often covered in a separate document. The primary purpose of the agreement, however, is to provide detailed ground rules for use of the software:

- Where the software may be installed and how many instances may be installed.
- How the software can be used.
- Whether the software may be copied, modified, or redistributed.
- Any copyrights that apply to the software.
- Ownership of the software—most often specifying that the provider retains all rights of ownership.
- Duration of the terms of the agreement.
- What constitutes correct usage of the software.

What Are Software Licenses Used for?

Developers release software for a number of reasons, whether it be to demonstrate a new idea, provide benefit to as many people as possible, or for financial and economic gain. In order to ensure that all parties involved in the process are able to benefit from the software, the terms and conditions for its use must be clearly defined.

These terms and conditions are expressed as a licensing agreement. Software licenses are critical for software providers and users alike for many reasons:

- Written authority for use of software – protects business users and individuals from liability and copyright infringement claims
- Clarification of the number of eligible users of the software
- Definition of what is included – maintenance, upgrades, support
- Warranty agreements and problem mediation process
- Distribution permission and limitations

- Use rights, such as copying or modifications
- Copyright definition, including software and any documentation
- Dates – for installation, training, support assistance, and license duration
- Termination terms, penalties, financial liabilities
- Any performance guarantees and remedies

Software licenses define the complete agreement between the licensor and licensee. The goal is to clarify the relationship from both a legal and technical viewpoint, so there are no surprises or guesswork regarding responsibilities while the agreement is in effect.

For mobile software, the license terms state how much of the user's sensitive personal data stored on the device an application vendor is permitted to access. These agreements are designed to protect personal information such as financial statements, location, or health data and prevent its misuse.

How Much Does a Software License Cost?

Software license pricing varies widely, depending on the type of software, how it is provided, and the supplier's cost to develop the software. SaaS providers typically provide a subscription model where the charge is based on the number of users. This grants businesses a great deal of cost control and flexibility.

Software license pricing will vary greatly among software providers that offer complete on-premises business solutions as opposed to those providing open-source objects for building internal applications and web functions. While these costs may seem unnecessary, much like an insurance policy, it will provide protection when you actually need it.