Operating System Tutorial provides the basic and advanced concepts of operating system . Our Operating system tutorial is designed for beginners, professionals and GATE aspirants. We have designed this tutorial after the completion of a deep research about every concept.

The content is described in detailed manner and has the ability to answer most of your queries. The tutorial also contains the numerical examples based on previous year GATE questions which will help you to address the problems in a practical manner.

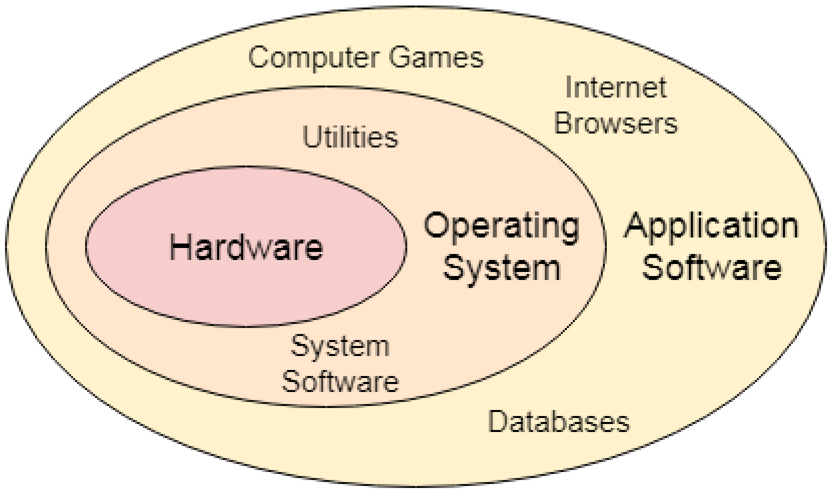
Operating System can be defined as an interface between user and the hardware. It provides an environment to the user so that, the user can perform its task in convenient and efficient way.

The Operating System Tutorial is divided into various parts based on its functions such as Process Management, Process Synchronization, Deadlocks and File Management.

Operating System Definition and Function

In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user.

We need a system which can act as an intermediary and manage all the processes and resources present in the system.



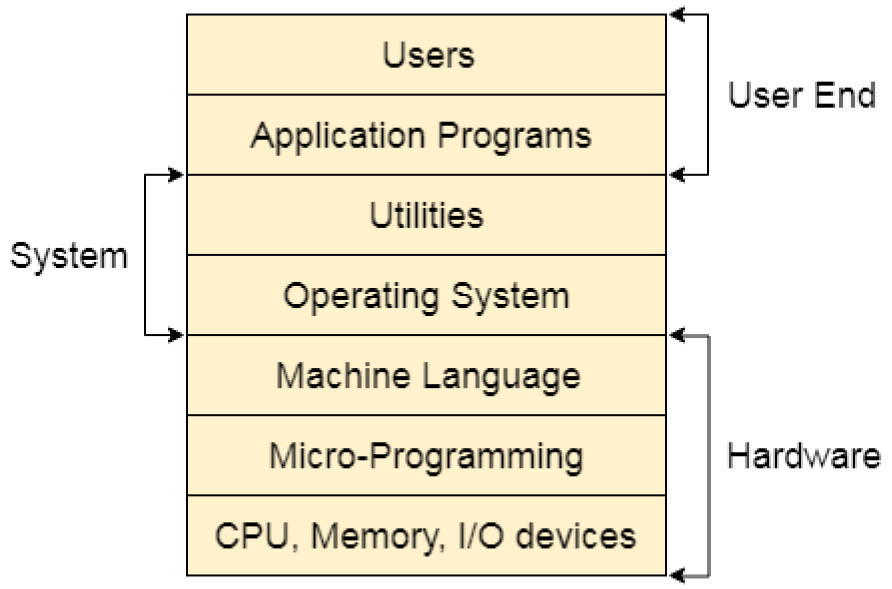
An **Operating System** can be defined as an **interface between user and hardware**. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.

The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.

Structure of a Computer System

A Computer System consists of:

- Users (people who are using the computer)
- Application Programs (Compilers, Databases, Games, Video player, Browsers, etc.)
- System Programs (Shells, Editors, Compilers, etc.)
- Operating System (A special program which acts as an interface between user and hardware)
- Hardware (CPU, Disks, Memory, etc)



What does an Operating system do?

1. Process Management
2. Process Synchronization
3. Memory Management
4. CPU Scheduling

5. File Management

6. Security

7. As we know that software is a set of instructions or programs instructing a computer to do specific tasks. Software is basically a generic term used to describe computer programs. In general Scripts, applications, programs and a set of instructions are all terms often used to describe software.
8. Now the basis of language in which software is developed and platform which is required for its execution we can classified software as in two divisions which are System software and Application software. Following are some basic differences between System software and Application software.

Sr. No.	Key	System Software.	Application Software.
1	Definition	System Software is the type of software which is the interface between application software and system.	On other hand Application Software is the type of software which runs as per user request. It runs on the platform which is provide by system software.
2	Development Language	In general System software are developed in low level language which is more compatible with the system hardware in order to interact with.	While in case of Application software high level language is used for their development as they are developed as some specific purpose software.
3	Usage	System software is used for operating computer hardware.	On other hand Application software is used by user to perform specific task.

Sr. No.	Key	System Software.	Application Software.
4	Installation	System software are installed on the computer when operating system is installed.	On other hand Application software are installed according to user’s requirements.
5	User interaction	As mentioned in above points system software are specific to system hardware so less or no user interaction available in case of system software.	On other hand in application software user can interacts with it as user interface is available in this case.
6	Dependency	System software can run independently. It provides platform for running application software.	On other hand in application software can’t run independently. They can’t run without the presence of system software..
7	Examples	Some examples of system software’s are compiler, assembler, debugger, driver, etc.	On other hand some examples of application software’s are word processor, web browser, media player, etc.

Translators
The most general term for a software code converting tool is “translator.” A translator, in software programming terms, is a generic term that could refer to a compiler, assembler, or interpreter; anything that converts higher level code into another high-level code (e.g., Basic,

C++, Fortran, Java) or lower-level (i.e., a language that the processor can understand), such as assembly language or machine code. If you don't know what the tool actually does other than that it accomplishes some level of code conversion to a specific target language, then you can safely call it a translator.

Compilers
Compilers convert high-level language code to machine (object) code in one session. Compilers can take a while, because they have to translate high-level code to lower-level machine language all at once and then save the executable object code to memory. A compiler creates machine code that runs on a processor with a specific Instruction Set Architecture (ISA), which is processor-dependent. For example, you cannot compile code for an x86 and run it on a MIPS architecture without a special compiler. Compilers are also platform-dependent. That is, a compiler can convert C++, for example, to machine code that's targeted at a platform that is running the Linux OS. A cross-compiler, however, can generate code for a platform other than the one it runs on itself.

A cross-compiler running on a Windows machine, for instance, could generate code that runs on a specific Windows operating system or a Linux (operating system) platform. Source-to-source compilers translate one program, or code, to another of a different language (e.g., from Java to C). Choosing a compiler then, means that first you need to know the ISA, operating system, and the programming language that you plan to use. Compilers often come as a package with other tools, and each processor manufacturer will have at least one compiler or a package of software development tools (that includes a compiler). Often the software tools (including compiler) are free; after all, a CPU is completely useless without software to run on it. Compilers will report errors after compiling has finished.

Interpreters
Another way to get code to run on your processor is to use an interpreter, which is not the same as a compiler. An interpreter translates code like a compiler but reads the code and immediately executes on that code, and therefore is initially faster than a compiler. Thus, interpreters are often used in software development tools as debugging tools, as they can execute a single in of code at a time. Compilers translate code all at once and the processor then executes upon the machine language that the compiler produced. If changes are made to the code after compilation, the changed code will need to be compiled and added to the compiled code (or perhaps the entire program will need to be re-compiled.) But an interpreter, although skipping the step of compilation of the entire program to start, is much slower to execute than the same program that's been completely compiled.

Interpreters, however, have usefulness in areas where speed doesn't matter (e.g., debugging and training) and it is possible to take the entire interpreter and use it on another

ISA, which makes it more portable than a compiler when working between hardware architectures. There are several types of interpreters: the syntax-directed interpreter (i.e., the Abstract Syntax Tree (AST) interpreter), bytecode interpreter, and threaded interpreter (not to be confused with concurrent processing threads), Just-in-Time (a kind of hybrid interpreter/compiler), and a few others. Instructions on how to build an interpreter can be found on the web.[i] Some examples of programming languages that use interpreters are Python, Ruby, Perl, and PHP.

Assemblers

An assembler translates a program written in assembly language into machine language and is effectively a compiler for the assembly language, but can also be used interactively like an interpreter. Assembly language is a low-level programming language. Low-level programming languages are less like human language in that they are more difficult to understand at a glance; you have to study assembly code carefully in order to follow the intent of execution and in most cases, assembly code has many more lines of code to represent the same functions being executed as a higher-level language. An assembler converts assembly language code into machine code (also known as object code), an even lower-level language that the processor can directly understand.

Assembly language code is more often used with 8-bit processors and becomes increasingly unwieldy as the processor's instruction set path becomes wider (e.g., 16-bit, 32-bit, and 64-bit). It is not impossible for people to read machine code, the strings of ones and zeros that digital devices (including processors) use to communicate, but it's likely only read by people in cases of computer forensics or brute-force hacking. Assembly language is the next level up from machine code, and is quite useful in extreme cases of debugging code to determine exactly what's going on in a problematic execution, for instance. Sometimes compilers will "optimize" code in unforeseen ways that affect outcomes to the bafflement of the developer or programmer such that it's necessary to carefully follow the step-by-step action of the processor in assembly code, much like a hunter tracking prey or a detective following clues

UNIX	MS-DOS	Command Function
--	cd -	Switch between current and last directory
cat	Type	Displays the contents of a file
cd	Cd	Moves from one directory to another
cd /u01/test	cd c:\u01\test	Change directory paths
cd ..	cd..	Go up in directory
chmod	Attrib	Sets file permissions
clear	Cls	Clear the screen
cp	copy	Copies a file (or a group of files)
diff	fc	Compare two files
cpio	xcopy	Backs up and recovers files

UNIX	MS-DOS	Command Function
date	date	Display the system date
doskey	<ctl> k (3)	Display command history
export PS1=?xx?	prompt	Change the command prompt text
find	grep	Find a character string in a file
gzip	dblspace	Compress a data file
ln	--	Forms a link to a file
lp	print	Queues a file for printing
lpstat	print	Displays the printing queue
ls -al	dir	Displays the contents of a directory
mem	lsdev (2)	Display RAM memory
mkdir	md	Creates a new subdirectory
move	cp (4)	Move a file to another directory
mv	rename	Renames a file
rm	del	Deletes a file (or group of files)
rmdir	rd	Deletes an existing directory
setenv (1)	set	Set an environment variable
sort	sort	Sorts lines in a file
ver	uname -a	Display OS version
Vi	edit	Creates and edits text

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system −

- Program execution
- I/O operations
- File System manipulation
- Communication
- Error Detection
- Resource Allocation
- Protection

Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

File system manipulation

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.