**CSC 201: OPERATING SYSTEM**

**General Overview**

Without its software, a computer is basically a useless lump of metal. With its software, a computer can store, process, and retrieve information; play music, videos, sends email, search the internet; and engage in many other valuable activities to earn its keep. Computer software can be divided roughly into two kinds: system software which manages the operation of the computer itself, and application software, which performs the actual work the user wants. The most fundamental system software is the Operating Systems, whose job is to control all the computer’s resources and provide a base upon which the application software can be written.

Operating system are primarily resource managers; the main resource they manage is computer hardware in the form of processors, storage, input/output devices, communication devices, and data. They perform many functions such as implementing the user interface, sharing hardware among users, allowing users to share data among themselves, preventing users from interfering with one another, scheduling resources among users, facilitating input/output, recovering from errors, accounting for resources usage, facilitating parallel operations, organizing data for secure and rapid access and handling network communications.

**Definition of Operating System**

An **operating system** (commonly abbreviated *OS* and *O/S*) is the infrastructure software component of a computer system that is responsible for the management and coordination of activities and the sharing of the limited resources of the computer. An operating system is the set of programs that controls a computer. The operating system acts as a host for applications that are run on the machine. As a host, one of the purposes of an operating system is to handle the details of the operation of the hardware. This relieves application programs from having to manage these details and makes it easier to write applications. Operating Systems can be viewed from two points of views: **Resource manager** and **Extended machines**. From Resource manager point of view, Operating Systems manage the different parts of the system efficiently and from extended machines point of view, Operating Systems provide a virtual machine to users that is more convenient to use.

**Goals of OS**

OS can further be described by what they do i.e. by their functions, goals and objectives. Therefore, we will quickly run you through some of the goals of the OS which are:

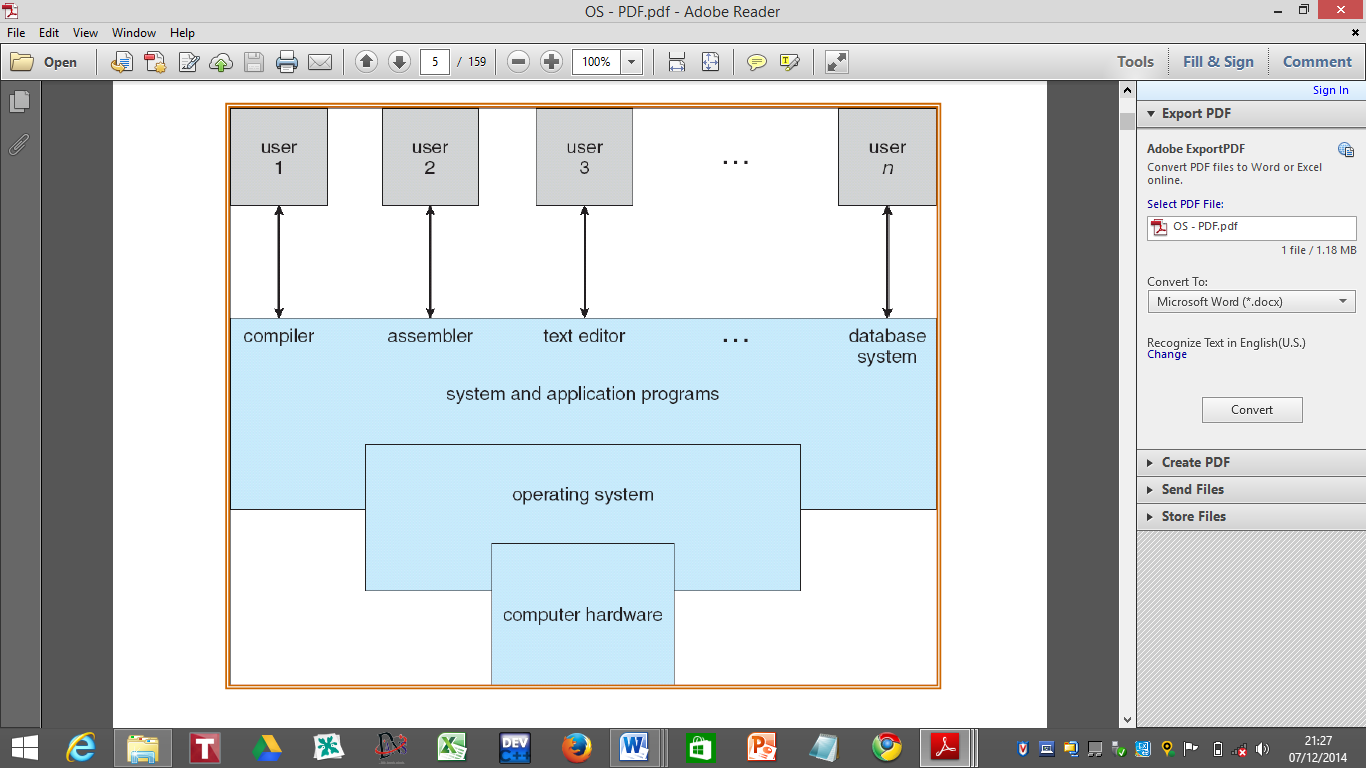
1. **Convenience for the User -** When there is no OS, users of computer system will need to write machine-level program in order to manipulate the hardware. With OS, users can now easily and conveniently use the computer with no stress of directly programming the hardware. OS provide a convenient interface for using the computer system.
2. **Efficiency -** An OS allows computer system resources to be used in an efficient manner. This is particularly important for large shared multi-user systems which are usually expensive. In the past, the efficiency (i.e. optimal use of the computer resources) considerations were often more important than convenience.
3. **Evolutionary Capabilities -** Ability to evolve also happens to be one of the goals of the OS. An OS should be constructed in such a way that it permits the effective development, testing and introduction of new system functions without interfering with its service.

**Views of OS**

OS can be viewed from the perspective of what they are. These views are diverse depending on the particular view point of a user. But some of these views are discussed below.

1. **OS as a User/Computer Interface -** A computer system can be viewed as a layered or hierarchical structure consisting of the hardware, operating system, utilities, application programs and users. The users of application programs are called the end-users and are generally not concerned with the computer’s architecture. The end-user views the computer system in terms of an application. The application is developed by the application programmer who uses a programming language and a language translator. A set of programs called the utilities is provided to assist the programmer in program creation, file management and the control of Input/Output (I/O) devices.

The most important system program, operating system masks the details of the hardware from the programmer and provides a convenient interface for using the system. It acts as a mediator, making it easier for the programmer and for application programs to access and use the available services and facilities.



***Abstract View of the Components of a Computer System***

1. **OS as a Resource Manager -** A computer system has a set of resources for the movement, storage and processing of data. The OS is responsible for managing these resources. Note that resources include CPU, file storage space, data, programs, memory space, I/O devices, etc.

The OS is like any other computer program in that it provides instructions for the processor. The key difference is in the purpose of the program. The OS directs the processor in the use of the other system resources and in the timing of its execution of other programs. The processor, in order to do any of these things, must cease execution of the OS program to execute other programs. Thus, the OS relinquishes control long enough to prepare the processor to do the next piece of work. The portion of the OS that is always in main memory is called the kernel or nucleus and it contains the most frequently used functions in the OS. The remainder of the main memory contains other user programs and data. The allocation of this resource (i.e. main memory) is controlled jointly by the OS and the memory management hardware in the processor.

1. **Services Provided by the OS -** The services provided by the OS can be categorised into two:
2. **Convenience for the Programmer/User -** The conveniences offered to the user are in diverse and following ways:
3. Program Creation: Although editors and debuggers are not part of the OS, they are accessed through the OS to assist programmers in creating programs.
4. Program Execution: OS ensures that programs are loaded into the main memory. I/O devices and files are initialised and other resources are prepared. The program must be able to end its execution either normally or abnormally. In case of abnormal end to a program, it must indicate error.
5. Access to I/O devices: Each I/O device requires its own set of instructions or control signal for operation. The OS takes care of the details so that the programmer can think in terms of reads and writes.
6. Controlled Access: In the case of files, control includes an understanding of the nature of the I/O device (e.g. diskette drive, CDROM drive, etc.) as well as the file format of the storage medium. OS deals with these details. In the case of the multi-user system, OS must provide protection mechanisms to control access to the files.
7. Communications: There are many instances in which a process needs to exchange information with another process. There are two major ways in which communication can occur:

* It can take place between processes executing on the same computer.
* It can take place between processes executing on different computer systems that are linked by a computer network.

Communications may be implemented via a shared memory or by a technique of message passing in which packets of information are moved between processes by the OS.

1. Error Detection: A variety of errors can occur while a computer system is running.

These errors include:

* CPU and memory or hardware error: This encompasses memory error, power failure, a device failure such as connection failure on a network, lack of paper in printer.
* Software errors: Arithmetic overflow, attempt to access forbidden memory locations, inability of the OS to grant the request of an application.

In each case, the OS must make a response that makes the less impact on running applications. The response may range from ending the program that caused the error, retrying the operation or simply reporting the error to the application.

1. **Efficiency of System: Single and Multi-User -** In the area of system efficiency, OS offer the following services:
2. System Access or Protection: In the case of a shared or public system, the OS controls access to the system and to specific system resources by ensuring that each user authenticates him/herself to the system, usually by means of passwords to be allowed access to system resources. It extends to defending external I/O devices including modems, network adapters from invalid access attempts and to recording all such connections for detection of break-ins.
3. Resources Allocation: In an environment where there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the OS. Some (such as CPU cycles, main memory and file storage) may have general request and release codes. For instances, in determining how best to use the CPU, the OS have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available and other factors. These routines may also be used to allocate plotters, modems and other peripheral devices.
4. Accounting: This helps to keep track of how much of and what types of computer resources are used by each user. Today, this record keeping is not for billing purposes but for simply accumulating usage statistics. This statistics may be available tool for researchers who want to reconfigure the system to improve computing services.
5. Ease of Evolution of OS: A major OS will evolve over time for a number of reasons such as hardware upgrades and new types of hardware e.g. The use of graphics terminals may affect OS design. This is because such a terminal may allow the user to view several applications at the same time through ‘windows’ on the screen. This requires more sophisticated support in the OS.
6. New Services: In response to user demands or the need of system managers, the OS may expand to offer new services.
7. Error correction: The OS may have faults which may be discovered over the course of time and fixes will need to be made.

Other features provided by the OS includes:

* Defining the user interface
* Sharing hardware among users
* Allowing users to share data
* Scheduling resources among users
* Facilitating I/O
* Recovering from errors

The OS interfaces with, programs, hardware, users such as administrative personnel, computer operators, application programmers, system programmers, etc.

**Objectives of Operating System**

Modern Operating systems generally have following three major goals. Operating systems generally accomplish these goals by running processes in low privilege and providing service calls that invoke the operating system kernel in high-privilege state.

1. **To hide details of hardware by creating abstraction -** An abstraction occurs when software hides lower level details and provides a set of higher-level functions. An operating system transforms the physical world of devices, instructions, memory, and time into virtual world that is the result of abstractions built by the operating system. There are several reasons for abstraction.

*First*, the code needed to control peripheral devices is not standardized. Operating systems provide subroutines called device drivers that perform operations on behalf of programs for example, input/output operations.

*Second*, the operating system introduces new functions as it abstracts the hardware. For instance, operating system introduces the file abstraction so that programs do not have to deal with disks.

*Third*, the operating system transforms the computer hardware into multiple virtual computers, each belonging to a different program. Each program that is running is called a process. Each process views the hardware through the lens of abstraction.

*Fourth*, the operating system can enforce security through abstraction.

1. **To allocate resources to processes (Manage resources) -** An operating system controls how **processes** (the active agents) may access **resources** (passive entities).
2. **Provide a pleasant and effective user interface -** The user interacts with the operating systems through the user interface and usually interested in the “look and feel” of the operating system. The most important components of the user interface are the command interpreter, the file system, on-line help, and application integration. The recent trend has been toward increasingly integrated graphical user interfaces that encompass the activities of multiple processes on networks of computers.

**How an Operating System Works**

When the power of computer is turned on, the first program that runs is usually a set of instructions kept in the computer's read- only memory (ROM). This code examines the system hardware to make sure everything is functioning properly. This **power-on self test** (POST) checks the CPU, memory and basic input-output system (BIOS) for errors and stores the result in a special memory location. Once the POST has successfully completed, the software loaded in ROM (sometimes called the BIOS or **firmware**) will begin to activate the computer's disk drives. In most modern computers, when the computer activates the hard disk drive, it finds the first piece of the operating system: the **bootstrap loader**.

The bootstrap loader is a small program that has a single function: It loads the operating system into memory and allows it to begin operation. In the most basic form, the bootstrap loader sets up the small driver programs that interface with and control the various hardware subsystems of the computer. It sets up the divisions of memory that hold the operating system, user information and applications. It establishes the data structures that will hold the myriad signals, flags and semaphores that are used to communicate within and between the subsystems and applications of the computer. Then it turns control of the computer over to the operating system. After basic processes have started, the OS runs user programs if available, otherwise enters the idle loop.

In the idle loop, OS performs some system management and profiling; OS halts the processor and enter in low-power mode. OS wakes up on the following conditions: interrupts from hardware devices or system calls from user programs.

**Historical Development of Operating System**

Historically, operating systems have been tightly related to the computer architecture, it is good idea to study the history of operating systems from the architecture of the computers on which they run. Operating systems have evolved through a number of distinct phases or generations which corresponds roughly to the decades.

**The 1940's - First Generation**

The earliest electronic digital computers had no operating systems. Machines of the time were so primitive that programs were often entered one bit at a time on rows of mechanical switches (plug boards). Programming languages were unknown (not even assembly languages). Operating systems were unheard of.

**The 1950's - Second Generation**

By the early 1950's, the routine had improved somewhat with the introduction of punch cards. The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701. The system of the 50's generally ran one job at a time. These were called **single-stream batch processing systems** because programs and data were submitted in groups or batches.

**The 1960's - Third Generation**

The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once. So operating systems designers developed the concept of **multiprogramming** in which several jobs are in main memory at once; a processor is switched from job to job as needed to keep several jobs advancing while keeping the peripheral devices in use. For example, on the system with no multiprogramming, when the current job paused to wait for other I/O operation to complete, the CPU simply sat idle until the I/O finished. The solution for this problem that evolved was to partition memory into several pieces, with a different job in each partition. While one job was waiting for I/O to complete, another job could be using the CPU.

Another major feature in third-generation operating system was the technique called spooling (simultaneous peripheral operations on line). In spooling, a high-speed device like a disk interposed between a running program and a low-speed device involved with the program in input/output. Instead of writing directly to a printer, for example, outputs are written to the disk. Programs can run to completion faster, and other programs can be initiated sooner when the printer becomes available, the outputs may be printed. Note that spooling technique is much like thread being spun to a spool so that it may be later be unwound as needed.

Another feature present in this generation was time-sharing technique, a variant of multiprogramming technique, in which each user has an on-line (i.e., directly connected) terminal. Because the user is present and interacting with the computer, the computer system must respond quickly to user requests, otherwise user productivity could suffer. Timesharing systems were developed to multiprogram large number of simultaneous interactive users.

**Fourth Generation**

With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the personal computer and the workstation age. Microprocessor technology evolved to the point that it became possible to build desktop computers as powerful as the mainframes of the 1970s. Two operating systems have dominated the personal computer scene: MS-DOS, written by Microsoft, Inc. for the IBM PC and other machines using the Intel 8088 CPU and its successors, and UNIX, which is dominant on the large personal computers using the Motorola 6899 CPU family.

**Types of Operating Systems**

OS can be categorised in different ways based on perspectives. Some of the major ways in which the OS can be classified are explored and introduced in this unit.

***Types of Operating Systems Based on the Types of Computer they Control and the Sort of Applications they Support***

Based on the types of computers they control and the sort of applications they support, there are generally four types within the broad family of operating systems. The broad categories are as follows:

1. **Real-Time Operating Systems (RTOS):** They are used to control machinery, scientific instruments and industrial systems. An RTOS typically has very little user-interface capability, and no end-user utilities, since the system will be a sealed box when delivered for use. A very important part of an RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time every time it occurs. In a complex machine, having a part move more quickly just because system resources are available may be just as catastrophic as having it not move at all because the system is busy. RTOS can be hard or soft. A hard RTOS guarantees that critical tasks are performed on time. However, soft RTOS is less restrictive. Here, a critical real-time task gets priority over other tasks and retains that priority until it completes.
2. **Single-User, Single-Tasking Operating System:** As the name implies, this operating system is designed to manage the computer so that one user can effectively do one thing at a time. The Palm OS for Palm handheld computers is a good example of a modern single-user, single-task operating system.
3. **Single-User, Multi-Tasking Operating System:** This is the type of operating system most people use on their desktop and laptop computers today. **Windows 98** and the **Mac O.S.** are both examples of an operating system that will let a single user have several programs in operation at the same time. For example, it is entirely possible for you as a Windows user to be writing a note in a word processor while downloading a file from the Internet and at the same time be printing the text of an e-mail message.
4. **Multi-User Operating Systems:** A multi-user operating system allows many different users to take advantage of the computer's resources simultaneously. The operating system must make sure that the requirements of the various users are balanced, and that each of the programs they are using has sufficient and separate resources so that a problem with one user does not affect the entire community of users. **Unix, VMS**, and mainframe operating systems, such as **MVS**, are examples of multi-user operating systems. It's important to differentiate here between multi-user operating systems and single-user operating systems that support networking.

**Windows 2000** and **Novell Netware** can each support hundreds or thousands of networked users, but the operating systems themselves are not true multi-user operating systems. The system administrator is the only user for Windows 2000 or Netware. The network support and the entire remote user logins the network enables are, in the overall plan of the operating system, a program being run by the administrative user.

***Types of OS based on the Nature of Interaction that takes place between the Computer User and His/Her Program during its Processing***

Modern computer operating systems may be classified into three groups, which are distinguished by the nature of interaction that takes place between the computer user and his or her program during its processing. The three groups are: called batch, time-shared and real time operating systems.

1. **Batch Processing OS -** In a batch processing operating system environment, users submit jobs to a central place where these jobs are collected into a batch, and subsequently placed on an input queue at the computer where they will be run. In this case, the user has no interaction with the job during its processing, and the computer’s response time is the turnaround time (i.e. the time from submission of the job until execution is complete, and the results are ready for return to the person who submitted the job).
2. **Time Sharing OS -** Another mode for delivering computing services is provided by time sharing operating systems. In this environment a computer provides computing services to several or many users concurrently on-line. Here, the various users are sharing the central processor, the memory, and other resources of the computer system in a manner facilitated, controlled, and monitored by the operating system. The user, in this environment, has nearly full interaction with the program during its execution, and the computer’s response time may be expected to be no more than a few second.
3. **Real Time OS -** The third class of operating systems, real time operating systems, are designed to service those applications where response time is of the essence in order to prevent error, misrepresentation or even disaster. Examples of real time operating systems are those which handle airlines reservations, machine tool control, and monitoring of a nuclear power station. The systems, in this case, are designed to be interrupted by external signal that require the immediate attention of the computer system.

In fact, many computer operating systems are hybrids, providing for more than one of these types of computing service simultaneously. It is especially common to have a background batch system running in conjunction with one of the other two on the same computer.

***Other Types of OS based on the Definition of the System/Environment***

A number of other definitions are important to gaining a better understanding and subsequently classifying operating systems:

1. **Multiprogramming Operating System -** A multiprogramming operating system is a system that allows more than one active user program (or part of user program) to be stored in main memory simultaneously. Thus, it is evident that a time-sharing system is a multiprogramming system, but note that a multiprogramming system is not necessarily a time-sharing system. A batch or real time operating system could, and indeed usually does, have more than one active user program simultaneously in main storage. Another important, and all too similar, term is ‘multiprocessing’.

A multiprocessing system is a computer hardware configuration that includes more than one independent processing unit. The term multiprocessing is generally used to refer to large computer hardware complexes found in major scientific or commercial applications.

1. **Network Operating Systems -** A networked computing system is a collection of physical interconnected computers. The operating system of each of the interconnected computers must contain, in addition to its own stand-alone functionality, provisions for handling communication and transfer of programs and data among the other computers with which it is connected. In a network operating system, the users are aware of the existence of multiple computers, and can log in to remote machines and copy files from one machine to another. Each machine runs its own local operating system and has its own user (or users).

Network operating systems are designed with more complex functional capabilities. Network operating systems are not fundamentally different from single processor operating systems. They obviously need a network interface controller and some low level software to drive it, as well as programs to achieve remote login and remote files access, but these additions do not change the essential structure of the operating systems.

1. **Distributed Operating Systems -** A distributed computing system consists of a number of computers that are connected and managed so that they automatically share the job processing load among the constituent computers, or separate the job load as appropriate particularly configured processors. Such a system requires an operating system which, in addition to the typical stand-alone functionality, provides coordination of the operations and information flow among the component computers. The distributed computing environment and its operating systems, like networking environment, are designed with more complex functional capabilities. However, a distributed operating system, in contrast to a network operating system, is one that appears to its users as a traditional uniprocessor system, even though it is actually composed of multiple processors. In a true distributed system, users should not be aware of where their programs are being run or where their files are located; that should all be handled automatically and efficiently by the operating system.

True distributed operating systems require more than just adding a little code to a uniprocessor operating system, because distributed and centralized systems differ in critical ways. Distributed systems, for example, often allow program to run on several processors at the same time, thus requiring more complex processor scheduling algorithms in order to optimize the amount of parallelism achieved.

**Functions of Operating Systems**

The major functions of operating systems are;

* **Processor management**: An operating system deals with the assignment of processor to different tasks being performed by the computer system.
* **Memory management**: An operating system deals with the allocation of main memory and other storage areas to the system programs as well as user programs and data.
* **Input/output management**: An operating system deals with the co-ordination and assignment of the different output and input device while one or more programs are being executed.
* **File management:** An operating system deals with the storage of file of various storage devices to another. It also allows all files to be easily changed and modified through the use of text editors or some other files manipulation routines.

Other functions of the operating systems are;

* Establishment and enforcement of a priority system: The operating system determines and maintains the order in which jobs are to be executed in the computer system.
* Automatic transition from job to job as directed by special control statements.
* Interpretation of commands and instructions.
* Coordination and assignment of compilers, assemblers, utility programs, and other software to the various user of the computer system.
* Facilitates easy communication between the computer system and the computer operator (human). It also establishes data security and integrity.

**Design Issues of an Operating System**

There are several problems that needed to be solved using the computer and because the machine is so expensive, it is important to keep it busy all the time to avoid idle time. Some design issues that have been dealt with by OS designers are:

* ***Flexibility*** - Flexible operating systems are taken to be those whose designs have been motivated to some degree by the desire to allow the system to be tailored, either statically or dynamically, to the requirements of specific applications or application domains. The use of object orientation is a common feature of many flexible operating systems. Java-based operating system developed by Sun Microsystems has in the kernel features that will enable a programmer to run JVM.
* ***Reliability*** - In general, **reliability** is the ability of a person or system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances. Reliability is generally considered important by end users. Not all companies making operating systems have a similar standard. Even among operating systems where reliability is a priority, there is a range of quality. Also, an operating system may be extremely reliable at one kind of task and extremely unreliable at another.
* ***Performance*** - The performance of computer hardware typically increases monotonically with time. Even if the same *could* be said of software, the rate at which software performance improves is usually very slow compared to that of hardware. In fact, many might opine that there is plenty of software whose performance has *deteriorated* consistently with time. Moreover, it is rather difficult to establish an objective performance metric for software as complex as an operating system: a *"faster OS"* is a *very* subjective, context dependent phrase. The issue is; how to increase performance of an Operating System?

*Few things Apple did to make Mac OS X Faster*

* Apple computers do not hibernate. Rather, when they "sleep", enough devices (in particular, the dynamic RAM) are kept alive (at the cost of some battery life, if the computer is running on battery power). Consequently, upon wakeup, the user perceives instant-on behavior: a very desirable effect.
* Similarly, by default the system tries to keep network connections alive even if the machine sleeps. For example, if you login from one PowerBook to another, and both of them go to sleep, your login should stay alive within the constraints of the protocols.
* ***Scalability*** is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged. A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a **scalable system.**
* ***Portability***: In order for an operating system's source code to produce a system behaving equally on several platforms, it has to know about the difference between the platforms' architecture, and abstract these differences enough so that a higher software layer can use it, and provide a uniform interface regardless of the underlying architecture. Or, citing the classical definition of an operating system: to manage resources, and provide an interface for using them.
* ***Security***: An OS should be designed in such a way that it encompasses many different techniques and methods which ensure safety from threats and attacks. It should allow different applications and programs to perform required tasks and stop unauthorized interference.

**Operating System Structures**

*Operating system structures includes:*

## System Components

## Operating System Services

## System Calls

## System Programs

## System Structure

## Virtual Machines

## System Design and Implementation

## System Generation

# System Components

# Common System Components

## Process Management

## Main Memory Management

## Secondary-Storage Management

## I/O System Management

## File Management

## Protection System

## Networking

## Command-Interpreter System

# *Process Management -* A process is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task. The operating system is responsible for the following activities in connection with process management.

### Process creation and deletion.

### process suspension and resumption.

### Provision of mechanisms for process synchronization and process communication

# *Main-Memory Management* - Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices. Main memory is a volatile storage device. It loses its contents in the case of system failure. The operating system is responsible for the following activities in connections with memory management:

### Keep track of which parts of memory are currently being used and by whom.

### Decide which processes to load when memory space becomes available.

### Allocate and deallocate memory space as needed.

# *Secondary-Storage Management -* Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory. Most modern computer systems use disks as the principle on-line storage medium, for both programs and data. The operating system is responsible for the following activities in connection with disk management:

### Free space management

### Storage allocation

### Disk scheduling

# *I/O System Management -* The I/O system consists of:

### A buffer-caching system

### A general device-driver interface

### Drivers for specific hardware devices

# *File Management -* A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. The operating system is responsible for the following activities in connections with file management:

### File creation and deletion.

### Directory creation and deletion.

### Support of primitives for manipulating files and directories.

### Mapping files onto secondary storage.

### File backup on stable (nonvolatile) storage media.

# *Protection System -* Protection refers to a mechanism for controlling access by programs, processes, or users to both system and user resources. The protection mechanism must:

### distinguish between authorized and unauthorized usage.

### specify the controls to be imposed.

### provide a means of enforcement.

# *Networking (Distributed Systems) -* A distributed system is a collection processors that do not share memory or a clock. Each processor has its own local memory. The processors in the system are connected through a communication network. A distributed system provides user access to various system resources. Access to a shared resource allows:

### Computation speed-up

### Increased data availability

### Enhanced reliability

# *Command-Interpreter System -* Many commands are given to the operating system by control statements which deal with:

### process creation and management

### I/O handling

### secondary-storage management

### main-memory management

### file-system access

### protection

### networking

# 2. Operating System Services

## Program execution – system capability to load a program into memory and to run it.

## I/O operations – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.

## File-system manipulation – program capability to read, write, create, and delete files.

## Communications – exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via *shared memory* or *message passing*.

## Error detection – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

# Additional Operating System Functions

## Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

### Resource allocation – allocating resources to multiple users or multiple jobs running at the same time.

### Accounting – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.

### Protection – ensuring that all access to system resources is controlled.

# 3. System Calls

## System calls provide the interface between a running program and the operating system. Three general methods are used to pass parameters between a running program and the operating system.

### Pass parameters in *registers*.

### Store the parameters in a table in memory, and the table address is passed as a parameter in a register.

### *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.

# Passing of Parameters as a Table

# 

# 4. System Programs (SP)

## System programs provide a convenient environment for program development and execution. This can be divided into:

### File manipulation

### Status information

### File modification

### Programming language support

### Program loading and execution

### Communications

### Application programs

## Most users’ view of the operation system is defined by system programs, not the actual system calls.

# 5. System Structure – Simple Approach

## MS-DOS – written to provide the most functionality in *the least space*

### not divided into modules

* levels not well separated: programs could access I/O devices directly
* excuse: the hardware of that time was limited (no dual user/kernel mode)

# MS-DOS Layer Structure

# 

## UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts.

### Systems programs

### The kernel

#### Consists of everything below the system-call interface and above the physical hardware

#### Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

# UNIX System Structure

# Layered Approach

## The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface. With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.

# An Operating System Layer

# 

# Layered Structure of the OS

## A layered design was first used in operating system. Its six layers are as follows:



# OS/2 – descendant of MS-DOS that adds multitasking and dual-mode operation with more layered fashion.

# Advantages

# Direct user access to low-level facilities is not allowed.

* Provide more control over hardware and more knowledge of which resources each user program is using.

# OS/2 Layer Structure

# 

# 6. Virtual Machines

## A virtual machine takes the layered approach to its logical conclusion. OS creates the illusion that a process has its own processor with its own (virtual) memory – a virtual copy of the underlying computer. The resources of the physical computer are shared to create the virtual machines.

### CPU scheduling can create the appearance that users have their own processor.

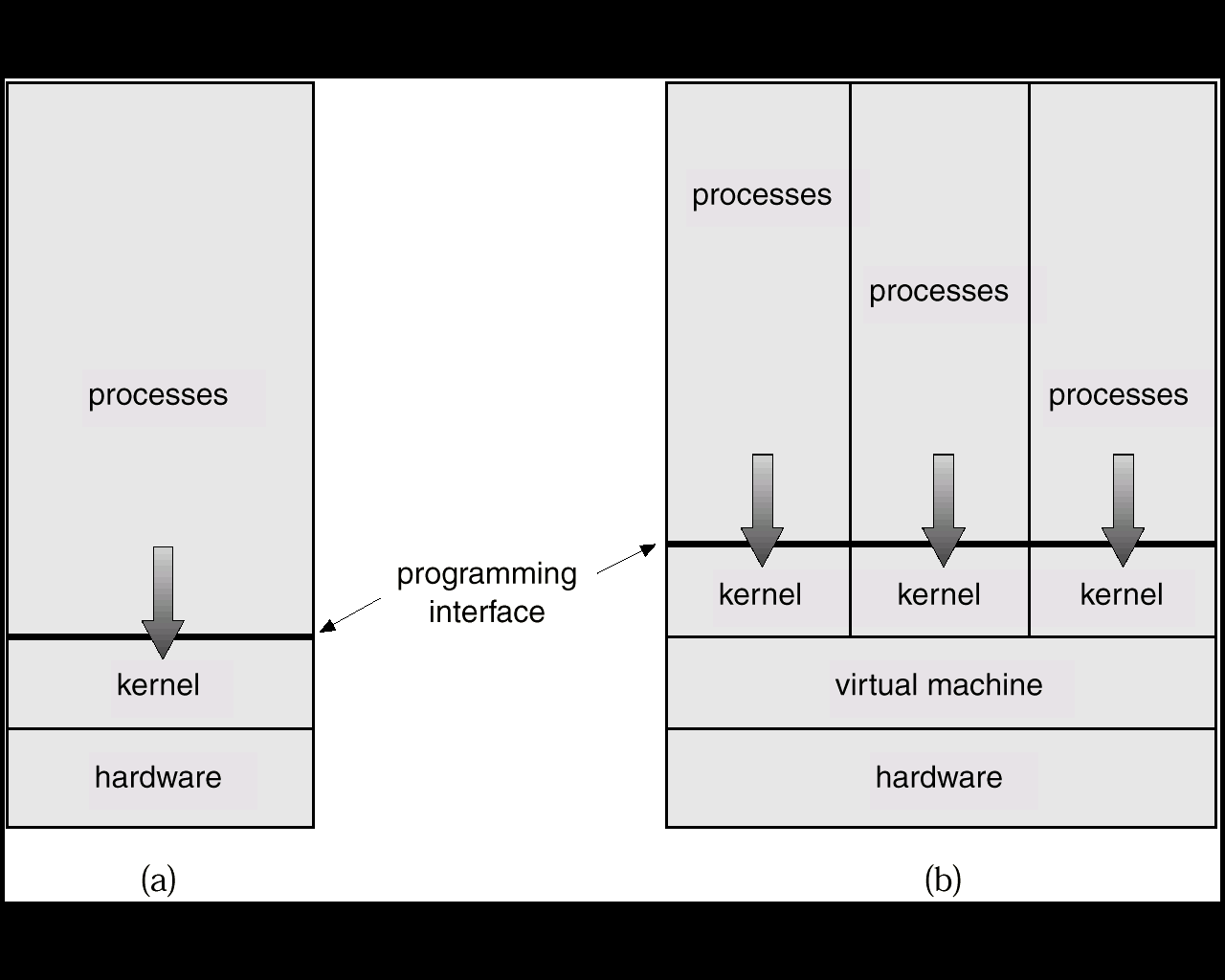
### Spooling and a file system can provide virtual card readers and virtual line printers.

### A normal user time-sharing terminal serves as the virtual machine operator’s console.

# System Models

Non-virtual Machine

Virtual Machine



# Advantages of Virtual Machines

## The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.

## A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.

# Disadvantage of Virtual Machines

## The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine.

**7. System Design and Implementation**

# System Design Goals

## User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast.

## System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

# Mechanisms and Policies

## Mechanisms determine how to do something, policies decide what will be done.

## The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later.

# System Implementation

## Traditionally written in assembly language, operating systems can now be written in higher-level languages.

## Code written in a high-level language:

### can be written faster.

### is more compact.

### is easier to understand and debug.

## An operating system is far easier to *port* (move to some other hardware) if it is written in a high-level language.

# 8. System Generation (SYSGEN)

## Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site.

## SYSGEN program obtains information concerning the specific configuration of the hardware system including

* What CPU is to be used?
* How much memory is available?
* What devices are available?
* What OS options are desired?

**Abstractions, Processes and Resources**

**Abstractions**

Abstraction means that the services provide a more convenient working environment for applications, by hiding some of the details of the hardware, and allowing the applications to operate at a higher level of functions. An operating system transforms the physical world of devices, instructions, memory, and time into virtual world that is the result of abstractions built by the operating system.

There are several reasons for abstraction.

*First*, the code needed to control peripheral devices is not standardized. Operating systems provide subroutines called device drivers that perform operations on behalf of programs for example, input/output operations.

*Second*, the operating system introduces new functions as it abstracts the hardware. For instance, operating system introduces the file abstraction so that programs do not have to deal with disks.

*Third*, the operating system transforms the computer hardware into multiple virtual computers, each belonging to a different program. Each program that is running is called a process. Each process views the hardware through the lens of abstraction.

*Fourth*, the operating system can enforce security through abstraction.

The fundamental operation of the operating system (OS) is to abstract the hardware to the programmer and user. The operating system provides generic interfaces to services provided by the underlying hardware. In a world without operating systems, every programmer would need to know the most intimate details of the underlying hardware to get anything to run. Worse still, their programs would not run on other hardware, even if that hardware has only slight differences.

**Processes**

The term "process" was first used by the designers of the MULTICS in 1960's. Since then, the term process, used somewhat interchangeably with 'task' or 'job'. The process has been given many definitions for instance

* A program in Execution.
* An asynchronous activity.
* The 'animated sprit' of a procedure in execution.
* The entity to which processors are assigned.
* The 'dispatchable' unit.

and many more definitions have given. As we can see from above that there is no universally agreed upon definition, but the definition "*Program in Execution*" seem to be most frequently used.

What is the relation between process and program. It is same beast with different name or when this beast is sleeping (not executing) it is called program and when it is executing becomes process. Well, to be very precise. Process is not the same as program.

A process is more than a program code. A process is an 'active' entity as oppose to program which is consider to be a 'passive' entity. As we all know that a program is an algorithm expressed in some suitable notation, (e.g., programming language). Being a passive, a program is only a part of process. Process, on the other hand, includes:

* Current value of Program Counter (PC)
* Contents of the processors registers
* Value of the variables
* The process stack (SP) which typically contains temporary data such as subroutine parameter, return address, and temporary variables.
* A data section that contains global variables.

A process is the unit of work in a system.

In Process model, all software on the computer is organized into a number of sequential processes. A process includes PC, registers, and variables. Conceptually, each process has its own virtual CPU. In reality, the CPU switches back and forth among processes. (The rapid switching back and forth is called multiprogramming).

**Resource**

A resource is any object which can be allocated within a system. Some examples of resources are processors (CPUs), input/output devices, files, and memory (RAM). Thus, we can restate the purpose of the operating system in terms of resources. The operating system manages resources (resource allocation) and provides an interface to resources for application programs (resource abstraction).

**Application Programming Interface**

An **application programming interface** (**API**) is a set of routines, data structures, object classes and/or protocols provided by libraries and/or operating system services in order to support the building of applications.

An API may be:

* Language-dependent, that is, only available in a particular programming language, utilizing the particular syntax and elements of the programming language to make the API convenient to use in this particular context.
* Language-independent, that is, written in a way that means they can be called from several programming languages. This is a desired feature for a service-style API which is not bound to a particular process or system and is available as a remote procedure call.

Just as drivers provide a way for applications to make use of hardware subsystems without having to know every detail of the hardware's operation, **application program interfaces** (APIs) let application programmers use functions of the [computer](http://computer.howstuffworks.com/pc.htm) and operating system without having to directly keep track of all the details in the [CPU](http://computer.howstuffworks.com/microprocessor.htm)'s operation. Let's look at the example of creating a hard disk file for holding data to see why this can be important.

A programmer writing an application to record data from a scientific instrument might want to allow the scientist to specify the name of the file created. The operating system might provide an API function named **MakeFile** for creating files. When writing the program, the programmer would insert a line that looks like this:

**MakeFile [1, %Name, 2]**

In this example, the instruction tells the operating system to create a file that will allow random access to its data (signified by the 1 -- the other option might be 0 for a serial file), will have a name typed in by the user (%Name) and will be a size that varies depending on how much data is stored in the file (signified by the 2 -- other options might be zero for a fixed size, and 1 for a file that grows as data is added but does not shrink when data is removed). Now, let's look at what the operating system does to turn the instruction into action. The operating system sends a query to the disk drive to get the location of the first available free storage location. With that information, the operating system creates an entry in the file system showing the beginning and ending locations of the file, the name of the file, the file type, whether the file has been archived, which users have permission to look at or modify the file, and the date and time of the file's creation. The operating system writes information at the beginning of the file that identifies the file, sets up the type of access possible and includes other information that ties the file to the application. In all of this information, the queries to the disk drive and addresses of the beginning and ending point of the file are in formats heavily dependent on the manufacturer and model of the disk drive. Because the programmer has written the program to use the API for disk storage, the programmer doesn't have to keep up with the instruction codes, data types and response codes for every possible hard disk and tape drive. The operating system, connected to drivers for the various hardware subsystems, deals with the changing details of the hardware. The programmer must simply write code for the API and trust the operating system to do the rest.

APIs have become one of the most hotly contested areas of the computer industry in recent years. Companies realize that programmers using their API will ultimately translate this into the ability to control and profit from a particular part of the industry. This is one of the reasons that so many companies have been willing to provide applications like readers or viewers to the public at no charge. They know consumers will request that programs take advantage of the free readers, and application companies will be ready to pay royalties to allow their software to provide the functions requested by the consumers.

You often have to rely on others to perform functions that you may not be able or permitted to do by yourself, such as opening a bank safety deposit box. Similarly, virtually all software has to request other software to do some things for it.

All computer operating systems, such as Windows, Unix, and the Mac OS, provide an application program interface for programmers. APIs are also used by video game consoles and other hardware devices that can run software programs. While the API makes the programmer's job easier, it also benefits the [end user](http://www.techterms.com/definition/enduser), since it ensures all programs using the same API will have a similar user interface.

**Types of APIs**

There are many different types of APIs for operating systems, applications or for websites. Windows, for example, has many [API sets](http://msdn.microsoft.com/en-us/library/windows/desktop/hh802935%28v=vs.85%29.aspx) that are used by system hardware and applications — when you copy and paste text from one application to another, it is the API that allows that to work.

Most [operating environments](http://www.webopedia.com/TERM/O/operating_environment.html), such as [MS-Windows](http://www.webopedia.com/TERM/M/MS_Windows.html), provide an API so that programmers can write applications consistent with the operating environment.  Today, APIs are also specified by websites. For example, Amazon or eBay APIs allow developers to use the existing retail infrastructure to create specialized web stores. Third-party software developers also use Web APIs to create software solutions for end-users.

Programmable Web, a site that tracks more than 9,000 APIs, lists Google Maps, Twitter, YouTube, Flickr and Amazon Product Advertising as the most popular APIs.

1. [Google Maps API](https://developers.google.com/maps/): Google Maps APIs lets developers embed Google Maps on webpages using a JavaScript or Flash interface. The Google Maps API is designed to work on mobile devices and desktop browsers.

**Example API Use :** [Charting Your Course Using the Google Maps API.](http://www.developer.com/services/article.php/3814971/Charting-Your-Course-Using-the-Google-Maps-API.htm)

2. [YouTube APIs](https://developers.google.com/youtube/): YouTube API: Google's APIs lets developers integrate YouTube videos and functionality into websites or applications. YouTube APIs include the YouTube Analytics API, YouTube Data API, YouTube Live Streaming API, YouTube Player APIs and others.

3. [Flickr API](http://www.flickr.com/services/api/): The Flickr API is used by developers to access the Flick photo sharing community data. The Flickr API consists of a set of callable methods, and some API endpoints.

4. [Twitter APIs](https://dev.twitter.com/): Twitter offers two APIs. The REST API allows developers to access core Twitter data and the Search API provides methods for developers to interact with Twitter Search and trends data.

5. [Amazon Product Advertising API](https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html): Amazon's Product Advertising API gives developers access to Amazon's product selection and discovery functionality to advertise Amazon products to monetize a website.