# Syllabus

## KCS-401: Operating System

### B.TECH. (COMPUTER SCIENCE AND ENGINEERING)

FOURTH SEMESTER (DETAILED SYLLABUS)

Course Outcome (CO) Bloom's Knowledge Level (KL)

**At the end of course, the student will be able to understand**

| | |
|---|---|
| **CO 1:** Understand the structure and functions of OS | K1, K2 |
| **CO 2:** Learn about Processes, Threads and Scheduling algorithms. | K1, K2 |
| **CO 3:** Understand the principles of concurrency and Deadlocks | K2 |
| **CO 4:** Learn various memory management scheme | K2 |
| **CO 5:** Study I/O management and File systems. | K2,K4 |

**Unit-I**

**Introduction :** Operating system and functions, Classification of Operating systems- Batch, Interactive, Time sharing, Real Time System, Multiprocessor Systems, Multiuser Systems, Multiprocess Systems, Multithreaded Systems, Operating System Structure- Layered structure, System Components, Operating System services, Reentrant Kernels, Monolithic and Microkernel Systems.

**Unit-II**

**Concurrent Processes:** Process Concept, Principle of Concurrency, Producer / Consumer Problem, Mutual Exclusion, Critical Section Problem, Dekker's solution, Peterson's solution, Semaphores, Test and Set operation; Classical Problem in Concurrency- Dining Philosopher Problem, Sleeping Barber Problem; Inter Process Communication models and Schemes, Process generation.

**Unit-III**

**CPU Scheduling:** Scheduling Concepts, Performance Criteria, Process States, Process Transition Diagram, Schedulers, Process Control Block (PCB), Process address space, Process identification information, Threads and their management, Scheduling Algorithms, Multiprocessor Scheduling. Deadlock: System model, Deadlock characterization, Prevention, Avoidance and detection, Recovery from deadlock.

**Unit-IV**

**Memory Management:** Basic bare machine, Resident monitor, Multiprogramming with fixed partitions, Multiprogramming with variable partitions, Protection schemes, Paging, Segmentation, Paged segmentation, Virtual memory concepts, Demand paging, Performance of demand paging, Page replacement algorithms, Thrashing, Cache memory organization, Locality of reference.

**Unit-V**

**I/O Management and Disk Scheduling:** I/O devices, and I/O subsystems, I/O buffering, Disk storage and disk scheduling, RAID. File System: File concept, File organization and access mechanism, File directories, and File sharing, File system implementation issues, File system protection and security.

**Text Book:**

**T1:** Silberschatz, Galvin and Gagne, **"Operating Systems Concepts"**, Wiley

**References:**

**R1:** Andrew S. Tanenbaum **"Modern Operating System "**Pearson Education

**R2:** D M Dhamdhere, "**Operating Systems : A Concept based Approach**", 2nd Edition,TMH

**R3:** William Stallings, "**Operating Systems: Internals and Design Principles**", 6th Edition, Pearson Education

**R4:** Harvey M Dietel, " **An Introduction to Operating System**", Pearson Education

**R5:** Sibsankar Halder and Alex A Aravind, "**Operating Systems**", Pearson Education

**Unit-I**

**Introduction :** Operating system and functions, Classification of Operating systems- Batch, Interactive, Time sharing, Real Time System, Multiprocessor Systems, Multiuser Systems, Multiprocess Systems, Multithreaded Systems, Operating System Structure- Layered structure, System Components, Operating System services, Reentrant Kernels, Monolithic and Microkernel Systems.

## Books :

**T1:** Silberschatz, Galvin and Gagne, **"Operating Systems Concepts"**, Wiley
**R1:** Andrew S. Tanenbaum **"Modern Operating System "**Pearson Education

# OVER VIEW OF OPERATING SYSTEM

**What is an Operating System**?

A program that acts as an intermediary between a user of a computer and the computer hardware

Operating system goals:

- Execute user programs and make solving user problems easier
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner

**Computer System Structure**

Computer system can be divided into four components

- Hardware – provides basic computing resources

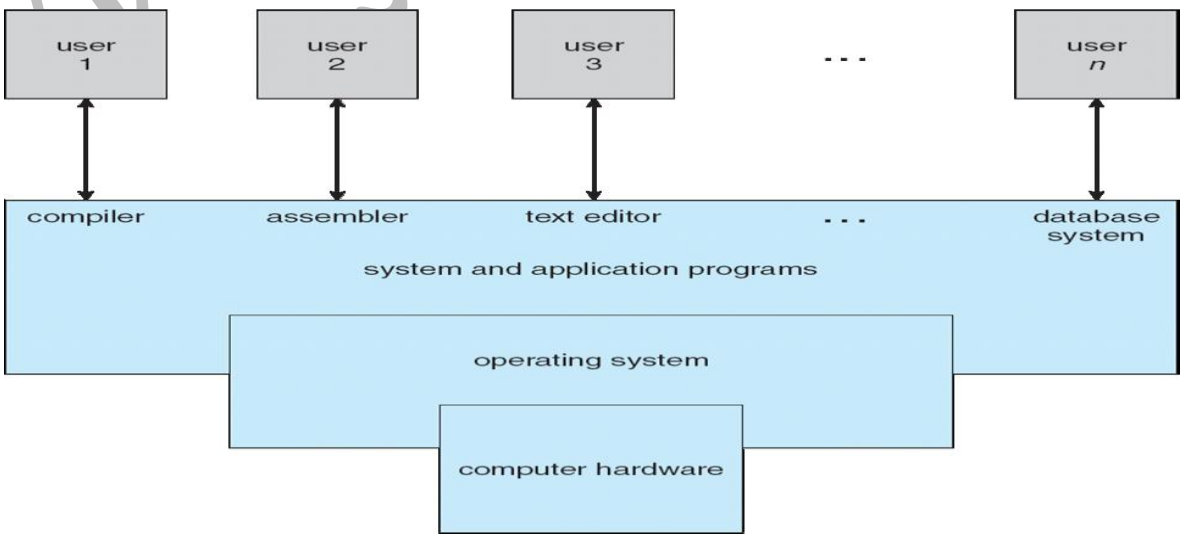
  **Ex:** CPU, memory, I/O devices
- Operating system

  **Ex:** Controls and coordinates use of hardware among various applications and users
- Application programs – define the ways in which the system resources are used to solve the computing problems of the users

  **Ex:** Word processors, compilers, web browsers, database systems, video games
- Users

  **Ex:** People, machines, other computers

**Four Components of a Computer System**



**Operating System Definition:**

- OS is a **resource allocator**
- Manages all resources
- Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
- Controls execution of programs to prevent errors and improper use of the computer
- No universally accepted definition
- Everything a vendor ships when you order an operating system" is good approximation But varies wildly
- "The one program running at all times on the computer" is the **kernel.** Everything else is either a system program (ships with the operating system) or an application program

**Computer Startup**

- **bootstrap program** is loaded at power-up or reboot
- Typically stored in ROM or EPROM, generally known as **firmware**
- Initializes all aspects of system
- Loads operating system kernel and starts execution

# Operating System Components:

We can create a system as large and complex as an operating system only by partitioning it into smaller pieces. Each piece should be a well-delineated portion of the system, with carefully defined inputs, outputs, and functions. Obviously, not all systems have the same structure. However, many modern systems share the goal of supporting the system components outlined in following Sections

### 1. Process Management

A program does nothing unless its instructions are executed by a CPU. A **process** can be thought of as a program in execution, but its definition will broaden as we explore it further.

A time-shared user program such as a compiler is a process.

A word-processing program being run by an individual user on a PC is a process.

A system task, such as sending output to a printer, is also a process.

A process needs certain resources-including CPU time, memory, files, and I/O devices-to accomplish its task. These resources are either given to the process when it is created, or allocated to it while it is running . We emphasize that a program by itself is not a process; a program is a *passive* entity, such as the contents of a file stored on disk, whereas a process is an *active* entity, with a **program counter** specifying the next instruction to execute. The execution of a process must be sequential. The CPU executes one instruction of the process after another, until the process completes. Further, at any time, at most one instruction is executed on behalf of the process. Thus, although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences. It is common to have a program that spawns many processes as it runs.

A process is the unit of work in a system. Such a system consists of a collection of processes, some of which are operating-system processes (those that execute system code) and the rest of which are user processes (those that execute user code). All these processes can potentially execute concurrently, by multiplexing the CPU among them.

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

**2. Main-Memory Management**

The main memory is central to the operation of a modern computer system. Main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices.

The operating system is responsible for the following activities in connection with memory management:

- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes are to be loaded into memory when memory space becomes available
- Allocating and deallocating memory space as needed

**3. File Management**

File management is one of the most visible components of an operating system. Computers can store information on several different types of physical media. Magnetic tape, magnetic disk, and optical disk are the most common media. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has unique characteristics. These properties include access speed, capacity, data-transfer rate, and access method (sequential or random).

For convenient use of the computer system, the operating system provides a uniform logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. The operating system maps files onto physical media, and accesses these files via the storage devices.

A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, or alphanumeric. Files may be free-form (for example, text files), or may be formatted rigidly (for example, fixed fields). A file consists of a sequence of bits, bytes, lines, or records whose meanings are defined by their creators.

The operating system is responsible for the following activities in connection with file management:

- Creating and deleting files
- Creating and deleting directories
- Supporting primitives for manipulating files and directories

- Mapping files onto secondary storage
- Backing up files on stable (nonvolatile) storage media

**4. I/O-System Management**

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the **I/O** subsystem.

The I/O subsystem consists of

- A memory-management component that includes buffering, caching, and spooling
- A general device-driver interface
- Drivers for specific hardware devices

**5. Secondary-Storage Management**

The main purpose of a computer system is to execute programs. These programs, with the data they access, must be in main memory, or **primary storage,** during execution. Because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, the computer system must provide **secondary storage** to back up main memory. Most modern computer systems use disks as the principal 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

**6. Networking**

**A distributed system** is a collection of processors that do not share memory, peripheral devices, or a clock. Instead, each processor has its own local memory and clock, and the processors communicate with one another through various communication lines, such as high-speed buses or networks. The processors in a distributed system vary in size and function. They may include small microprocessors, workstations, minicomputers, and large, general-purpose computer systems.

The processors in the system are connected through a **communication network,** which can be configured in a number of different ways. The network may be fully or partially connected. The communication-network design must consider message routing and connection strategies, and the problems of contention and security.

**7. Protection System**

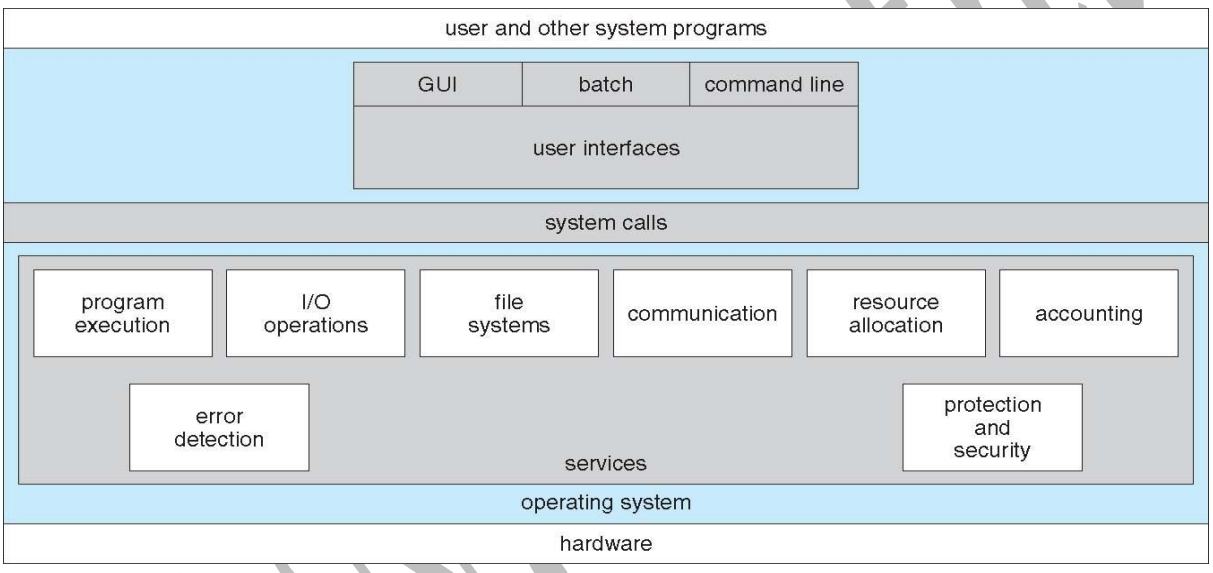
If a computer system has multiple users and allows the concurrent execution of multiple processes, then the various processes must be protected from one another's activities. For that purpose, mechanisms ensure that the files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system. Protection is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide means for specification of the controls to be imposed and means for enforcement.

### 8. Command-Interpreter System

One of the most important systems programs for an operating system is the command interpreter, which is the interface between the user and the operating system. Some operating systems include the command interpreter in the kernel. Other operating systems, such as MS-DOS and UNIX, treat the command interpreter as a special program that is running when a job is initiated, or when a user first logs on (on time-sharing systems).

## Operating-System Services

An operating system provides an environment for the execution of programs .It provides certain services to programs and to the users of those programs. The specific services provided, of course, differ from one operating system to another, but we can identify common classes. These operating-system services are provided for the convenience of the programmer, to make the programming task easier.




### 1. Program execution:

The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

I/O operations: A running program may require I/O. This I/O may involve a file or an I/O device. For specific devices, special functions may be desired (such as to rewind a tape drive, or to blank a CRT screen). For efficiency and protection, users usually cannot control 1/0 devices directly. Therefore, the operating system must provide a means to do I/O.

### 2. File-system manipulation:

The file system is of particular interest. Obviously, programs need to read and write files. Programs also need to create and delete files by name. Communications: In many circumstances, one process needs to exchange information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a computer network. Communications may be implemented via

shared memory, or by the technique of message passing, in which packets of information are moved between processes by the operating system.

**3. Error detection:**

The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

**Systems with multiple users can gain efficiency by sharing the computer resources among the users.**

**4. Resource allocation:**

When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code.

**5. Accounting:**

We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

**6. Protection:**

The owners of information stored in a multiuser computer system may want to control use of that information. When several disjointed processes execute concurrently, it should not be possible for one process to interfere with the others, or with the operating system itself. Protection involves ensuring that all access to system resources is controlled.

## Operating System Structure:

Now that we have seen what operating systems look like on the outside (i.e., the programmer's interface), it is time to take a look inside. In the following sections, we will examine five different structures that have been tried, in order to get some idea of the spectrum of possibilities. These are by no means exhaustive, but they give an idea of some designs that have been tried in practice.

The five designs are monolithic systems, layered systems, virtual machines, exokernels, and client-server systems.

**1. Monolithic Systems**

By far the most common organization, this approach might well be subtitled "The Big Mess." The structure is that there is no structure. The operating system is written as a collection of procedures, each of which can call any of the other ones whenever it needs to. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and

each one is free to call any other one, if the latter provides some useful computation that the former needs.

To construct the actual object program of the operating system when this approach is used, one first compiles all the individual procedures, or files containing the procedures, and then binds them all together into a single object file using the system linker. In terms of information hiding, there is essentially none—every procedure is visible to every other procedure .

Even in monolithic systems, however, it is possible to have *at* least a little structure. The services (system calls) provided by the operating system are requested by putting the parameters in a well-defined place (e.g., on the stack) and then executing a trap instruction. This instruction switches the machine from user mode to kernel mode and transfers control to the operating system. The operating system then fetches the parameters and determines which system call is to be carried out.

This organization suggests a basic structure for the operating system:

1. A main program that invokes the requested service procedure.
2. A set of service procedures that carry out the system calls.
3. A set of utility procedures that help the service procedures.

In this model, for each system call there is one service procedure that takes care of it. The utility procedures do things that are needed by several service procedures, such as fetching data from user programs. This division of the procedures into three layers is shown in Fig.

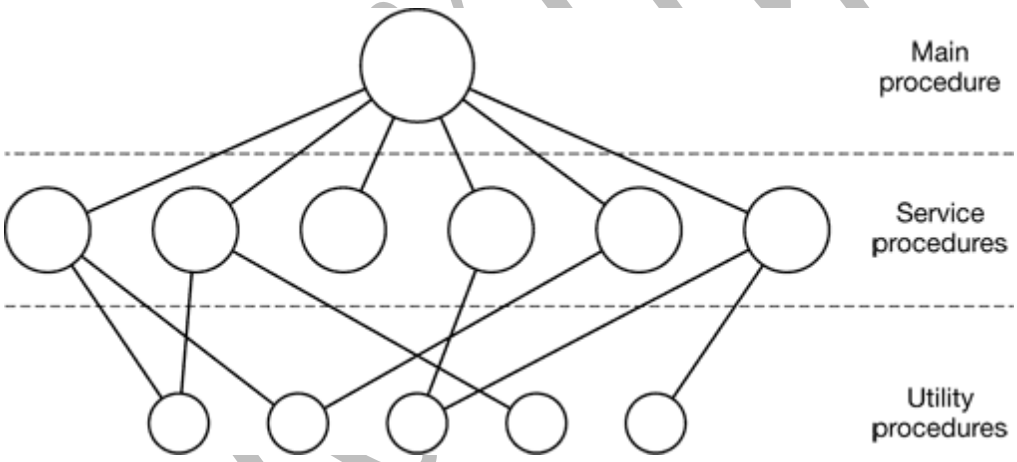



**Figure** A simple structuring model for a monolithic system.

**2. Layered Systems:**

The first system constructed in this way was the THE system built at the Technische Hogeschool Eindhoven in the Netherlands by E. W. Dijkstra (1968) and his students. The THE system was a simple batch system for a Dutch computer, the Electrologica X8, which had 32K of 27-bit words (bits were expensive back then).

**The system had 6 layers:**

**Layer 0:** Layer 0 dealt with allocation of the processor, switching between processes when interrupts occurred or timers expired. Above layer 0, the system consisted of sequential processes, each of which could tie programmed without having to worry about the fact that multiple processes were running on a single processor. In other words, layer 0 provided the basic multiprogramming of the CPU.

| Layer No. | Functions performed by that layer |
|---|---|
| *Layer 5* | *The operator* |
| *Layer 4* | *User programs* |
| *Layer 3* | *Input/output management* |
| *Layer 2* | *Operator-process communication* |
| *Layer 1* | *Memory and drum management* |
| *Layer 0* | *Processor allocation and multiprogramming* |

**Figure.** Structure of the THE operating system.

**Layer 1**: Layer 1 did the memory management. It allocated space for processes in main memory and on a 512K word drum used for holding parts of processes (pages) for which there was no room in main memory. Above layer 1, processes did not have to worry about whether they were in memory or on the drum; the layer 1 software took care of making sure pages were brought into memory whenever they were needed.

**Layer 2:** Layer 2 handled communication between each process and the operator console. Above this layer each process effectively had its own operator console.

**Layer 3:** Layer 3 took care of managing the I/O devices and buffering the information streams to and from them. Above layer 3 each process could deal with abstract I/O devices with nice properties, instead of real devices with many peculiarities.

**Layer 4:** Layer 4 was where the user programs were found. They did not have to worry about process, memory, console, or I/O management.

**Layer 5:** The system operator process was located in layer 5.

**3. Microkernel: Refer to Classroom notes**

**4. Virtual Machines:**This system, originally called CP/CMS and later renamed VM/370 ( Seawright and MacKinnon, 1979), was based on an astute Observation : a timesharing system provides

- Multiprogramming and
- An extended machine with a more convenient interface than the bare hardware.

The essence of VM/370 is to completely separate these two functions.

The heart of the system, known as the **virtual machine monitor,** runs on the bare hardware and does the multiprogramming, providing not one, but several virtual machines to the next layer up,

as shown in Fig. 1-26. However, unlike all other operating systems, these virtual machines are not extended machines, with files and other nice features. Instead, they are *exact* copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine has.

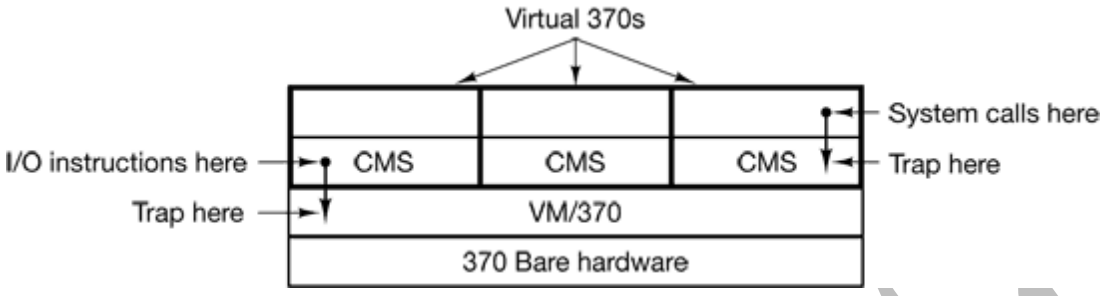



**Figure.**The structure of VM/370 with CMS.

Because each virtual machine is identical to the true hardware, each one can run any operating system that will run directly on the bare hardware. Different virtual machines can, and frequently do, run different operating systems Some run one of the descendants of OS/360 for batch or transaction processing, while other ones run a single-user, interactive system called **CMS (Conversational Monitor System)** for interactive timesharing users. When a CMS program executes a system call, the call is trapped to the operating system in its own virtual machine, not to VM/370, just as it would if it were running on a real

machine instead of a virtual one. CMS then issues the normal hardware I/O instructions for reading its virtual disk or whatever is needed to carry out the call. These I/O instructions are trapped by VM/370, which then performs them as part of its simulation of the real hardware. By completely separating the functions of multiprogramming and providing an extended machine, each of the pieces can be much simpler, more flexible, and easier to maintain.

**Exokernels:**

With VM/370, each user process gets an exact copy of the actual computer. With virtual 8086 mode on the Pentium, each user process gets an exact copy of a different computer. Going one step further, researchers at M.I.T. have built a system that gives each user a clone of the actual computer, but with a subset of the resources .Thus one virtual machine might get disk blocks 0 to 1023, the next one might get blocks 1024 to 2047, and so on.

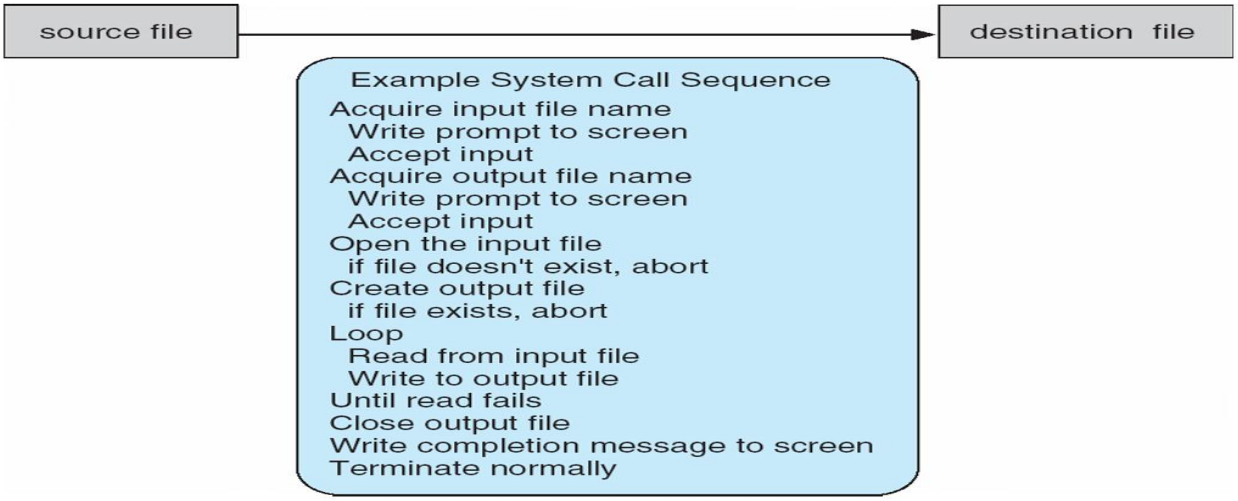
At the bottom layer, running in kernel mode, is a program called the **exokernel.** Its job is to allocate resources to virtual machines and then check attempts to use them to make sure no machine is trying to use somebody else's resources. Each user-level virtual machine can run its own operating system, as on VM/370 and the Pentium virtual 8086s, except that each one is restricted to using only the resources it has asked for and been allocated.

The advantage of the exokernel scheme is that it saves a layer of mapping. In the other designs, each virtual machine thinks it has its own disk, with blocks running from 0 to some maximum, so the virtual machine monitor must maintain tables to remap disk addresses (and all other resources). With the exokernel, this remapping is not needed.

# System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call usenThree most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?(Note that the system-call names used throughout this text are generic)

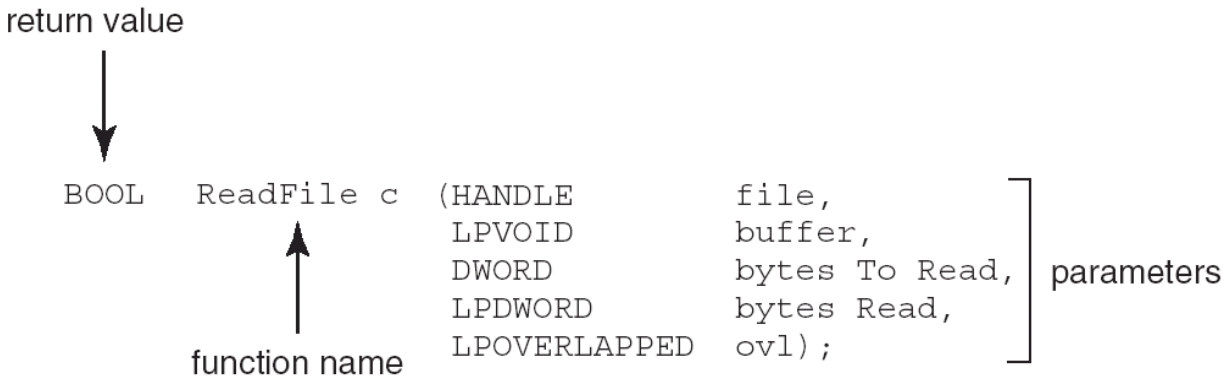
**Example of System Calls**




**Example of Standard API**

Consider the ReadFile() function in the

Win32 API—a function for reading from a file




A description of the parameters passed to ReadFile()

- HANDLE file—the file to be read
- LPVOID buffer—a buffer where the data will be read into and written from

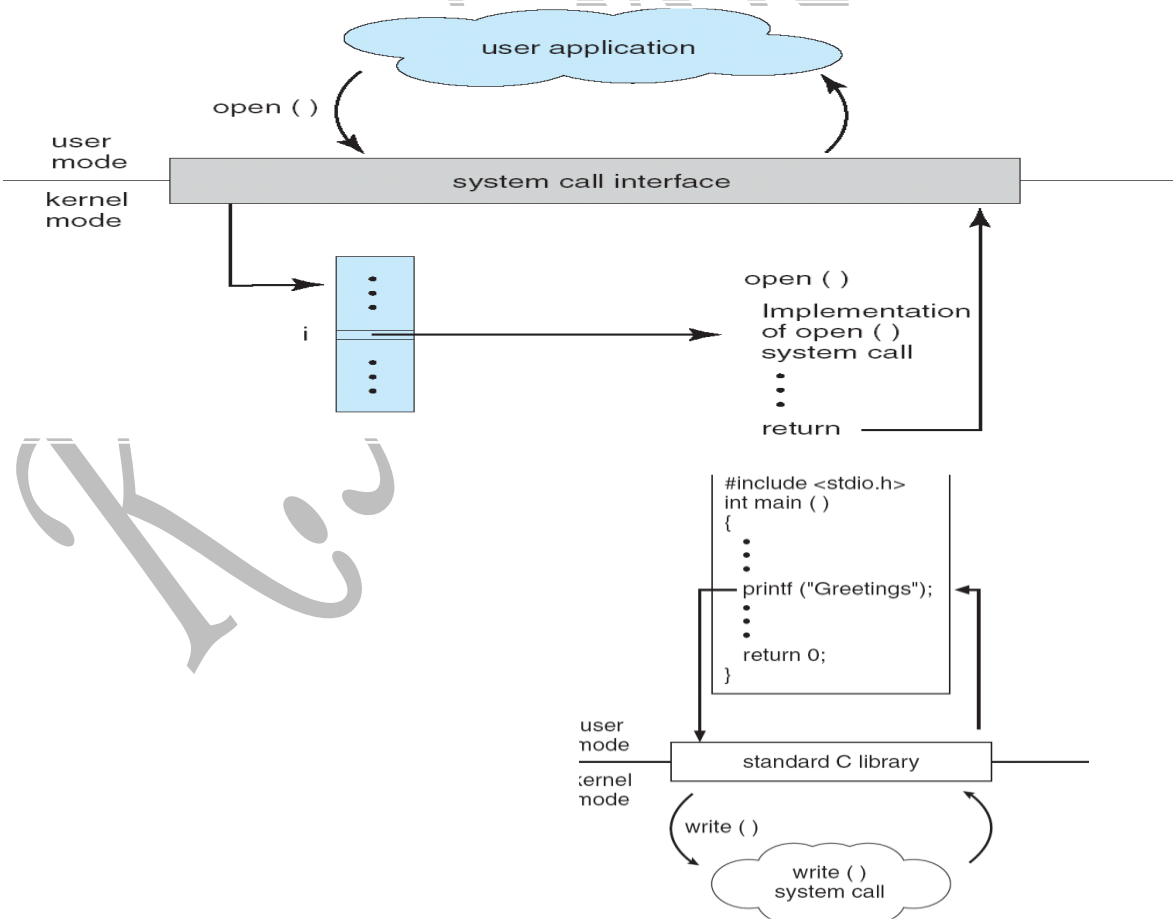
- DWORD bytesToRead—the number of bytes to be read into the buffer
- LPDWORD bytesRead—the number of bytes read during the last read
- LPOVERLAPPED ovl—indicates if overlapped I/O is being used

**System Call Implementation**

- Typically, a number associated with each system call
- System-call interface maintains a table indexed according to these
- Numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
- Just needs to obey API and understand what OS will do as a result call
- Most details of OS interface hidden from programmer by API

Managed by run-time support library (set of functions built into libraries included with compiler)

**API – System Call – OS Relationship**




**Standard C Library Example**

**System Call Parameter Passing**

- Often, more information is required than simply identity of desired system call
- Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
- Simplest: pass the parameters in *registers*
  - In some cases, may be more parameters than registers
- Parameters stored in a *block*, or table, in memory, and address of block passed as a parameter in a register

This approach taken by Linux and Solaris

- Parameters placed, or *pushed*, onto the *stack* by the program and *popped* off the stack by the operating system

Block and stack methods do not limit the number or length of parameters being passed.