UNIT-I
INTRODUCTION

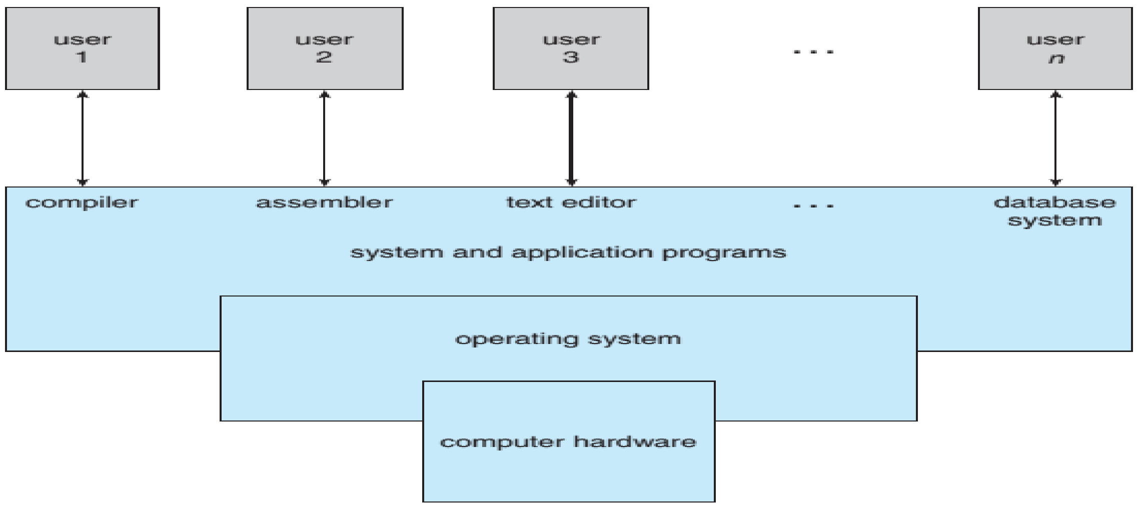
Operating System acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

- Mainframe operating systems are designed primarily to optimize utilization of hardware.
- Personal computer (PC) operating systems support complex games, business applications and everything in between.
- Mobile computer operating systems provides an environment in which a user can easily interface with the computer to execute programs.

Components of Computer system

A computer system can be divided roughly into four components:

1. Hardware
2. Operating system
3. Application programs
4. Users



- Hardware such as Central processing unit (CPU), Memory and the Input/Output (I/O) devices provides the basic computing resources for the system.
- Application programs such as word processors, spreadsheets, compilers and Web browsers define the ways in which these resources are used to solve users' computing problems.
- Operating system controls the hardware and coordinates its use among the various application programs for the various users. Operating system provides an **Environment** within which other programs can do useful work.

DEFINING OPERATING SYSTEMS

Universally there is no accepted definition for operating system:

From the computer's point of view an operating system is viewed as a **Resource Allocator** and a **Control Program**.

1. The operating system acts as the manager of the resources such as CPU time, memory space, file-storage space, I/O devices and so on. Resource allocation is important where many users access the same mainframe or minicomputer.
2. An operating system is a **control program** that manages the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.
3. Operating system is software that is used for controlling and allocating resources. The fundamental goal of computer systems is to execute user programs and to make solving user problems easier. Computer hardware alone is not easy to use, application programs are developed. These programs require certain common operations such as controlling the I/O devices.
4. The operating system is also defined as the program that is running at all times on the computer is called the **Kernel**.

<u>Terms related to operating system</u>

- **System programs**: These are associated with the operating system but are not necessarily part of the kernel.
- **Application programs**: Programs which are not associated with the operation system.
- **Middleware**: It is a set of software frameworks that provide additional services to application developers.
  Ex: Mobile operating system of Apple's iOS and Google's Android features a core kernel along with middleware that supports databases, multimedia and graphics.
- **Firmware**: Read Only memory (ROM), EEPROM are considered as firmware.

GENERATIONS OF OPERATING SYSTEMS

The first true digital computer was designed by the English mathematician Charles Babbage (1792–1871). Ada Lovelace was the world's first programmer. **Ada** programming language was named after her.

First Generation (1945–55)

- In this generation Vacuum Tubes and Plugboards are used.
- These machines were enormous, filling up entire rooms with tens of thousands of vacuum tubes and they are very slower.
- All programming was done in absolute machine language.
- There is no Operating system.
- All the problems were straightforward numerical calculations, such as grinding out tables of sines, cosines, and logarithms.

Second Generation (1955–65)

- In this generation transistors are used in computers.
- These machines called mainframes, they were very costly, only big corporations or major government agencies or universities could afford their multimillion dollar price tags.
- Batch systems were used in this generation.
- Large second-generation computers were used mostly for scientific and engineering calculations, such as solving the partial differential equations occur in physics and engineering.
- They were largely programmed in FORTRAN and assembly language.
- Operating systems FMS (the Fortran Monitor System) and IBSYS are used.

Third Generation (1965–1980)

- In this generation Integrated Circuits are used.
- IBM introduced System/360 machine which run by OS/360.
- System/360 is a software compatible machine.
- The 360 was the first major computer to use (small-scale) Integrated Circuits (ICs).
- Performance was improved by using IC's.
- Multiprogramming model was used in 3rd generation.
- Job spooling technique is introduced.
- Third-generation operating systems were suited for big scientific calculations and massive commercial data processing.
- MULTICS computer, Client server model, introduction of middleware, distributed operating systems, mini computers, UNIX OS were introduced in this generation.

Fourth Generation (1980–Present)

- LSI (Large Scale Integration) circuits were used in this generation.
- Personal Computers are introduced. 8-bit and 16 bit microprocessors were used.
- CP/M(Control Program for Microcomputers), the Apple DOS, Microsoft Disk Operating System (MS-DOS) are used. These are command line systems.
- GUI (Graphical User Interface) based OS also introduced in this generation. These are user friendly computers.
- Mac OS X, Windows OS versions, UNIX OS version such as LINUX was introduced.
- Network operating systems and distributed operating systems are developed.

Types of Operating systems

- Batch operating system
- Multiprogramming systems
- Multi-tasking or Time shared systems

Batch operating systems

- In Batch os a set of similar types of jobs are grouped as batches and they are executed once.
- Input devices: Punch cards, paper tapes, magnetic tapes.
- Output devices: printers, tape drives, punch cards.
- A special program called monitor, manages the execution of each program in the batch.
- Batch OS provides Automatic Job sequencing, that means it automatically loads the next program once the current program execution is completed.
- Example: Jobs with FORTRAN based programming are grouped as one batch.
- The operator selects similar punch cards and groups them.
- The operating system loads FORTRAN compiler and executes all FORTRAN programs at once.
- The job was submitted to the computer operator in form of punch cards.
- At some later time the output appeared.
- It uses First In First Out approach.

Multi-programming systems

- The operating system keeps several jobs in memory simultaneously.
- Operating system selects a job from main memory and starts executes it.
- If the job needs any I/O and it needs wait for I/O to complete then the CPU switches from that job to another job. Then CPU start executing new job without waiting for the old job to finish I/O operation.
- Advantages: It improves the CPU utilization.

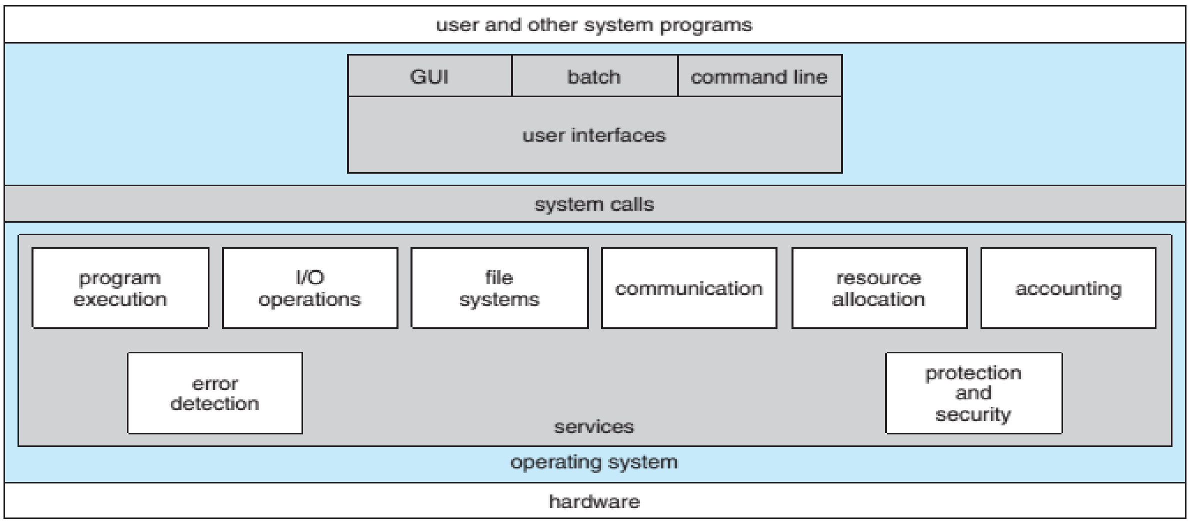
Multi-tasking systems or Time sharing systems

- It is a logical extension of multiprogramming.
- In time sharing systems, several jobs are kept in the main memory.
- Initially OS picks one job and sends it to CPU.
- Let OS select Job1 and send it to CPU for execution.
- Job1 is executed for certain time called Time slice, once the time slice is completed, then OS switches the Job1 to Job2 without considering whether the job1 is finished its eexecution or not.
- Job2 also executed for the time slice period and switches to next job.

OPERATING-SYSTEM SERVICES

An Operating system provides an environment for the execution of programs. OS provides certain services to programs and to the users of those programs.
Operating system services are provided for the convenience of the programmer and to make the programming task easier.
The list of Operating system services that are helpful to the user:

1. User interface
2. Program Execution
3. I/O Operation
4. File system manipulation
5. Communication
6. Error Detection
7. Resource allocation
8. Accounting
9. Protection and Security



User Interface

Almost all operating systems have a **User Interface (UI)**. An UI is generally of 3 types:

- **Command-Line Interface (CLI)** uses text commands and a method for entering them.
- **Batch Interface:** The commands and directives to control those commands are entered into files and those files are executed.
- **Graphical User Interface (GUI)** is a window system with a pointing device (i.e. mouse) to direct I/O, choose from menus and make selections and a keyboard to enter text.

Program Execution

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

I/O operations

A running program may require I/O, which may involve a file or an I/O device. Example are blanking a display screen, recording to a CD or DVD drive or.

File-system manipulation

- Programs need to read and write files and directories.

- They need to create and delete them by name, search for a file and list file information.
- Operating systems include permissions management to allow or deny access to files or directories based on file ownership.

Communications

- One process in its life time needs to communicate with another process to exchange information.
- Communication occurs between processes that are executing on the computer.
- Communications may be implemented via **Shared Memory or Message Passing**.
- In **Shared memory** communication two or more processes read and write to a shared section of memory.
- In **Message passing** communication the packets of information in predefined formats are moved between processes by the operating system.

Error Detection

The operating system needs to be detecting and correcting errors constantly. The different types of errors occurred are:

- CPU and Memory hardware errors such as a memory error or a power failure.
- I/O devices errors such as a parity error on disk, a connection failure on a network, or lack of paper in the printer
- User Program errors such as an arithmetic overflow, an attempt to access an illegal memory location.
- For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

Resource Allocation

- When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
- Operating system resources are CPU cycles, main memory, file storage, I/O devices etc.
- The operating system manages many of these resources.
- CPU cycles, main memory and file storage may have special allocation code. I/O devices may have much more general request and release code.
- In determining how best to use the CPU, operating systems 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.
- There may also be routines to allocate printers, USB storage drives and other peripheral devices.

Accounting

- Operating system keeps track of different kind of resources used by all users.
- This record keeping may be used for accounting so that users can be billed or simply for accumulating usage statistics.
- Usage statistics are valuable tool for researchers who wish to reconfigure the system to improve computing services.

Protection and security

- Protection ensures that all access to system resources is controlled because when several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself.
- Security of the system defends the system from outsider's attacks such as invalid access of system resources such as I/O devices etc. Security starts with user authentication by providing username and password to gain access to system resources.

USER AND OPERATING-SYSTEM INTERFACE

There are two different types of interfaces are available:

1. Command Line Interface (CLI) or Command interpreter.
2. Graphical user interface (GUI)

Command Line Interface (CLI) or Command Interpreter

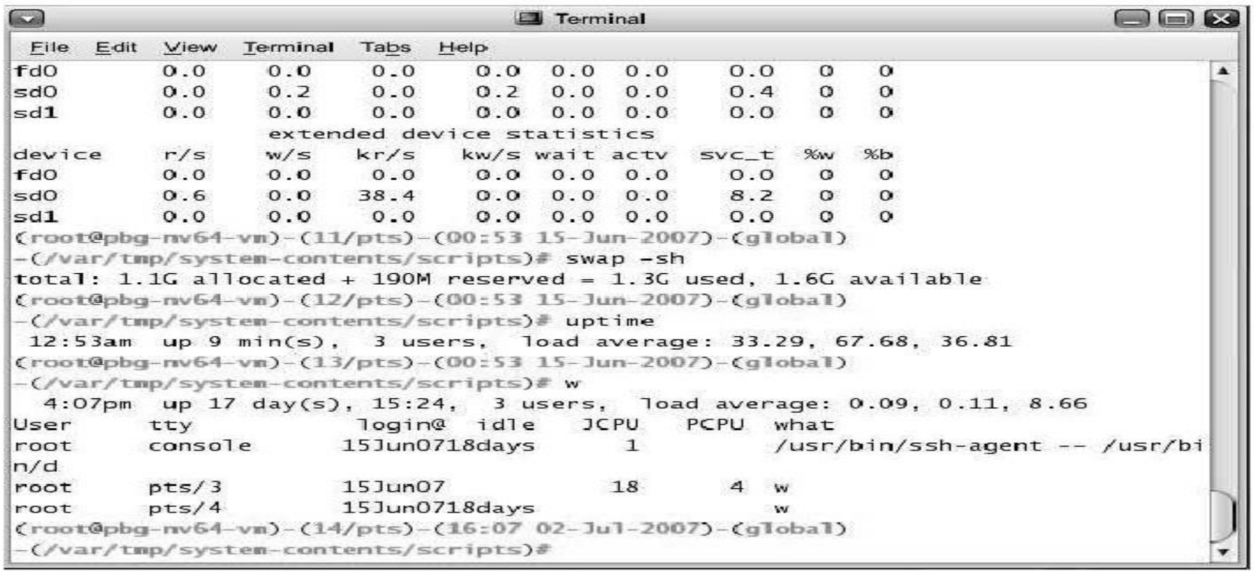
Command Line Interface allows users to directly enter commands to be performed by the operating system.

- Windows and UNIX treat the command interpreter as a special program that is running when a job is initiated. Command interpreters in UNIX or Linux are called as Shells.
- Other operating systems include the command interpreter in the kernel.
- UNIX and Linux systems uses different shells such as **Bourne** shell, **C** shell, **Bourne-Again** shell, **Korn** shell etc.
- The main function of the command interpreter is to get and execute the next user-specified command mostly to manipulate files: create, delete, list, print, copy, execute etc.
- Programmers can add new commands to the system easily by creating new files with the proper names.
- Command-interpreter program does not have to be changed for new commands to be added.

These commands can be implemented in two ways:

1. The command interpreter itself contains the code to execute the command. A command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call.
2. The command interpreter does not understand the command and then it merely uses the command to identify a file to be loaded into memory and executed.
   For example to delete a file in UNIX we use the following command: **rm file.txt**
   The command would search for a file called rm and load the file into memory and then execute it with the parameter **file.txt**.

The function associated with the **rm** command would be defined completely by the code in the file **rm**.



```
fd0        0.0     0.0     0.0     0.0  0.0  0.0     0.0   0    0
sd0        0.0     0.2     0.0     0.2  0.0  0.0     0.4   0    0
sd1        0.0     0.0     0.0     0.0  0.0  0.0     0.0   0    0
                  extended device statistics
device     r/s     w/s    kr/s    kw/s wait actv  svc_t  %w   %b
fd0        0.0     0.0     0.0     0.0  0.0  0.0     0.0   0    0
sd0        0.6     0.0    38.4     0.0  0.0  0.0     8.2   0    0
sd1        0.0     0.0     0.0     0.0  0.0  0.0     0.0   0    0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
 12:53am  up 9 min(s),  3 users,  load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
  4:07pm  up 17 day(s), 15:24,  3 users,  load average: 0.09, 0.11, 8.66
User     tty           login@  idle   JCPU   PCPU  what
root     console      15Jun0718days      1         /usr/bin/ssh-agent -- /usr/bi
n/d
root     pts/3        15Jun07           18      4  w
root     pts/4        15Jun0718days                w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```

Figure 2.2 The Bourne shell command interpreter in Solrais 10.

Graphical User Interfaces

Graphical user interface (GUI) is a user friendly interfacing with the operating system.

- In GUI rather than entering commands directly via a command-line interface, users employ a mouse-based window and menu system characterized by a **desktop** metaphor.
- The user moves the mouse to position its pointer on images or **icons**, on the screen that represent programs, files, directories and system functions.

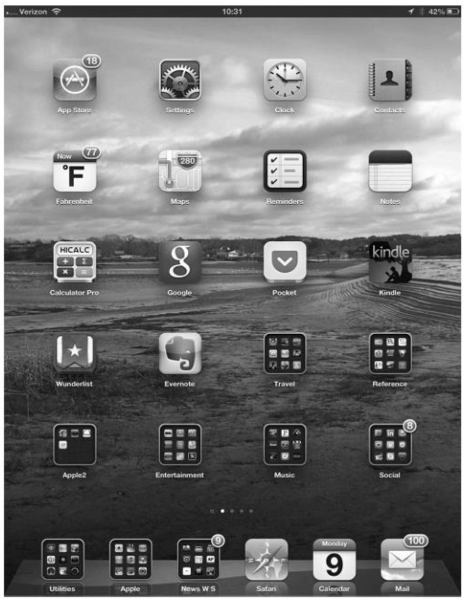


Figure 2.3 The iPad touchscreen.

- Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory known as a **folder** or pull down a menu that contains commands.
- The first Graphical user interfaces appeared on the Xerox Alto computer in 1973.
- Graphical interfaces became more widespread with the advent of Apple Macintosh computers in the 1980s.

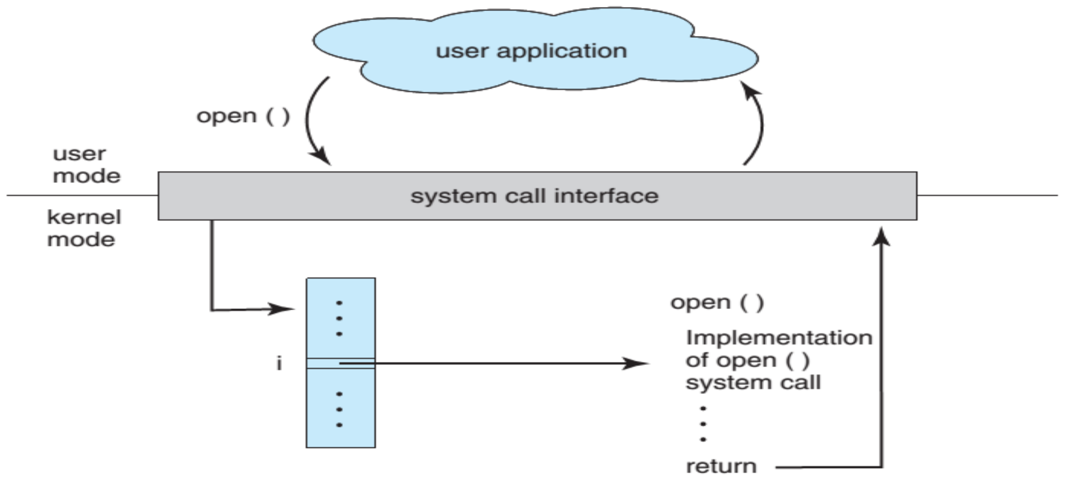
- Microsoft’s first version of Windows Version 1.0 was based on the addition of a GUI interface to the MS-DOS operating system.
- Smartphones and handheld tablet computers uses a touchscreen interface. Users interact with touchscreen by pressing and swiping fingers across the screen.
- **K Desktop Environment** (**KDE**) and the **GNOME** desktop by the GNU project are GUI designs which run on Linux OS.

1.5 System calls

System calls provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf. When a system call is executed, it is typically treated by the hardware as software interrupt. The kernel examines the interrupting instruction to determine what system call has occurred.

System calls provide an interface to the services made available by an operating system.

- System calls are generally available as routines written in C and C++ and also written using assembly-language instructions.
- Each operating system has its own name for each system call.
- Application developers design programs according to an **Application Programming Interface (API)**.
- The API specifies a set of functions that are available to an application programmer including the parameters that are passed to each function and the return values the programmer can expect.
- The functions that make up an API typically invoke the actual system calls on behalf of the application programmer.



Example: The Windows function CreateProcess( ) which is used to create a new process actually invokes the NTCreateProcess( ) system call in the Windows kernel.

For most programming languages, the run-time support system (a set of functions built into libraries included with a compiler) provides a **System Call Interface** that serves as the link to system calls made available by the operating system.

- The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.
- A number is associated with each system call and the system-call interface maintains a table indexed according to these numbers.
- The system call interface then invokes the intended system call in the operating-system kernel and returns the status of the system call and any return values.

The below figure shows the relationship between an API, the system-call interface and the operating system for open( ) system call.

Types of system calls:

1. **Process management system calls**

   pid=fork() : create a child process identical to the parent

   pid=waitpid(pid,&statloc,opts): wait for a child to terminate

   s=execve(name,argc,envp): replace a prcess core image

   exit(status): terminate process execution and return status

   pid=getpid(): return the callers process id

2. **File management system calls**

   Fd=create(name,mode): obsolete way to create a new file.

   Fd=mknod(name,mode,addr): create a regular, special, or directory i-node.

   Fd=open(file, how): poen a file for reading , writing or both

   N=read(fd,buffer, nbytes): read data from a file into a buffer

   N=write(fd,buffer, nbytes): write data from a buffer into a file.

   S=close(fd): close an open file.

3. **Device and File system management system calls**

   S=pipe(&fd[0]): create a pipe

   S=ioctl(fd,req,argp): perform special operations on a file

   S=fcntl(fd,cmd): file locking and other operations.

   S=rename(old,new): give a file to new name

   S=access(name,amode): check a file’s accessibility.

1.6 OPERATING-SYSTEM STRUCTURE

1. Simple Structure
2. Layered Approach
3. Microkernels

Simple Structure

MS-DOS operating systems do not have well-defined structures.

- It was designed and implemented by few people and started as small, simple and limited systems and then grew beyond their original scope that the implementers didn't imagine it would be so popular.
- In MS-DOS, the interfaces and levels of functionality are not well separated.
- In MS-DOS, application programs are able to access the basic I/O routines to write directly to the display and disk drives.
- Such freedom leaves MS-DOS vulnerable to errant or malicious programs, causing entire system crashes when user programs fail.

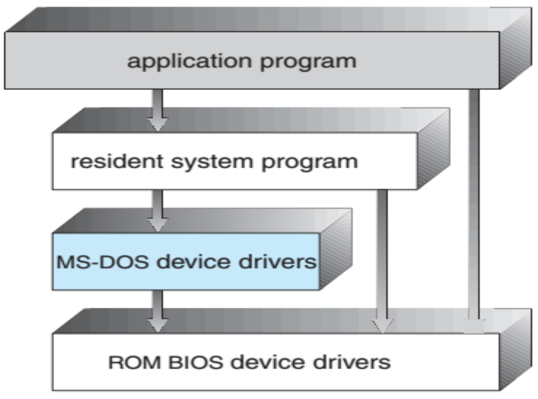


Figure 2.11 MS-DOS layer structure.

Original UNIX operating system is also have limited structuring, it was limited by its hardware functionality.

- It consists of two separable parts: **Kernel** and **System programs**.
- The kernel is is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved.

The traditional UNIX operating system is layered structured to some extent.

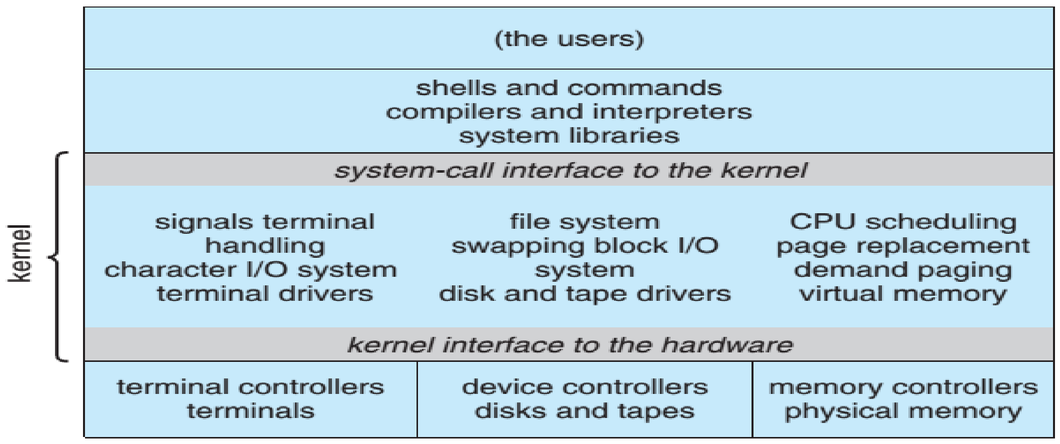
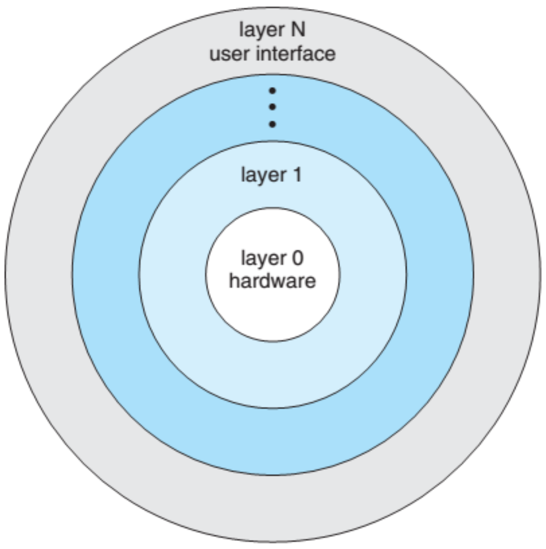


Figure 2.12 Traditional UNIX system structure.

- Everything below the system-call interface and above the physical hardware is the kernel.
- The kernel provides the file system, CPU scheduling, memory management and other operating-system functions through system calls and all these functionalities to be combined into one level.
- This monolithic structure was difficult to implement and maintain.

Layered Approach

- In Layered approach the operating system is broken into a number of layers (levels).
- The **bottom layer 0** is the Hardware and the **highest layer *N*** is User interface.
- A operating-system layer consists of data structures and a set of routines that can be invoked by higher-level layers and can also invoke operations on lower-level layers.
- An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data.



Advantages of Layered Approach

The main advantage of the layered approach is simplicity of construction and debugging.

- Each layer uses functions (operations) and services of only lower-level layers.
- This approach simplifies debugging and system verification.
- The first layer can be debugged without any concern for the rest of the system because it uses only the basic hardware to implement its functions.
- Once the first layer is debugged and functions correctly while the second layer is debugged and so on.
- If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged.

Other advantage of Layered structure is Data Hiding:

- Each layer is implemented only with operations provided by lower-level layers.
- A layer does not need to know how these operations are implemented instead a layer just knows what these operations do.
- Hence, each layer hides the existence of certain data structures, operations and hardware from higher-level layers.

Problems with Layered Approach

The major difficulty with the layered approach involves appropriately defining the various layers.

- A layer can use only lower-level layers hence we have plan carefully which layer to be kept in which place.
- **Example:** The device driver for the backing store (i.e. disk space used by virtual-memory algorithms) must be at a lower level than the memory-management routines, because memory management requires the ability to use the backing store.
- The backing-store driver would normally be above the CPU scheduler, because the driver may need to wait for I/O and the CPU can be rescheduled during this time.

Other problem with layered implementations is that they tend to be less efficient.

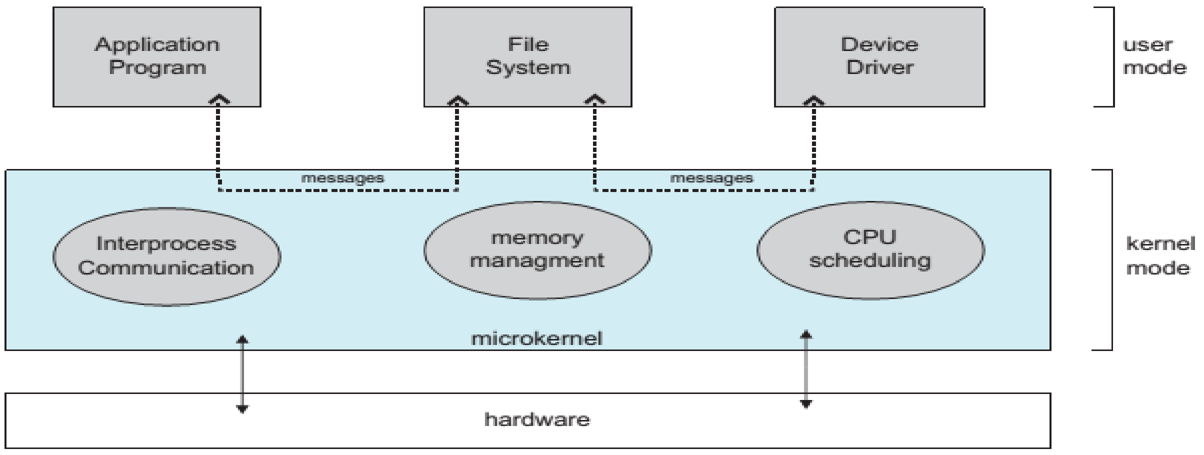
- When a user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware.
- At each layer, the parameters may be modified, data may need to be passed and so on. Each layer adds overhead to the system call.
- The net result is a system call that takes longer time than a non-layered system.

1.7 Microkernels

As the UNIX OS is expanded, the kernel also became large and difficult to manage.

- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **Microkernel** approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs resulting a smaller kernel (i.e.) Microkernel.
- There is little consensus regarding which services should remain in the kernel and which service should be implemented in user space.
- Microkernels provide minimal process and memory management and also communication facility.
- The main function of the microkernel is to provide **Message Passing Communication** between the client program and the various services that are also running in user space.

The below figure shows the architecture of a Microkernel



Example: if the client program wishes to access a file, it must interact with the file server. The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.

Advantages of Microkernel

Microkernel approach is makes extending the operating system easier.

- All new services are added to user space and consequently do not require modification of the kernel.
- Even-though when the kernel must have to be modified, the changes in the kernel will be very few, because the microkernel is a smaller kernel.
- The resulting operating system is easier to port from one hardware design to another.
- The microkernel also provides more security and reliability, since most services are running as user processes rather than kernel processes.
- If a service fails, the rest of the operating system remains untouched.

Disadvantage of Microkernel

- The performance of microkernels can suffer due to increased system-function overhead.

1.8 Virtualization

Virtualization is a technology that allows operating systems to run as applications within other operating systems.

- Virtualization allows an entire operating system written for one platform to run on another platform.
- With **virtualization** an operating system that is natively compiled for a particular CPU architecture runs within another operating system also native to that CPU.
- Virtualization first came about on IBM mainframes as a method for multiple users to run tasks concurrently.
- Running multiple virtual machines allows many users to run tasks on a single user system.

Example: An Apple laptop running Mac OS X on the x86 CPU can run a Windows guest to allow execution of Windows applications.

- VMware is a virtualization software application. If VMware is installed on Windows OS, then Windows is the **host** operating system and the VMware application was the virtual machine manager (VMM).

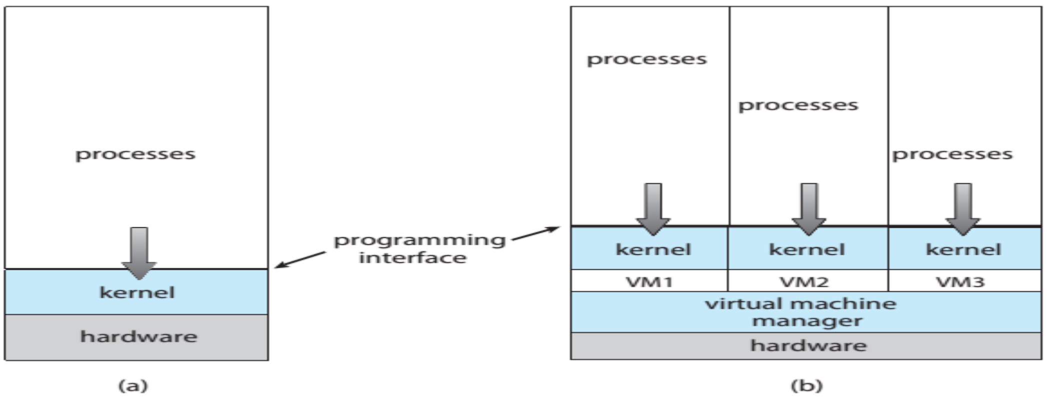
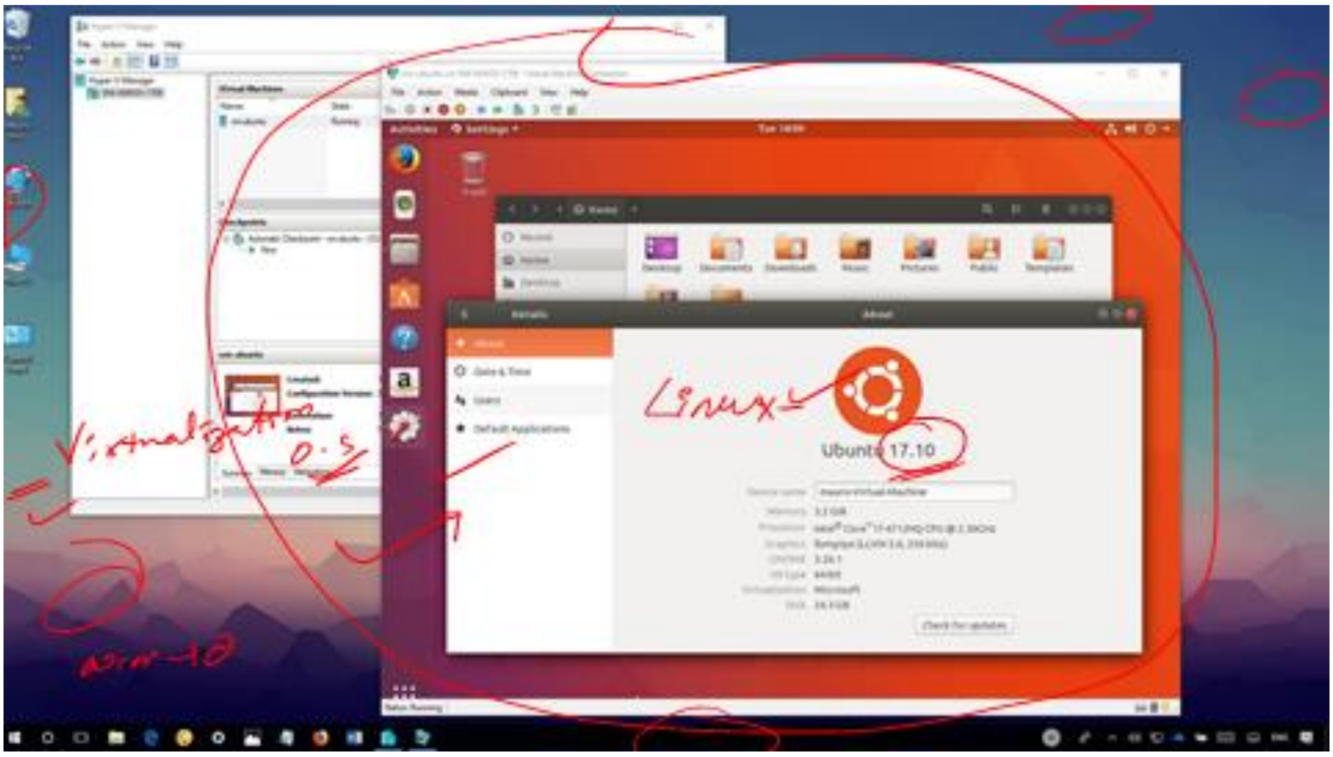


Figure 1.20 VMware.

- If VMware application runs one or more guest copies of other operating systems such as Linux, then LINUX will be the guest operating system.
- The VMM runs the guest operating systems, manages their resource use and protects each guest from the others.

Softwares: *Virtual Box, Microsoft Hyper V, Citrix Hypervisor, QEMU, Parallel Desktop, Xen project etc.*


Linux
Ubuntu 17.10
Virtualization
O.S
win-10
Check for updates