Introduction:- An O.S is a program that manages the computer h/w. It acts as an intermediary between the computer user and the computer h/w. There are many variations of the O.S. like,

(1) Mainframe O.S :- Designed primarily to optimize utilization of h/w.

(2) Personal computer O.S :- Support complex games, business applications and everything in betn.

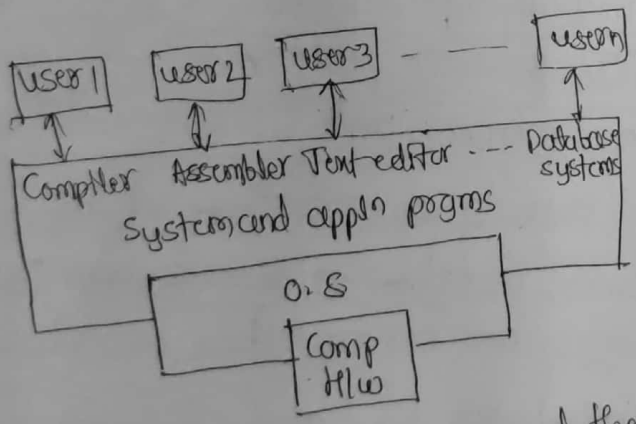
(3) O.S. for handheld computers :- Designed to provide an environment in which a user can easily interface with the comp to execute programs.

∴ Some O.S are designed to be convinient, others to be efficient and some others are combinations of the two.

∵ an O.S is large and complex, it must be created piece by piece.

Abstract view of components of a comp-system :- A comp-system can be divided roughly into 4 components.

(1) H/w (2) O.S
(3) Appln programs
(4) users.



(1) The comp. h/w (CPU, Memory and the I/O devices) provides the basic computing resources for the system.

(2) 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. O.S. controls the h/w and coordinates ~~the~~ its use among the various appln programs for various users.

To understand more fully the O.S's role, it can be viewed from,

(1) User's view
(2) System view.

① Single user ② Multi-user
✓ Ease of use — Mainframe or Mincomp
✓ Performance
✗ Resource utilization — Resource utilization

User View:- User's view of computer varies according to the interface being used. A single user can utilize the resources. The goal is to maximize the work that the user is performing. In this case, O.S. is designed mostly for ease of use, with some attention paid to performance and none paid to resource utilization - how various h/w and s/w resources are shared. Performance is of course, important to the user but such systems are optimized for the single-user experience rather than the requirements of multiple users.

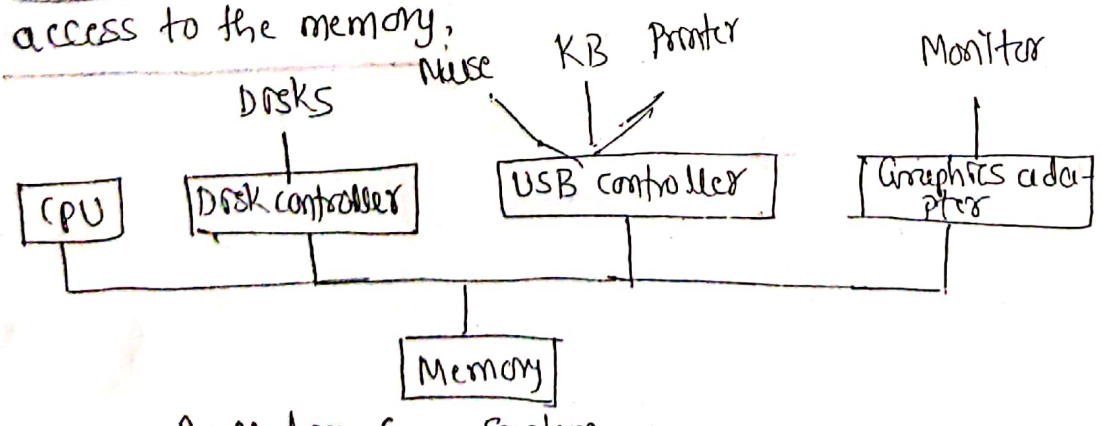
In other cases, a user sits at a terminal connected to a mainframe or a minicomputer. Other users are accessing the same computer through other terminals. These users share resources and may exchange information. O.S. in such cases is designed to maximize resource utilization - to assume that all available CPU time, memory and I/O are used efficiently and that no individual user takes more than her fair share.

In still other cases, users sit at workstations connected to n/ws of other workstations and servers. These users have dedicated resources at their disposal, but they also share resources such as networking and servers - file, compute and print servers. ∴ Their O.S. is designed to compromise betn individual usability and resource utilization.

System view :- From the computer's point of view, O.S is the program most intimately involved with the h/w. In this context we can view an O.S. as a resource allocator. A comp system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices and so on. O.S. acts as the manager of these resources. For confirming requests of resources, O.S must decide how to allocate them to specific programs and users so that it can operate the comp system efficiently and fairly. ① Resource allocator ② Controller } O.S acts as

A slightly diff. view of an O.S. emphasizes the need to control the various I/O devices and user programs. An O.S is a control prg. A control prg manages the execution of user prgms to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.

Computer-system operation :- A modern general-purpose comp. system consists of one or more CPUs and a no of device controllers connected thr' a common bus that provides access to shared memory. Each device controller is in charge of a specific type of device. The CPU and the dev. controllers can execute concurrently, competing for memory cycles. To ensure orderly access to the shared memory a mem. controller is provided whose function is to synchronize the access to the memory.



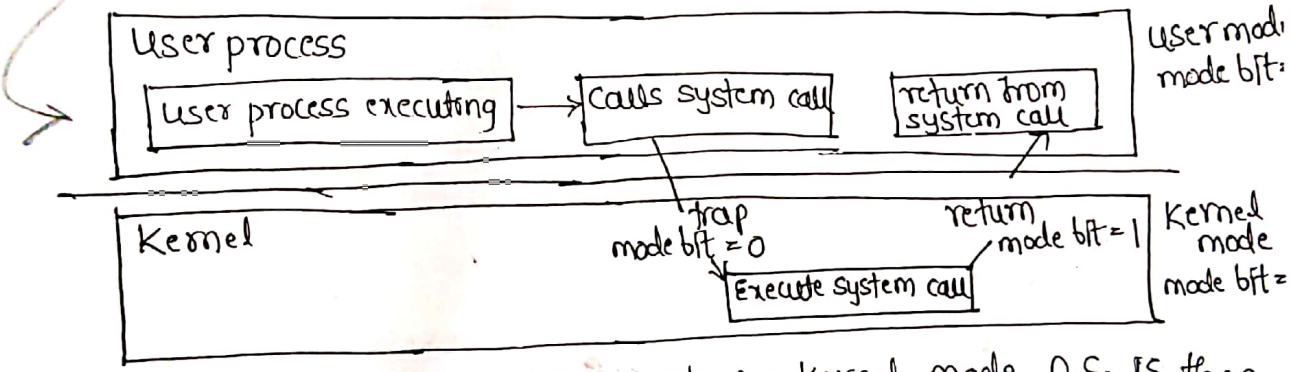
A Modern Comp. System

Imp Operating system operations :- Modern operating systems are interrupt driven. If there are no processes to execute, no devices to service and no users to whom to respond, an OS will sit quietly waiting for something to happen. Events are almost always signaled by the occurance of an interrupt or a trap. A trap (or an exception) is a software-generated interrupt caused by either by an error (eg: division by zero or invalid memory access) or by a specific request from a user program, that an o.s. service be performed. The interrupt-driven nature of an o.s. define that system's general structure. For each type of interrupt, separate segments of code in the o.s. determine what action should be taken. An interrupt-service-routine (ISR) is provided that is responsible for dealing with the interrupt.

Since the o.s. and the users share the hardware and software resources of the comp. system, we need to make sure that an error in a user program could cause problems only for the one program running. With sharing many processes could be adversely affected by a bug in one program. eg: If a process gets struck in an infinite loop, this loop could prevent the correct operation of many other processes. A properly designed o.s must ensure that an incorrect program cannot cause other programs to execute incorrectly.

Dual-mode operation :- In order to ensure the proper execution of the o.s, we must be able to distinguish between the execution of o.s. code and user-defined code. There are two separate modes of operation ie user mode and kernel-mode also called supervisor mode and system mode or privileged mode.

A bit called the mode bit, is added to the hardware of the computer to indicate the current mode: Kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed on behalf of the O.S. and one that is executed on behalf of the user. When the computer system is executing on behalf of a user application, the system is in user mode. However when a user application requests a service from the O.S. (via a system call), it must transition from user to kernel mode to fulfill the request. Transition from user to kernel mode is as shown below.



At system boot time, h/w starts in kernel mode. O.S. is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, h/w switches from user mode to kernel mode. Thus whenever the O.S. gains control of the computer, it is in kernel mode. The system always switches to user mode before passing control to a user program.

The dual mode of operation provides us with the means for protecting the O.S. from errant users and errant users from one another. We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions. The h/w allows privileged instructions to be executed only in kernel mode. If an attempt is made to

execute a privileged instruction in user mode, the h/w does not execute the instruction but rather treats it as illegal and traps it to the o.s. Instruction to switch to kernel mode is an example of a privileged instruction. Some other examples include I/O control, timer management and interrupt management.

Life cycle of instruction execution is as follows. Initial control resides in the o.s, where instructions are executed in kernel mode. When control is given to a user application the mode is set to user mode. Eventually control is switched back to the o.s. via an interrupt, a trap or a system call.

System calls provide the means for a user program to ask the o.s. to perform tasks reserved for the o.s. on the user program's behalf. A system call is invoked in a variety of ways, depending on the functionality provided by the underlying processor. In all forms, it is a method used by a process to request action by the o.s.

System call is treated by h/w as a s/w interrupt.

control passes through interrupt vector to a service routine in the o.s (mode bit = 0 kernel mode)

When a system call is executed, it is treated by the h/w as a s/w interrupt. Control passes through the interrupt vector to a service routine in the o.s. and the mode bit is set to kernel mode. The system call service routine is a part of o.s. Kernel examines the interrupting instruction to determine what system call has occured – a parameter indicates what type of service the user program is requesting. Additional information needed for the request may be passed in registers, on the stack or in memory. Kernel verifies the parameters are correct and legal, executes the request and returns control to the instruction following the system call.

Why dual-mode? Lack of support for the dual-mode can cause serious shortcomings in an o.s.
eg: MS-DOS was written for the Intel 8088 architecture, which has no mode bit and hence no dual mode. Hence a user program can wipe out the o.s. by writing over it with data, and multiple programs are able to write to a device at the same time with potentially disrupted results. Now the recent versions of the Intel CPU are supporting dual-mode operation. Other examples are Microsoft Vista, Windows XP, Unix, Linux & solaris etc

Dual mode also protects the os and also helps to detect errors. These errors are normally handled by the o.s. If a user program fails in some way – such as by making an attempt either to execute an illegal instruction or to access memory that is not in the user's address space, then the h/w traps to the o.s. The trap transfers control through the interrupt vector to the o.s., just as an interrupt does. When a prg error occurs, o.s. must terminate the program abnormally and an appropriate message is given and the memory of the program may be dumped. ~~and~~ This memory contents are dumped to a file so that the programmer can examine it and correct it and restart the program.

Timer:- One should ensure that the o.s. maintains control over the CPU. We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the o.s. To achieve this, we need a timer. A timer can be set to interrupt the computer after a specified period. Period may be fixed or variable. A variable timer is generally implemented by a fixed rate clock and a counter. O.S sets the counter. Every time the clock ticks, counter is decremented. When the counter reaches zero, an interrupt occurs. Thus we can use the ~~counter~~ timer to prevent a user program from running too long.

Processor Management :- A program in execution is called a process.

eg: A time-shared user program such as a compiler.
A word-processing program being run by an user on a PC.
A system-task, such as sending O/p to a printer.

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. When the process terminates, O.S. will reclaim any reusable resources.

A program by itself is not a process, a program is a passive entity like the contents of a file stored on disk, whereas a process is an active entity. A single-threaded process has one program counter specifying the next instruction to execute. The execution of such a process must be sequential. CPU executes one instruction of the process after another, until the process completes.

A process is the unit of work in a system. Such a system consists of a collection of processes, some of which are O.S. processes and the rest of which are user processes those that execute user code. All these processes can potentially execute concurrently - by multiplexing on a single CPU, for example.

O.S. is responsible for the following activities in connection with process management.

(1) Scheduling processes and threads on the CPUs.
(2) Creating and deleting both user and system processes.
(3) Suspending and resuming processes.
(4) Providing mechanisms for process synchronization.
(5) Providing mechanisms for process communication.

memory free so that the next program can be loaded

Memory Management :- Main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or bytes has its own address. Processor reads instructions from main memory during the instruction fetch-cycle and performs both reads and writes data from main memory during the data-fetch cycle (in a von Neumann Arch). After execution the program terminates and the memory space will be freed so that the next program can be loaded and executed.

To improve the utilization of the CPU and the speed of the computer's response to its user, general purpose computers must keep several programs in memory, creating a need for memory management. There are many memory management schemes and there is a need to consider many factors especially the hardware design of the system before using a specific scheme. O.S. is responsible for the following activities in connection with memory management.

(1) Keeping track of which parts of memory currently being used and by whom.

(2) Deciding which processes and data to move into and out of memory.

(3) Allocating and deallocating memory space as needed.

Storage Management :- O.S. maps files onto physical media and accesses these files via the storage devices.

File-system Management :- Computers can store information on several different types of physical media, like magnetic disk, optical disk and magnetic tape etc. Each one is controlled by a device, such as disk drive, tape drive etc. Each device has variable access speed, capacity, data-transfer rate and access methods.

A file is a collection of related information defined by creat

O.S is responsible for the following activities in connec

File management.

(1) Creating and deleting files
(2) " " " directories to organize files.
(3) Supporting primitives for manipulating files and directorie
(4) Mapping files onto secondary storage.
(5) Backing up files on stable (non-volatile) storage media

Mass-storage management :- Main memory is too small to a
store all data and programs and also its contents are los
power is lost. Hence secondary storage to back up main m
is required. Most programs including compilers, assemblers,
processors, editors and formatters are stored on a disk until
loaded into memory. Hence proper management of disk stor
is of much importance. OS is responsible for the following
activities in connection with disk management.

(1) Free space management
(2) Storage allocation
(3) Disk scheduling.

Caching :- Caching is an important principle of computer systems.
Information is normally kept in some storage system. As it is used
it is copied into a faster storage system — the cache — on a temporary
basis. In addition internal programmable registers, such as index
registers, provide a high-speed cache for main memory. There are also
caches that are implemented totally in h/w. eg: most systems have
instruction cache to hold the instructions expected to be executed next.
Without this cache, CPU would have to wait several cycles. Hence
most systems have one or more high-speed data caches in the memory
hierarchy. Because caches have limited size, cache management is
an important design problem.

I/O systems:- The I/O system consists of several components:

(1) A memory-management component that includes buffering, caching and spooling.

(2) A general device-driver interface.

(3) Drivers for specific h/w devices.

Protection and security: Protection is a mechanism for controlling the access of processes or users to the resources defined by a computer system. Protection can improve reliability by detecting latent errors at the interfaces between component

A system can have adequate protection but still be prone to failure and allow inappropriate access. Consider a user whose authentication information is stolen. The data could be copied or deleted, even though file and memory protection are working. It is the job of security to defend a system from external and internal attacks. Such attacks spread across a huge range and include viruses and worms, denial of service attacks, identity theft and theft of service. Prevention of some of these attacks is considered as o.s. function on some systems and in some other systems, it is left to additional s/w.

Distributed Systems:- A distributed system is a collection of physically separate, possibly heterogeneous computer systems that are networked to provide the users with access to various resources that the system maintains. Distributed systems depend on networking for their functionality. Networks vary by the protocols used, the distance between nodes and the transport media.

P.T.O

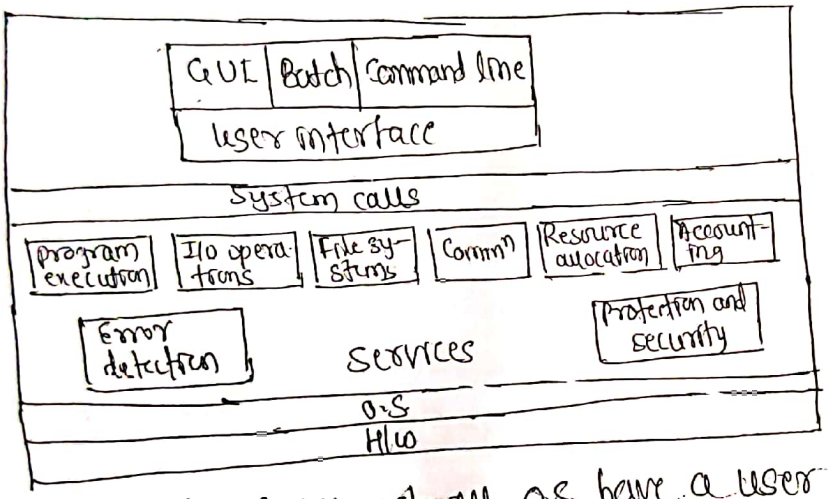
Operating System services:- We can view an O.S from different of view.

(1) One view focuses on the services that the o.s provides.

(2) Another on the interface that it makes available to users & programmers.

(3) On its components and their interconnections.

Specific services provided, differ from one O.S to another. These os services are provided for the convinience of the programmer, to make the programming task easier. Below fig shows one view of O.S services These set of O.S services provides functions that are helpful to users.



(1) User interface:- Allmost all OS have a user interface, which can take any one of the following forms.

* Command-line interface (CLI) — Uses text commands and a method to enter them.
* Batch interface - Commands are stored in files and those files are executed.
* Graphical user interface (GUI) - Menus are provided.

(2) Program Execution:- System must be able to load a prg into memory and to run that prg. Prg must be able to end the program normally or abnormally.

(3) I/O operations:- A running program may require I/O, which may involv a file or an I/O. users usually cannot control I/O devices directly. ∴ O.S must provide a means to do I/O operations.

(4) File-system manipulation:- Programs need to read and write files and directories. They also need to create and delete them by their names.

P.T.O.

(5) Communication :- Communication happens between the processes to exchange the information. It may happen between the processes that are executing on same or different computers, tied together by a computer N/w. This may be implemented via shared memory or through message passing, in which packets of information are moved between processes by the O.S.

(6) Error detection :- O.S needs to be constantly aware of possible errors. Errors may occur in CPU and memory h/w, in I/O devices and in the user program. For each type of error, O.S. should take appropriate action to ensure correct and consistent computing.

Another set of functions exists not for helping the user but for ensuring the efficient operation of the system itself. These are,

(1) Resource allocation :- When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Different types of resources have to be managed by O.S Some may have special allocation code whereas others have general request and response code.

(2) Accounting :- We want to keep track of which users use how much and what kind of comp. resources. This record keeping may be used for accounting or for simply accumulating usage statistics.

(3) Protection and security :- Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password to gain access to system resources.

System calls :- System calls provide an interface to the servic made available by an O.S. These calls are generally available as routines written in C and C++, although certain low lev tasks may need to be written using assembly-language instructions.

How system calls are used? Let us see how system calls are used through the following example.

eg: Write a program to read data from one file and copy them to another file.

Following are the steps to do it.

(1) Names of the I/p and O/p files are required. Names can be specif in different ways depending on O.S. design.

* Ask the user for the names of 2 files. (Requires system calls).

(i) To write a prompting message screen.

(2) Read from the K.B.

* On mouse-based and icon-based systems, a menu of file names is usually displayed on a window. User can then select the sour name and a window can be opened for the destination name to be specified. This sequence also requires many I/O system calls.

(2) Once the two file names are obtained, program must open the input file and create the output file. Each operation requires another system call.

There are also possible error conditions for each operation. Re

* It may not find a file with that name or the file is protected against access. Then program should print a message on the consol (another sequence of system calls), then terminate the program (" " " " ") abnormally.

(3) If the I/p file exists, we must create new O/p file. If the file with same name exists, this may cause the program to abort (a systen call) or we have to delete the existing file (system call) and create a new one (system call). In an interactive system, ask the user

to replace the existing file (system call) or to abort the progra

(4) Both the files are setup. Data from i/p file will be read (system and write to the o/p file (system call). Each read and write opera should return status information regarding possible error condition

(5) After the entire file is copied, program may close the file (s·c), write a message on the console or terminal (s·c)

From the above example, we can see that even simple programs may make heavy use of O.S. Hence systems execute thousands of system calls per second. This system call sequence is as shown below.


```
Acquire i/p file name
write prompt to screen
Accept i/p
Acquire o/p file name
write prompt to screen
Accept i/p
open the i/p file
If file doesn't exist, abort
create o/p file
If file exists, abort.
Loop
    Read from i/p file
    write to o/p file
until Read fails
close o/p file
write msg to screen
Terminate Normally
```

Most programmers never see this level of detail. Application developers design programs according to an application programming interface (API). Every API specifies a set of functions that are available to an applicat programmer and these functions typically invoke the actual system calls on behalf of the Application programmer. There are 3 types of AP (1) WIN 32 API (2) POSIX-API (3) Java API.

Example of a standard API: Consider a ReadFile() function in a Win32 API appears as follows.

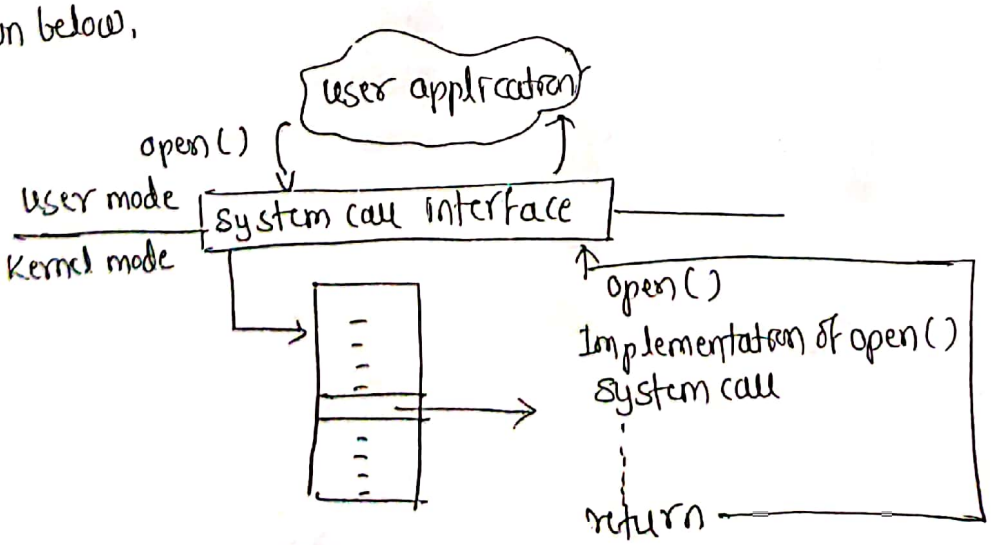
```
return value
    ↓
BOOL    ReadFile c (HANDLE  file,
    ↑               LPVOID  buffer,
function name       DWORD   bytes TO Read,
                    LPWORD  bytes Read,
                    LPOVERLAPPED ovl);
```

parameters

Description of parameters passed to ReadFile() is as follows.

* HANDLE file – the file to be read.
* LPVOID buffer – a buffer where the data will be read into and written from.
* DWORD bytes To Read – no of bytes to read into the buffer.
* LPDWORD bytes Read – " " " read during the last read.
* LPOVERLAPPED ovl – indicates if overlapped, I/O is being used.

Typically a no. is associated with each system call and the system-call-interface maintains a table indexed according to these numbers. The SCI then invokes the intended system call in the os ~~and~~ Kernel and returns the status of the system call and any return values. Caller need not know how the system call is implemented or what it does during execution. The relationship between an API, system call interface and the O.S. is as shown below.



Types of system calls :- Can be grouped into 6 major categ.

(1) Process control (2) File manipulation (3) Device manipulati.

(4) Information maintenance (5) Communications (6) Protection.

* Process Control

a. end, abort
b. load, execute
c. Create process, terminate process
d. get process attributes, Set process attributes.
e. wait for time.
f. wait event, signal event
g. allocate and free memory.

* File management

a. Create file, delete file
b. open, close.
c. read, write, reposition
d. get file attributes, set file attributes.

* Device management

a. request device, release device
b. read, write, reposition.
c. get device attributes, set device attributes.
d. logically attach or detach devices.

* Information maintenance

a: get time or date, set time or date.
b. get system data, set system data
c. get process, file or devic attributes.
d. set process, file or device attributes.

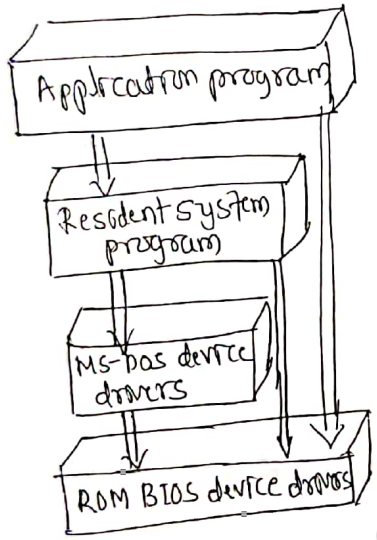
* Communications

a. Create, delete communication connection.
b. Send, receive messages.
c. transfer status information
d. attach or detach remot devices.

operating system structure: A common approach is to partition the task into small components rather than have one monolithic system. Each of these modules should be a well-defined portion of the system, with carefully defined i/ps, o/ps and functions.

Simple structure:- Many commercial O.S do not have well-defined structures. Such systems started as small, simple and limited systems. ~~and~~ eg: MS-DOS

It was written to provide the most functionality in the least space, so it was not divided into modules carefully. Fig. below shows its structure.



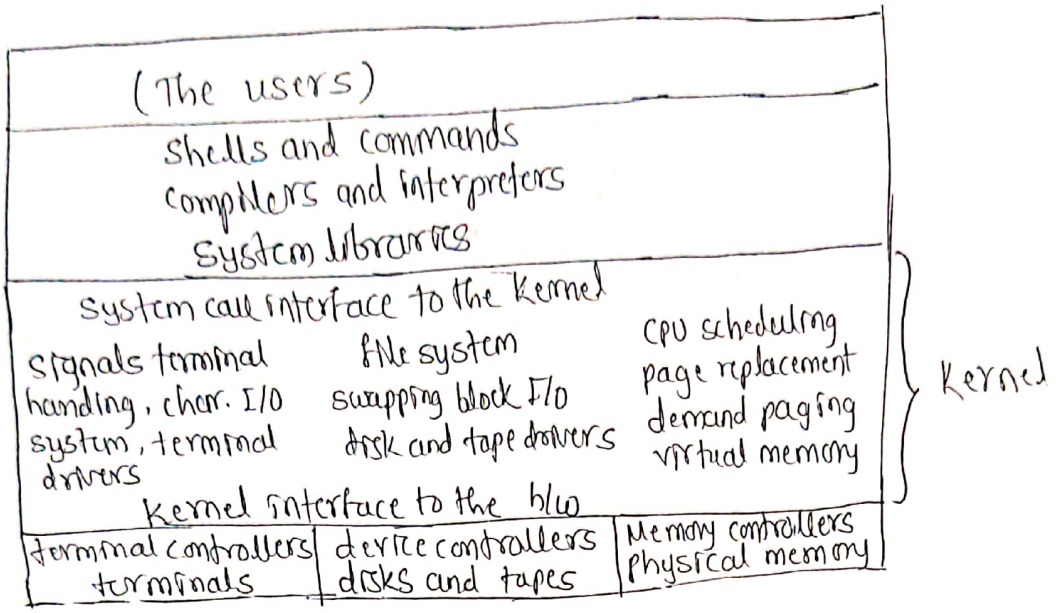
MS-DOS layer structure:- In MS-DOS, the interfaces and levels of functionality are not well separated.

eg: 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 programs causing entire system crashes when user programs fail. It was also limited by the h/w.

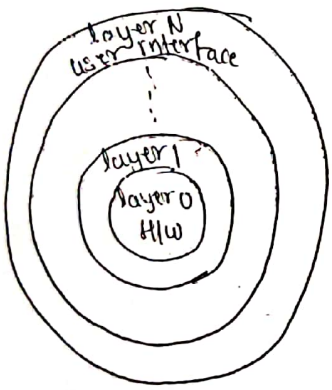
eg: (2) Another example of limited structuring is the original UNIX OS. Like MS-DOS, UNIX initially was limited by h/w functionality. It consists of two separable parts: the kernel & the system program. Kernel is further separated into a series of interfaces and device drivers. Traditional UNIX system structure is as shown below.

Everything below the system-call interface and above the physical h/w is the kernel. The kernel provides the file system, CPU scheduling, memory management and other OS functions through the system calls. This structure was difficult to implement and maintain.

Traditional UNIX system structure :-



Layered Approach :- With proper h/w support, O.S can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS and UNIX systems. A system can be made modular in many ways. One method is the layered approach in which O.S is broken into a no of layers.



Bottom layer (layer 0) is the h/w and layer N is the user interface. A typical O.S layer consists of data structures and a set of routines that can be invoked by higher-level layers.

The main advantage of this approach is the simplicity of construction and debugging. The layers are selected so that each uses functions and services of only lower-level layers. This approach simplifies debugging. The first layer can be debugged without any concern for the rest of the system, ∵ it uses only the h/w to implement the functions. If an error is found, during debugging of a particular layer, then error must be on that layer, ∵ the layers below that are already debugged.

Each layer is only implemented with those operations provided by lower level layers. A layer does not need to

know how these operations are implemented but it should only know what these operations do.

Major difficulty with the layered approach involves appropriately defining the various layers. eg: device driver for backing store must be at a lower level than the memory-mgm routines ∵ memory mgmt requires the ability to use the backing-store.

A final problem with the layered approach is that, they tend to be less efficient than other types.
eg: 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-mgmt layer, which in turn calls the CPU-scheduling layer, which is then passed to the hlw. At each layer, the parameters may be modified, data may need to be passed and so on. Each layer adds overhead to the syste call, the net result is a system call that takes longer tim than does one on a non-layered system.

Microkernels :- As UNIX expanded, the kernel became large and was difficult to manage. In the mid-1980s, researcher developed an O.S. called Mach, which modularized the kernel using the microkernel approach. This method structure the O.S. by removing all non-essential components from the kernel and implementing them as system-level and user-level programs. The result is a smaller kernel. To achieve this few services should remain in the kernel and few services should be implemented in user space. However this approach provides minimal process and memory management, in addition to a communication facility. The main function of the micro-kernel is to provide a communication facility between the client program and the various services that are also running in user space. Communication is provided by message passing.

eg: If the client program ~~and~~ wishes to access a file, it interact with the file server. The client program and server never interact directly. Rather they communicate indirectly by exchanging messages with the microkernel.

Advantages :- (1) Ease of extending the O.S - All new services are added to user space and do not need any modification of the kernel. When it is to be modified, changes tend to be fewer, because microkernel is a small kernel.

(2) Porting will be easy.

(3) Provides more security and reliability — because most services are running as user rather than kernel processe

Several O.S. make use of Microkernels.

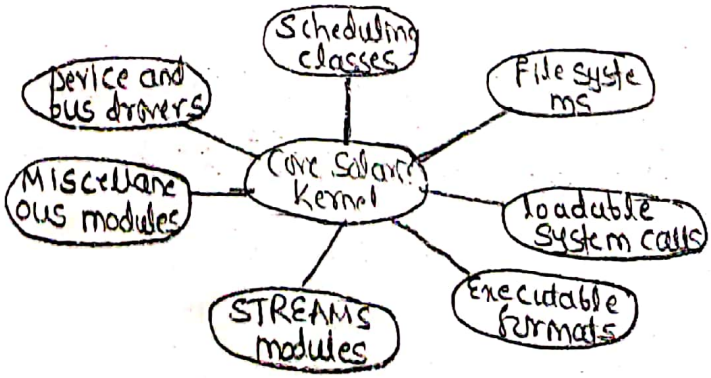
eg:- Tru64 UNIX, MAC OS X ~~Kernel~~, QNX

Disadvantage :- Suffer from decreased performance due to increase system function overhead.

Modules :- The best current methodology for O.S design involves using of object-oriented programming techniques to create a modular kernel. Here the kernel has a set of core components and links in additional services either during boot time or during run time. Such a strategy uses dynamically loadable modules and is common in modern implementations of UNIX such as Solaris, Linux and Mac O.S-X. Solaris O.S. structure is organized around a core kernel with seven types of modules as shown below.

(1) Scheduling classes
(2) File systems
(3) Loadable system calls
(4) Executable formats
(5) STREAMS modules
(6) Miscellaneous modules
(7) Device and bus drivers.

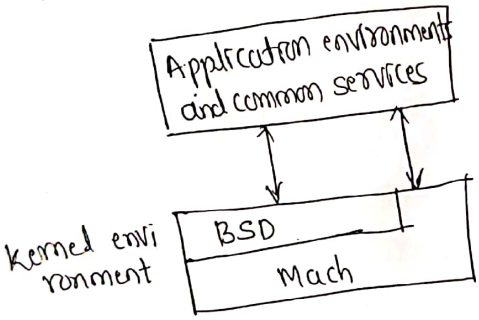
Solaris loadable modules



h a design allows the kernel to provide core services yet also lows certain features to be implemented dynamically.
g: device and bus drivers for specific h/w can be added to the kernel and support for different file systems can be added as loadable modules. This structure resembles a layered system in that, each kernel section has defined, protected interfaces but it is more flexible than a layered system, in that any module can call any other module. Further similar to microkernel approach, primary module has only core functions and knowledge of how to load and communicate with other modules. ~~do not~~ Comparatively this approach is more efficient because modules do not need to invoke message passing in order to communicate.

Hybrid Structure :- eg: Mac OS X

It is a layered system in which one layer consists of the Mach microkernel. Structure of Mac OS X is as shown below.



BSD (Berkley S/w Distributions)

Here top layers include application environments and a set of services providing a graphical interface to applications. Below these layers is the kernel environment, which consists primarily of the Mach microkernel and the BSD kernel.

* Mach provides memory mgmt, support for remote procedure calls (RPC) and inter-process communication (IPC) facilities, including message passing and thread scheduling.
* BSD component provides a BSD command line interface, support for networking and file systems, and implementation of POSIX APIs. In addition to Mach and BSD, kernel environment provides an I/O Kit for development of device drivers & dynamically loadable modules.

Features of UNIX OS:

UNIX can be used by multiple users:- Windows is essentially a single-user system where the memory, CPU and harddisk are all dedicated to a single user. In UNIX, the resources are actually shared between all users. UNIX is a multi-user system. Computer breaks up a unit of time into several segments and each user is allotted a segment. At any point of time, machine will be doing the job of a single user. The moment the allocated time expires, the previous job is kept in pending and the next user's job is taken up. This process goes on till the clock has turned full-cycle and the first user's job is taken up once again. This, the kernel does several times in one second and keeps all ignorant and happy. This all is handled by the process management system.

One user can run multiple tasks :- In UNIX, a single user can also run multiple tasks concurrently. It is useful for a user to edit a file, print another one on the printer, send e-mail and browse the web, all without leaving any of the applications. The kernel is designed to handle a user's multiple needs. This is known as multitasking. In this situation, only one job runs in the foreground while the rest run in the background. One can switch jobs between background and foreground, suspend or even terminate them. This is also an important component of the process management system.

The Featureless File:- The 2 powerful concepts in UNIX are,

a. Files have places b. Processes have life.

ie Files can be placed at a specific location and also can be moved from one place to another.

A file to UNIX is just an array of bytes and can contain virtually anything - text, object code or a directory structure. However the dominant file type is text.

Pattern Matching :- UNIX support a pattern matching feature which helps in reducing the typing load.
eg:- * can be affixed to a string and a list of filenames will be displayed with those characters.
eg: prog* (all files names that stcart with prog will be displayed).

Programming facility :- UNIX shell is also a programming language, it was designed for a programmer, not a casual end user. It has all components like control structures, loops and variables, that establish it as a powerful programming language in its own right. These features are used to design shell scripts - programs that also include UNIX commands in their syntax. Shell scripts are extremely useful for text manipulation tasks. Many of the system's functions can be controlled and automated by using these shell scripts.

Portability and system calls :- UNIX is written in C. Though there are thousand commands handling specialized functions, they all use a handful of functions called system calls. These calls are built into the kernel and all library functions and utilities are written using them. All UNIX flavors use the same system calls.

General features of a command :- A UNIX command consists of a single word generally using alphabetic characters.

Internal and external commands :- "Shell" is a special co
that starts running the moment you log-in. Shell takes th
command that you enter as its input and looks at its own
PATH variable to find out where it is located.

Since ls is a file having an independent existence in the /bin directory (or /usr/bin), it is called an external command. Most commands are external in nature but there are some that are not really found anywhere or not executed even if they are found.

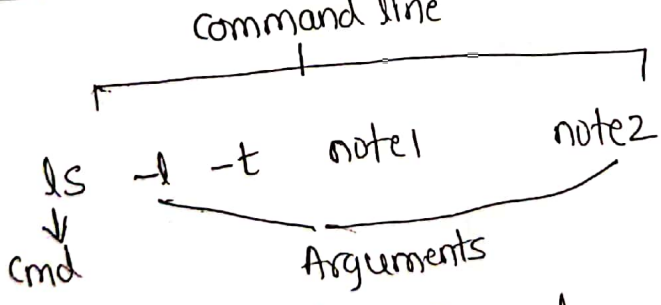
① eg: echo command

$ echo

echo is a shell built-in. When you type echo, the system won't look in its PATH to locate it. Rather it will execute it from its own set of built-in commands that are not stored as separate files. These built-in commands of ~~echo~~ which echo is a member, are known as internal commands.

② The type command itself is a built-in and whether or not you are able to execute it depends on the shell you use.

Command structure :- Unix command.



The entire command has 5 words separated by spaces. ls command here has 4 arguments. Similarly am and i are arguments of the who command.

specifies block size (it 18). This is how tar command is used to back up all files in the current directory to a floppy diskett. Here is an example, how we can combine various options.

eg: tar -cvfb /dev/fd0 18 * (Note that we should also plac their own parameters in the same sequence.)

Exceptions and variations:- All commands don't compulsorily use options and arguments.

eg: (1) clear - don't accept any arguments.

(2) who, date - May or may not be specified with arguments.

(3) ls - permits more variations. (or without any arguments, with only options, with only file-names and can use combination of both.)

Difference between internal and external commands:

Internal	External
(1) Commands are built into the shell.	(1) Are not built into the shell.
(2) No process needs to be spawned for the execution	(2) New process has to be spawned.
(3) Fast in execution	(3) Comparatively slow.
(4) eg: cd, pwd, echo etc	(4) ls, cat

Options and filenames :- The two arguments that start with a — are known as options. An option changes a command's default behavior because -l and -t options show their attributes.

The command with its arguments and options is entered in one line that is referred to as command line. This line can be considered complete only after the user has hit the Enter key.

Some commands accept a single filename, some accept two and indefinite also (depends on system)

eg. ls -l -a -t chap01 chap02 chap03

cp chap01 chap02 prgs

rm chap01 chap02

If there are any error messages that has been generated by the command and not by the shell.

Note: An option is normally preceded by a minus sign (-) to distinguish it from other arguments. And there must not be any whitespace between — and the option letter.

Combining options :- UNIX has few rules that govern the use of options. Options that begin with a — sign can be normally combined with only one — sign.

eg: ls -l -a -t In this example, ~~other~~ options, l, a and t can be combined as,

ls -tal or ls -atl.

Some commands won't let you combine options in the way you did just as above, because they are followed by their own arguments as shown below.

tar -cv -f /dev/fd0 -b 18 *

↑ ↑ option parameters

Here -f is followed by the filename /dev/fd0 and -b