Chapter - 1

What is an Operating System?

⇓

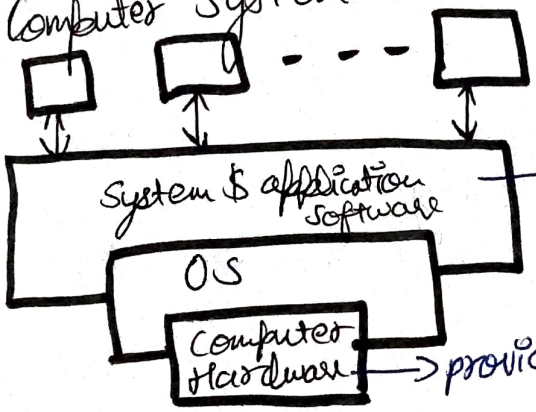
program that acts as an intermediary between a user of a computer and the compo. hardware.

Goals :- ⇒ Execute user programs & make solving user problems easier.

⇒ Make computer system convenient to use

⇒ Use the computer hardware in an efficient manner.

Computer System Structure.



→ compiler, text editor, assembler, database system.

→ provides basic computing resources.

OS

↓ "resource allocator"

⇒ manages all resource

⇒ decides b/w conflicting requests for efficient and fair resource use.

↓ control program.

⇒ controls execution of programs to prevent errors & improper use of computer.

⇒ "The one program running at all times on the computer" is the kernel.

Everything else is either → a system program or

→ an application program.

Computer Startup

⇒ "bootstrap program" is loaded at power-up or reboot.

⇒ stored in ROM or EPROM ⇒ known as firmware

⇒ initializes all aspects of system

⇒ loads OS kernel and starts execution.

Computer System Organization

⇒ One or more CPUs, device controllers connect through common bus providing access to shared memory.

⇒ Concurrent execution of CPUs and devices competing for memory cycles.

Computer-System Operation

⇒ I/O devices & CPU ⇒ execute concurrently.

⇒ Each → device controller → is in charge of particular device types

↓

local buffer.

⇒ CPU ⇒ moves data from/to main memory to/from local memory.

⇒ Input/Output is from device to local buffer of controller.

⇒ Device controller informs CPU that it has finished its operation by causing an interrupt.

↓

Interrupt Handling

Common functions of Interrupts

⇒ An operating system is interrupt driven.

Interrupt is generally a signal emitted by hardware or software when a process or an event needs immediate attention.

In I/O devices one of the bus control lines is dedicated for this purpose and is called the Interrupt Service Routine (ISR).

When a device raises Interrupt (process 2) → transfers control to ISR through Interrupt vector that contains address of all service routines

→ Then it loads PC with address of first instruction of ISR

Before loading → Address of interrupted instruction is moved or saved to Interrupt architecture

→ As preserves state of CPU by storing registers and PC.

→ Determine interrupt type → requested action performed → Interrupt is enabled → Interrupted program (i+1) is resumed.

Interrupt handling (Interrupt Timeline from ppt).

Polling

first device encountered with IRQ bit set is the device to be served first.

easy, but time waste interrogating IRQ bit of all devices.

Vectored

A device requesting an interrupt identifies itself directly by sending a special code to the processor over the bus.

can be starting address of ISR

enables processor identify.

Interrupt Nesting

I/O device is organised by priority structure

I/O Structure

Storage Definitions & Notions

Bit :- Basic unit of computer storage
↳ store either 0 / 1

Byte :- Smallest convenient chunk of storage
1 Byte = 8 bits.

Word :- Computer architecture native unit of data.
↳ made of one or more bytes.

1 KB = 1024 Bytes
1 MB = $(1024)^2$ Bytes → 1 Million
1 GB = $(1024)^3$ Bytes → 1 billion
1 TB = $(1024)^4$ Bytes
1 PB = $(1024)^5$ Bytes

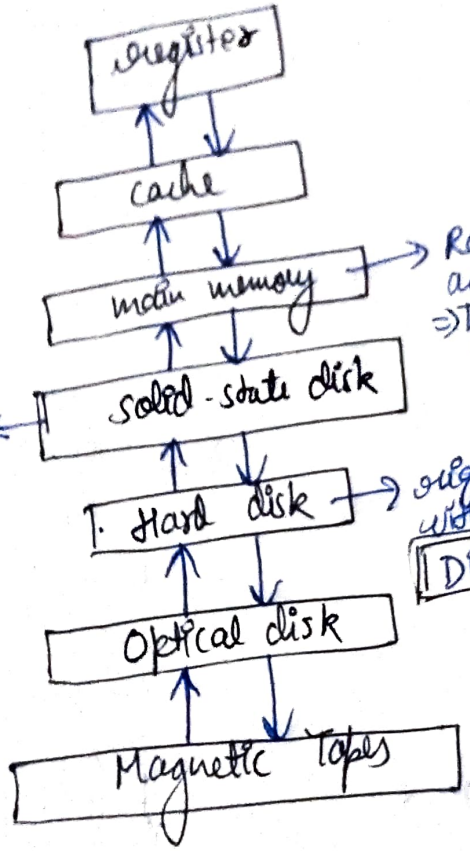
Storage Structure & Hierarchy

Storage systems organized in hierarchy :-

⇒ Speed
⇒ cost
⇒ Volatility

Caching :- copying information into faster storage system; main memory can be viewed as a cache for secondary memory

Device Driver :- for each controller to manage I/O
⇒ provides uniform interface b/w controller → kernel



Caching :- ⇒ Info. normally performed at many levels in a comp.

⇒ Info. copied from :- slower to faster storage temporarily.

⇒ smaller than storage being cached.

⇒ cache management info. design problem

⇒ cache size & replacement policy.

storage (cache) checked first to determine if information is there :-

(i) Yes → used directly from cache (fast)

(ii) No → data copied to cache & used.

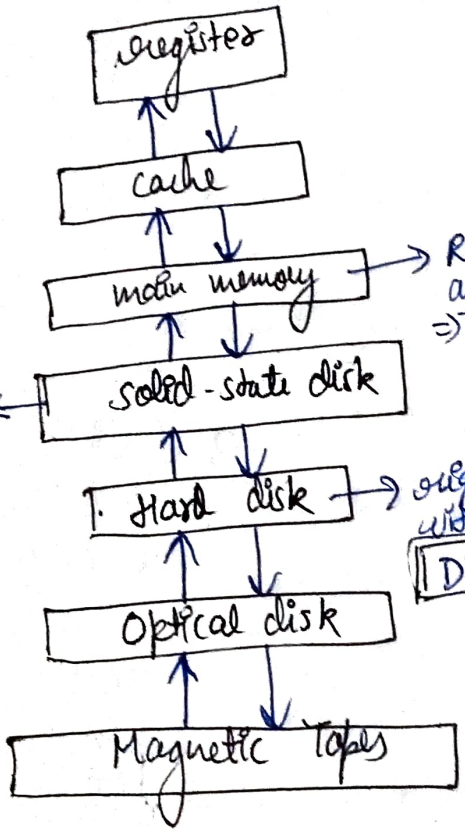
Storage Structure & Hierarchy

Storage systems organised in hierarchy :-

⇒ Speed
⇒ cost
⇒ Volatility.

Caching :- copying information into faster storage system, main memory can be viewed as a cache for secondary memory.

Device Driver :- for each controller to manage I/O
⇒ Provides uniform interface b/w controller → kernel.



Caching :- Info. principle performed at many levels in a comp.

⇒ Inform. copied from :- slower to faster temporarily.

⇒ smaller than ~~faster~~ storage being cached.
⇒ Cache management info. design problem
⇒ Cache size & replacement policy.

faster storage (cache) checked first to determine if information is there :-
(i) Yes → used directly from cache (fast)
(ii) No → data copied to cache & used.

Direct Memory Access Structure

⇒ Used for high speed I/O devices able to transmit information at close to memory speeds.

⇒ Device controllers transfers block of data from buffer storage directly to main memory without CPU intervention.

⇒ Only one interrupt is generated per block, rather than one per byte.

Computer-System Architecture

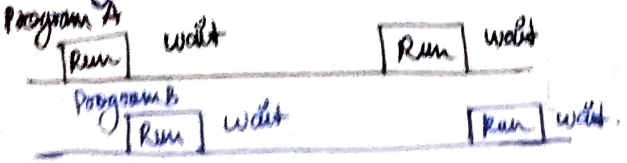
General-purpose processors are categorised as follows:-

1.) Single-Processor Systems
2.) Multiprocessor Systems.
3.) Clustered Systems.

Operating System Structure

1. Multiprogramming → context switch

⇒ Computer running more than one program at a time.

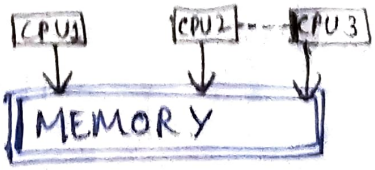


Ex:- Running excel and firefox simultaneously.

Here, CPU does not waste its resources and gives program B an opportunity to run.

2. Multiprocessing → parallel processing

⇒ Computer using more than one CPU at a time.



⇒ The term also refers to the ability of a system to support more than one processor within a single computer system.

⇒ These multi-processors share the computer bus, sometimes the clock, memory & peripheral devices also.

Difference between Multi programming & Multiprocessing

⇒ A System can be both multi-programmed by having multiple programs running at the same time and multiprocessing by having more than one physical processor.

⇒ Difference b/w multiprocessing and multi-programming is that Multiprocessing is basically executing multiple processes at the same time on multiple processors, whereas multiprogramming is keeping several programs in main memory and executing them concurrently using a single CPU only.

⇒ Multiprocessing occurs by means ~~of~~ of parallel processing whereas ~~Multi processing~~ Multi-programming occurs by switching from one process to other (phenomenon called as context switching).

3. Multitasking ⟹ (time sharing + multiprogramming)

⇒ Tasks sharing a common resource (like 1 CPU).

⇒ multi-tasking refers to execution of multiple tasks (say processes, programs, threads etc.) at a time.

⇒ Multitasking is a logical extension of multi programming. The major way in which multitasking differs from multi-programming is that multiprogramming works solely on the concept of context switching whereas multitasking is based on time sharing alongside the concept of context switching.

⇒ Multi-tasking systems working: (From ppt)
CPU makes the processes to share time slices between them and execute accordingly. As soon as time quantum of one process expires, another begins its execution.

Chapter-2

Operating System Services

1. Provide an environment for execution of programs and services to programs and users.
2. One set provides functions that are helpful to the user:-
   - ⇒ User Interface
   - ⇒ Program execution
   - ⇒ I/O Operations.
   - ⇒ File-system manipulation
   - ⇒ Communications
   - ⇒ Error detection
3. Another set of OS functions exists for ensuring the efficient operation of the system itself via resources sharing:-
   - ⇒ Resource allocation
   - ⇒ Accounting.
   - ⇒ Protection and security.

User Operating System Interface

There are two fundamental approaches for users to interface with the OS:-

1. Provides:- Command-Line Interface (CLI) or Command Interpreter that allows users to directly enter commands that are to be performed by OS.
   - ⇒ Sometimes implemented or included in the kernel.
   - ⇒ Others, such as Windows XP and UNIX, treat CI as a special program.
   - ⇒ On systems with multiple command interpreters to choose from the interpreters are known as shells.
   - ⇒ Sometimes commands built-in, sometimes just names of programs.
     If latter, adding new features doesn't require shell modification.

2. GUI:- Graphical user Interface

(a) User-friendly desktop:- metaphor interface

- ⇒ Usually, mouse, keyboard & monitor
- ⇒ Icons represent files, programs, actions, etc.
- ⇒ Various mouse buttons over objects in interface cause various actions (provide information, options, execute function, open directory (known as a folder)
- ⇒ Invented at Xerox PARC.

* Many systems now include both CLI and GUI interfaces:-

⇒ Microsoft Windows is GUI with CLI "command" shell.

⇒ Apple Mac OS X is "Aqua" GUI interface with UNIX kernel underneath and shells available.

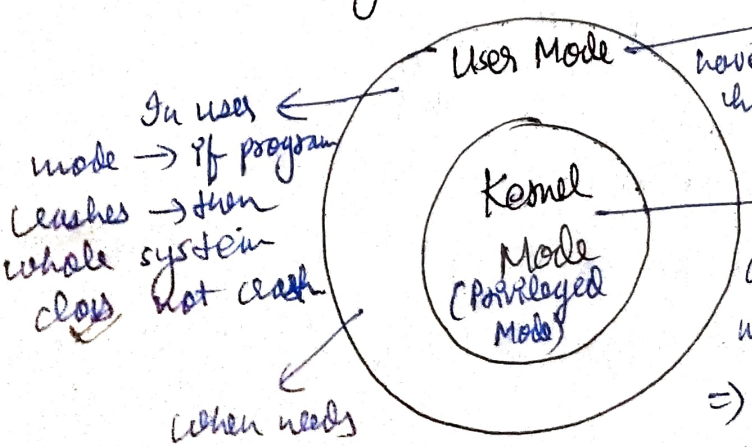
⇒ Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME).

* Touchscreen devices require new interfaces.
  - ⇒ Mouse not possible or not required.
  - ⇒ Actions and selection based on gestures.
  - ⇒ Virtual keyboard for text entry.
  - ⇒ Voice commands.

System Calls

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

Two modes in which a system program can execute:-



In user mode → if program crashes → then whole system does not crash.

That program does not have direct access to the memory hardware & resources.

That program has direct access to hardware, memory & resources.

⇒ If program crashes at kernel mode → it can cause whole system crash.

When needs system call. access → makes a call to OS → then switch for an instance from user to kernel at context switching.

⇒ System call is the programmatic way in which a computer program requests a service from the kernel of OS.

⇒ Typically written in high level language (C or C++)

⇒ Accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use.

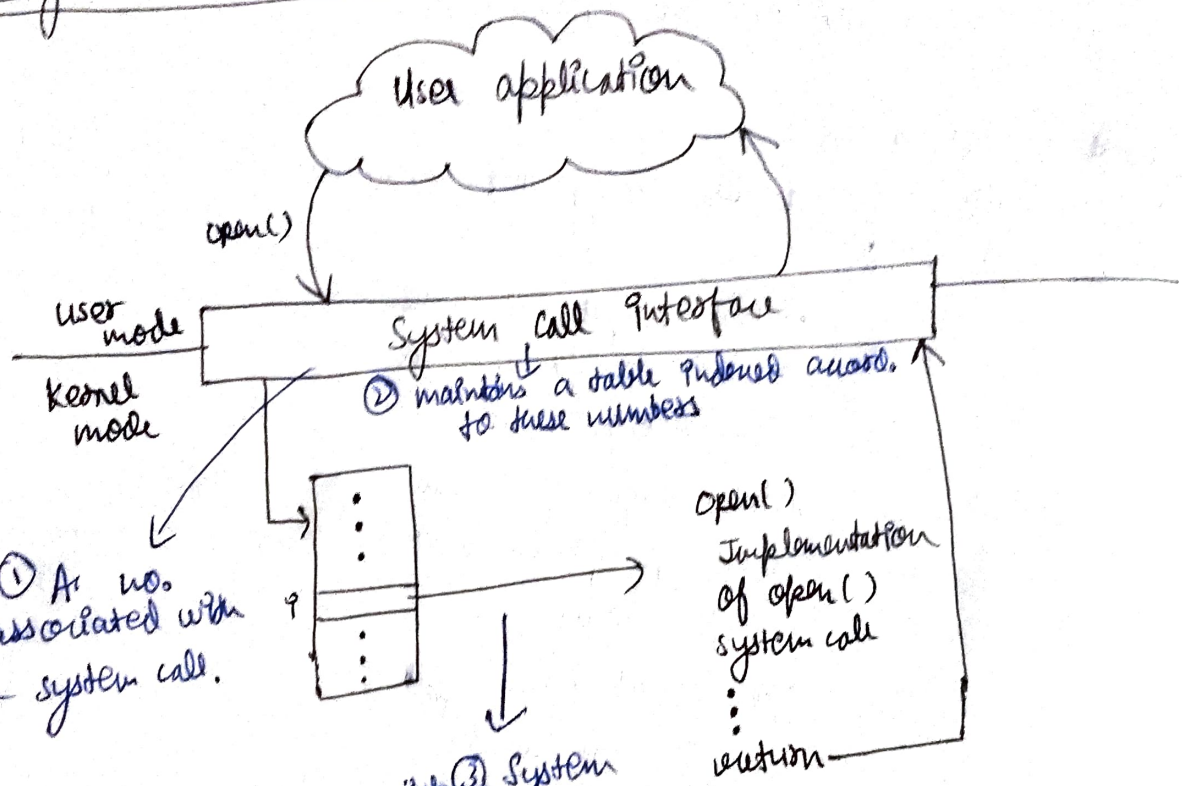
Eg:- Win32 API for windows.
POSIX API for POSIX-based systems.
JAVA API for JVM.

Example of System call :-
System call sequence to copy the contents of one file to another file.

Aquire Input file name
Write prompt to screen
Take/Accept input
Aquire Output file name
Write prompt to screen
Accept Input
Open Input file,
If not exist, abort
Create output file,
If already exist, abort.

Loop
Read from Input file
Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally.

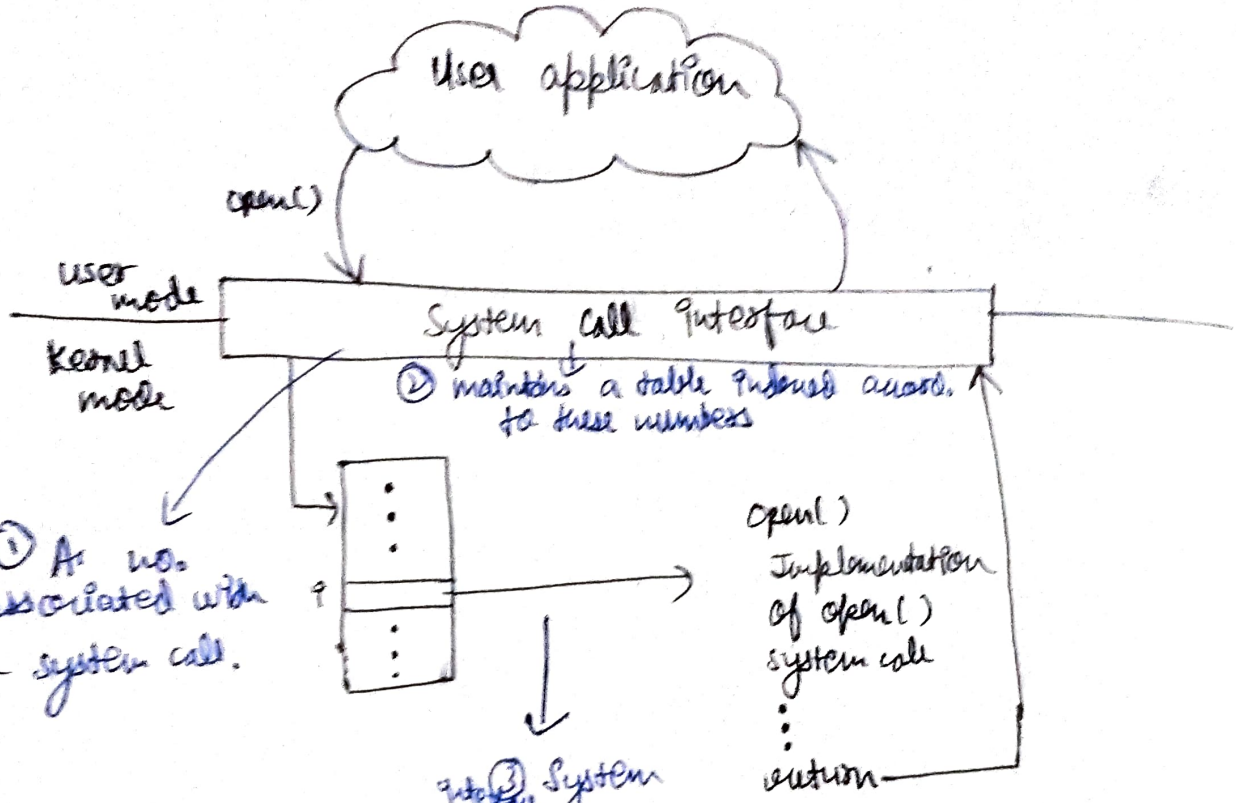
System Call Implementation



Caller need to know nothing but :-

=> needs to obey API and understand what OS will do as a result call.

System Call Implementation



Caller need to know nothing but :-

=> needs to obey API and understand what OS will do as a result call.

System Call Parameter Passing

Three general methods used to pass parameters to the OS:-

1. Simplest:- pass parameters in registers.
2. Parameters stored in block, or table, in memory, and address of block passed as a parameter in a register.
   ↳ approach taken by Linux and Solaris.
3. Parameters placed or pushed onto the stack by the program and popped off the stack by OS.

* Block and stack methods do not limit the no. or length of parameters being passed.

Types of System Calls

1. Process control
   - create process, terminate process
   - end, abort
   - load, execute
   - get process attributes, set process attributes
   - wait for time
   - wait event, signal event
   - allocate & free memory.
   - Dump memory if error.
   - Debugger for determining bugs, single step execution.
   - Locks for managing access to shared data b/w processes.

2. File management
create file, delete file
open file, close file
read, write, reposition
get & set file attributes.

3. Device Management
request device, release device
read, write, reposition
get device attributes, set device attributes
logically attach or detach devices.

4. Information maintenance
get time or date, set time or date
get system data, set system data
get and set process, file or device attributes.

5. Communications
create, delete communication connection
send, receive messages if message passing model to host name or process name.
↳ from client to server.
shared memory model create & gain access to memory regions.
transfer status information.
attach and detach remote devices

6. Protection
Control access to resources
Get and set permissions.
Allow and deny user access.

2. File management
create file, delete file
open file, close file
read, write, reposition
get & set file attributes.

3. Device Management
request device, release device
read, write, reposition
get device attributes, set device attributes
logically attach or detach devices.

4. Information maintenance
get time or date, set time or date
get system data, set system data
get and set process, file or device attributes.

5. Communications
create, delete communication connection
send, receive messages if message passing model to host name or process name.
↳ from client to server.
Shared memory model create & gain access to memory regions.
transfer status information.
attach and detach remote devices

6. Protection
Control access to resources
Get and set permissions.
Allow and deny user access.

System Programs

System programs provide a convenient environment for program development and execution.

Some of them are simply user interfaces to system calls, others are considerably more complex.

They are divided :-

1. File Management

⇒ Create, delete, copy, rename, print, dump, list and generally manipulate files and directories.

2. Status Information

⇒ Info - date, time, amount of available memory, disk space, no. of users.

⇒ Others provide detailed performance, logging and debugging info.

⇒ Typically, these programs format and print output to the terminal or other output devices.

⇒ Some systems implement a registry → used to store and retrieve configuration information.

3. File modification

⇒ Text editors to create and modify files.

⇒ Special commands to search contents of files or perform transformations of text.

4. Programming - language support

⇒ Compilers, assemblers, debuggers and interpreters sometimes provided.

5. Program loading and execution

Absolute loaders, relocatable loaders, linkage editors and overlay loaders, debugging systems for higher-level and machine language.

6. Communications

⇒ Provide the mechanism for creating virtual connections among processes, users and computer systems.

⇒ Allows users to send messages to one another's screens, browse web pages, send e-mail messages, log in remotely, transfer files from one machine to another.

7. Background services

⇒ Launch at boot time → sometimes → from system startup, then terminate
→ from system boot to shutdown

⇒ Provide facilities like disk scheduling, process scheduling, error logging, printing.

⇒ Run in user context not kernel context

⇒ known as services, subsystems, daemons.

8. Application Programs

⇒ Don't pertain to system and run by users

⇒ Not typically considered part of OS

⇒ Launched by command line, mouse click, finger poke.

Structures of OS

⇒ General-purpose as it is very large program

⇒ Various ways to structure ones-

① Simplest Structure - MS-DOS //

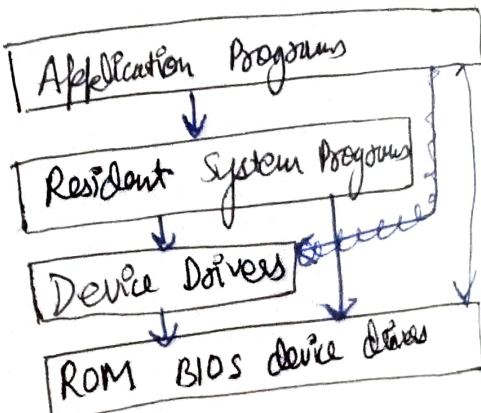
↳ does not have very well defined structure.

↳ most functionality in least space

↳ not divided into modules

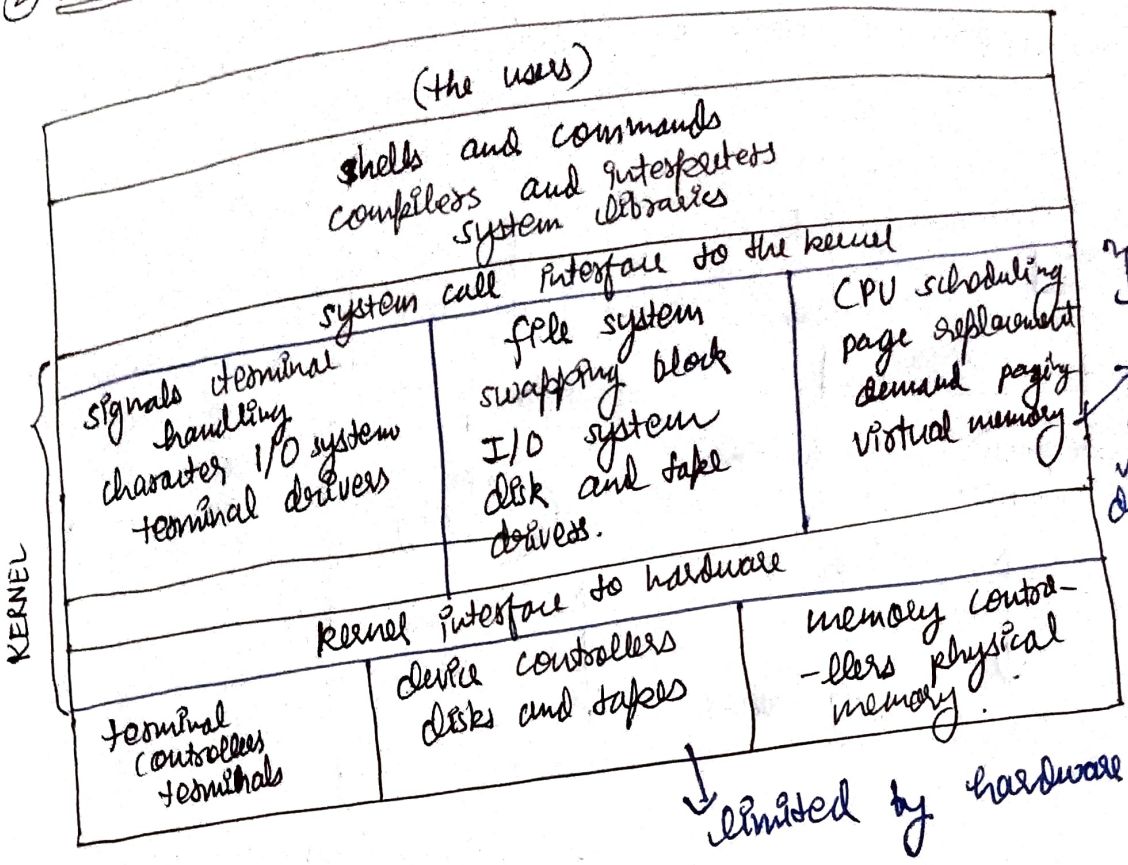
↳ Application programs are directly able to access Base hardware.

→ Interfaces and level of functionality are not well separated.



→ Vulnerable to errant or malicious programs.
↓
system crash.

② Non Simple Structure (Monolithic Structure) – UNIX



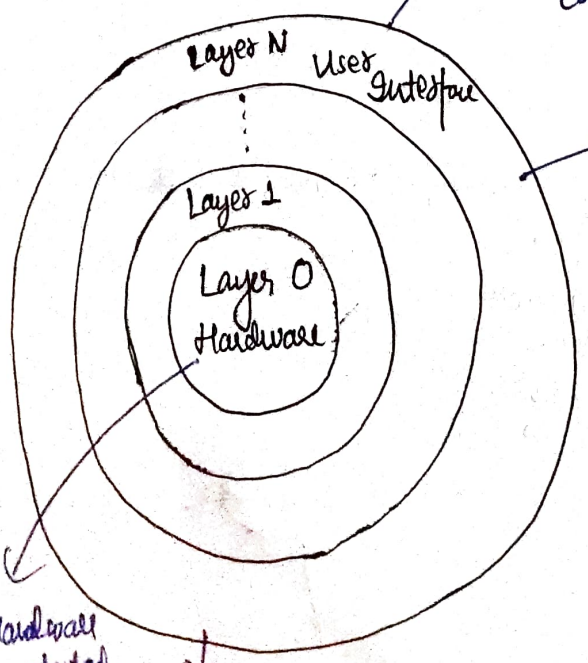
→ limited structuring

→ Implementation and maintenance very difficult due to low level.

↓ limited by hardware functionality

③ Layered Structure

→ divided into layers each built on top of lower layers.
Easy to implement (due to layers)



→ With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.

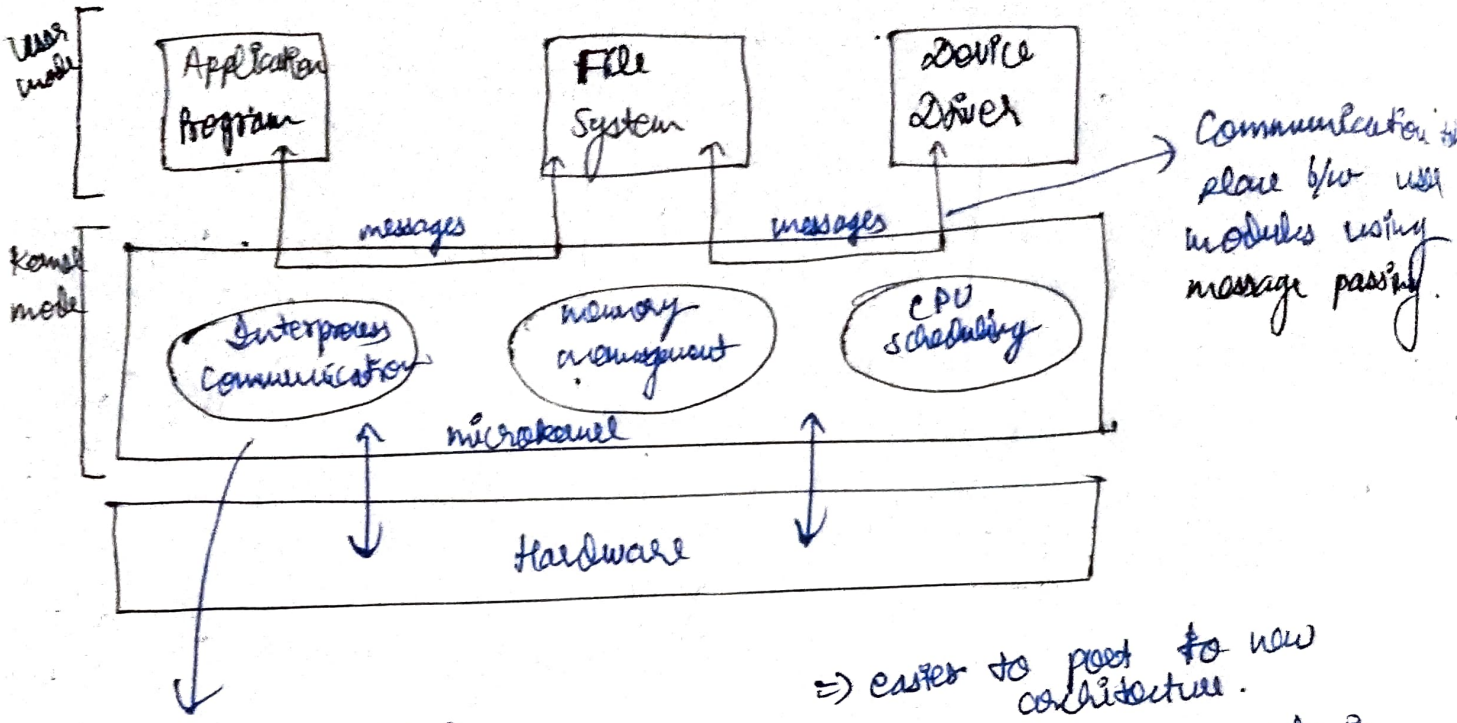
Eg: Layer → backing store (disk space)
↓ this layer has to be below memory management
as it needs.

Hardware is protected.

↓ Not efficient → as it does each layer request takes time.

④ Microkernel System Structure

We remove all non-essential components from the kernel and we implement them as system and user level programs.



⇒ Easier to extend microkernel.

⇒ easier to port to new architecture.

⇒ More reliable (less code in running kernel mode)

⇒ More secure.

Disadvantage

⇒ Performance overhead of user ~~kernel~~ space to kernel communication.

→ Eg- Linux, Solaris, etc.

⑤ Modules / Modular Structure

↳ similar to layers but more flexible

↓

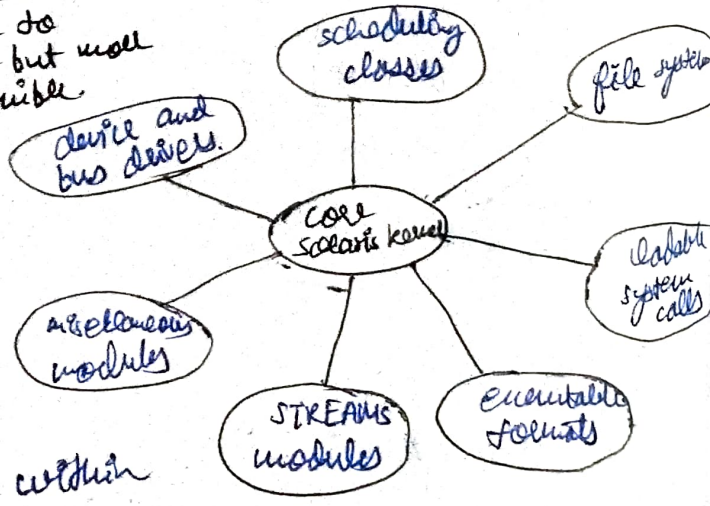
Modern OS implement loadable kernel modules.

⇒ Uses object-oriented approach.

⇒ Each core component is seperate.

⇒ Each talks to others over known interfaces.

⇒ Each is loadable as needed within



Hybrid Systems

↳ Most OS are not actually one pure model.

⇒ Hybrid combines multiple approaches to address performance, security, usability needs.

Ex:- 1. Linux & Solaris :- in kernel address space, so monolithic + modular
↓
dynamic loading of functionality.

2. Windows :- monolithic + microkernel
↓
for diff. subsystem personalities.

3. Apple MAC OS X → hybrid layered Aqua UI
+
Cocoa programming environment.