Textbk: Galvin 8/9th edition

Date: 3/10/24

Operating System

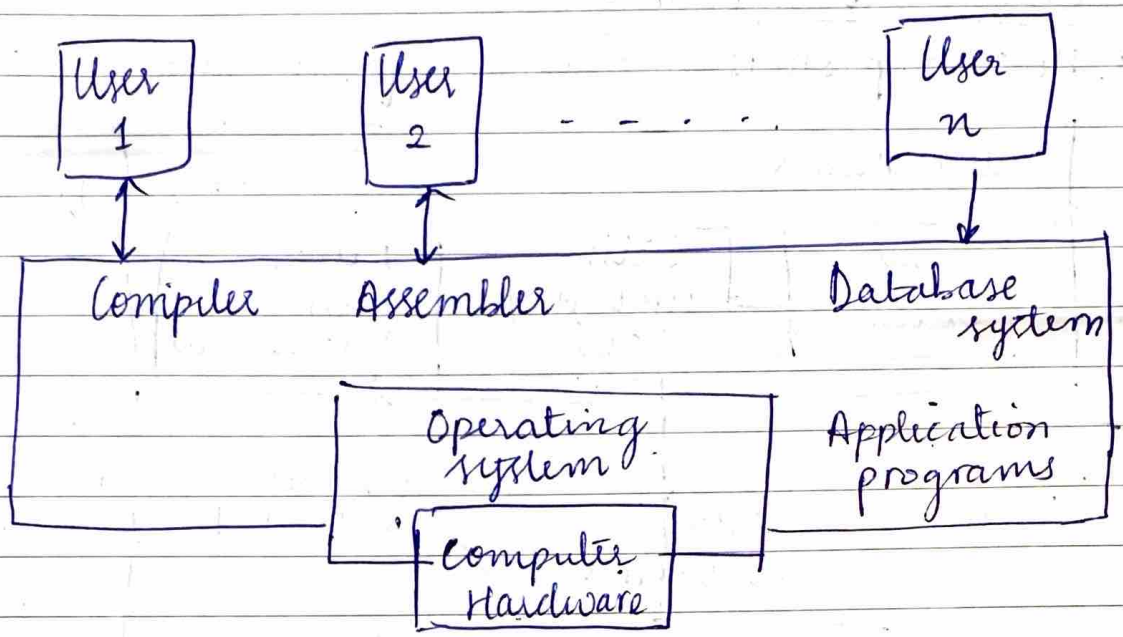


Fig: Parts of computer in OS

* Four layers → User, Appli, OS, H/W.
  H/W → I/O, memory, CPU.

* User can access H/W only through OS permission

* Functions of OS:
  Process, memory, I/O, file management

Goals / learn

* OS acts as interface b/w user and hardware.

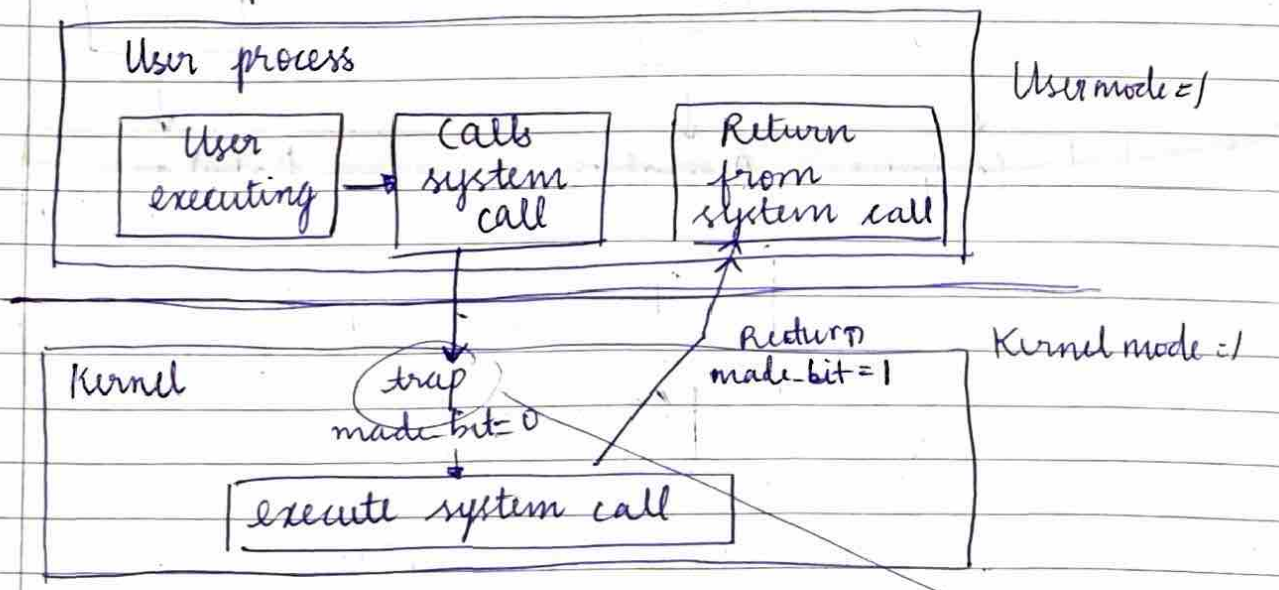
* Hardware cannot be replaced frequently unlike software if the hardware is injected/manipulated/corrupted so we use the OS to protect the hardware.

Date :

4/10/24

(*) Dual mode operation

(*) (*) * How OS protects hardware from user?



* mode is a bit where 0 is kernel
1 is user

* program — source code.

note: p1.obj → program
① (compiled version of source code)
② passive entity
③ in secondary memory / virtual / logical

p1.exe → process (running program) ①
state / instance.
② active entity
③ ~~resides~~ in main memory / physical

* CPU switches b/w the modes.

Date :

* When executing, loader (system program) loads the program from S.M to M.M to execute it.
because CPU is connected to the M.M (main memory)
→ because processor is directly connected

Q) Login → execute the progs. → execute it again and again → will it stay in main memory or go to SM?

Q) Which is better?
- let it (.exe) stay in M.M and move to S.M after logout (or) repeatedly it is loaded everytime executed.

* OS ⟷ Kernel mode. executes system program. (OS related tasks)
CPU → User mode ↓ executes application programs.

* When switched on, it is in kernel mode
* When login, it is in user mode. (user starts interacting)
* You can access the resources (memory, I/O, CPU) you need ~~to~~ | Kernel mode.
* Trap is system call that shifts ~~from~~ from user to kernel.
* Each device has a driver ∫ which contains the software.
* It is not password protected but it ~~has~~ is privileged instructions.
↳ perform operations that require direct access to hardware or other resources.

Date : _____

such as setting up memory mappings or accessing I/O devices.
- It only runs in kernel mode.
- Protects the hardware from being modified by the user.

Define with eg: Multicore, multiprocessor, multiprogramming, multitask, multithreading.

Multicore

Processor is an IC that has two or more processor cores attached for enhanced performance and reduced power consumption. Each core (processing unit) can independently execute tasks.

Eg: Modern intel or Intel core i7 has 2 to 8 cores.

Multi processor

consists of multiple processors (CPUs) that work together. This setup allows parallel processing and is often found in servers.

Eg: UNIX, LINUX, Solar.

Multiprogramming (context switching)

is a method where multiple programs are loaded into memory and executed concurrently. The OS manages the switching b/w programs, optimizing CPU usage.

* Define time sharing

Date :

Eg: Desktop operating systems like Windows, macOS

Multitask (time sharing)
refers to ability of OS to execute multiple tasks or processes at the same time.
This can involve either time-slicing (rapid switching b/w tasks) or running tasks in parallel on multiple cores.

→ a single program is broken into multiple tasks.

Eg: Windows.

Multi threading

allows multiple threads (smaller units of process) to run concurrently within a single process.

eg: Web browser (multiple tabs)

Development stages of OS

① Batch OS → no direct interaction b/w user & system
② Multiprogramming OS
③ Time sharing OS
④ Real time OS

① Inefficient ; has to use operator for making any changes. (traditional OS) ; CPU time gets wasted
② Allows multiple programs to run simultaneously on single processor
Context switching ; if one prog gets stuck, CPU switches to next program.
∴ CPU time is not wasted.

priority scheduling, fair sched. CFS

eg: macOS, Windows.

(3) Time sharing OS → divided interaction b/w user and system.
↳ use CPU scheduling
↳ shares CPU time among n users.
↳ round robin scheduling algo

eg: LINUX.

(4) Real-time OS
↳ complete application within the time constraints
↳ ensures critical tasks are completed on time.

eg: airline traffic control system, robots

(*)(*) Functions of OS.

1. Process management
2. memory management
3. Storage management
4. Protection and Security

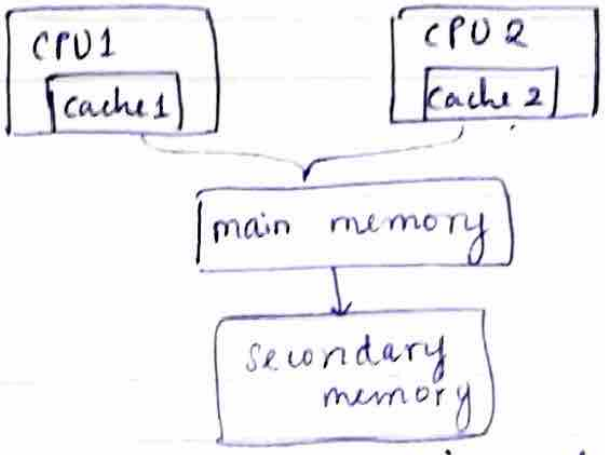
1 Process management
* creating and deleting both user and system processes
ctrl+z * suspending and resuming processes.
* providing mechanism for process synchronization.
* " process communication
* " deadlock handling

resources → space is available for the process in S.M.

two CPU. Banking one IP both CPU [share]

* synchronization → wait for an event to complete before starting next event.

eg: Banking system: deposit and withdraw.



Balance is global data.

Scenario: CPU1 handling deposit
CPU2 handling withdraw

CPU1	CPU2
Balance: 5000	5000
① Deposit : 1000	② withdraw : 500
	③ store 4500 in balance

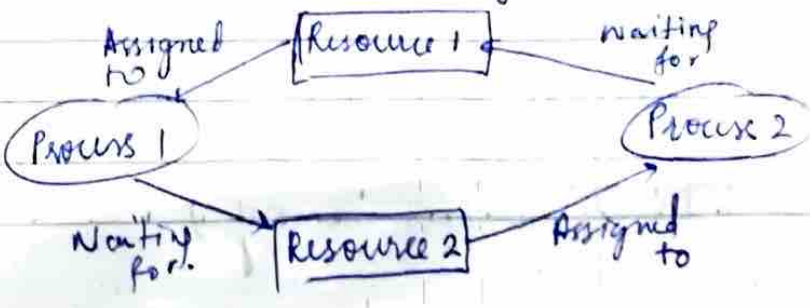
Fetch, Add

* If there is no synchronization, the answer will be 6000 not 5500. i.e. data is inconsistent
* balance is changed in cache only not m m or s.m
* Atomic operation — completing all steps of a process.

eg for exam: a = a+10
Fetch a
a+10
store value in a.

⇒ Deadlock.

Limited resources and many processes.



* When resource is not available for a process, CPU puts that process in wait/lock state.
* Both the process cannot continue since the resources are busy, so they end up in deadlock state.

Date :

2. Memory management
   * Keeping track of which parts of memory are currently being used and by whom (log/table maintained)
   * Deciding which processes and data to move into and out of memory. (virtual mem. and demand paging)
   * Allocating and deallocating memory space as needed. (malloc ~~used~~, calloc, free).

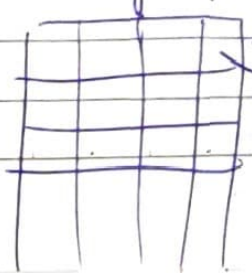
stderr.no

note: OS is loaded / available in intial address.

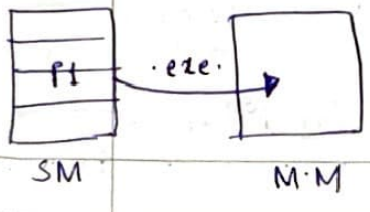
For eg: A program needs 10 kb
Space in M.M is 2kb.
So the prog. is divided into segments and loaded one by one in M.M based on ∂ instruction needed to be executed.
this technique → ~~virtual memory~~ demand paging
(virtual memory)
(on demand the part of prog. ~~the~~ req. is loaded).

3. File system management
   * Creating and deleting files and directories (mkdir, vi, gedit, rm, etc.)
   * Primitives to manipulate files and directories (rm, cat, rmdir, etc.)
   * Mapping files onto secondary storage.
   * Backup files onto stable (non-volatile) storage media

Files are stored in sectors which contains blocks each of approx 512 bytes.



(secondary memory).



Addres in S·M and M·M is different
Memory management unit needs to identify the equivalent M·M address of S·M addres (Mapping algorithm).

searching (searching file in a directory)	seeking (seeking a data inside a file)

creating files
↳ partition into drives.

8) Differentiate b/w Protection and security of system.
eg: Password.

T1 1) Types of operating system.
↳ Windows, Linux, macOS

2) Functions of OS.
↳ process management, memory management, file system management, protection and security, storage management.

3) Kernel space - CPU executes OS/related tasks.
User space - CPU executes user related tasks.

4) Types of kernel
↳ monolithic, microkernel, hybrid, exokernel.

5) ~~Types~~ Diff. process and programs.

6) Distributed system
↳ collection of compiler prog. that utilize computational resources across multiple, separate computation nodes to achieve a common shared goal

Parallel system
↳ S/W process data simultaneously, and sharing the memory space.

7) Context switching, time sharing system, Real time, loader, linker, Scheduler.

OS services (provided by user and system itself) 18/10/24

1. User Interface
2. Program execution
3. File system manipulation
4. Error detection
5. Communication.
6. I/O operations
7. Resource allocation
8. Accounting
9. Protection & Security

1, 2, 3, 4, 5, 6 — User service.
7, 8, 9 — System services

1. Windows → mouse (graphical) } communicate b/w user
   Linux → commands }
2. Allows user to create program/process, (provide environment) on behalf of user does loading from memory.
3. create, modify, delete, search, seeking file.
5. system to system } (comm^{n} b/w programs & process)
   remote → within system.
   - (a) shared memory
   - (b) message passing (send and receive). } modes of comm^{n} of process.
4. Syntax error, illegal access of memory, H/W not working. (OS informs user). * no debugging by OS *
6. keyboard, monitor
7. Fair allocation of resource b/w users
8. Recording the actions during the session.
9. Protection
   - control the access to resources.
   - protect from internal attack.
   - allow only authorized user.

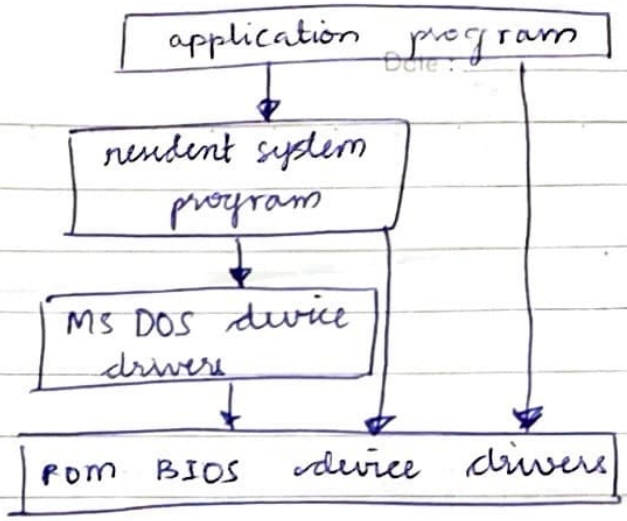
   eg: password.

   Security
   - defend from external attacks.

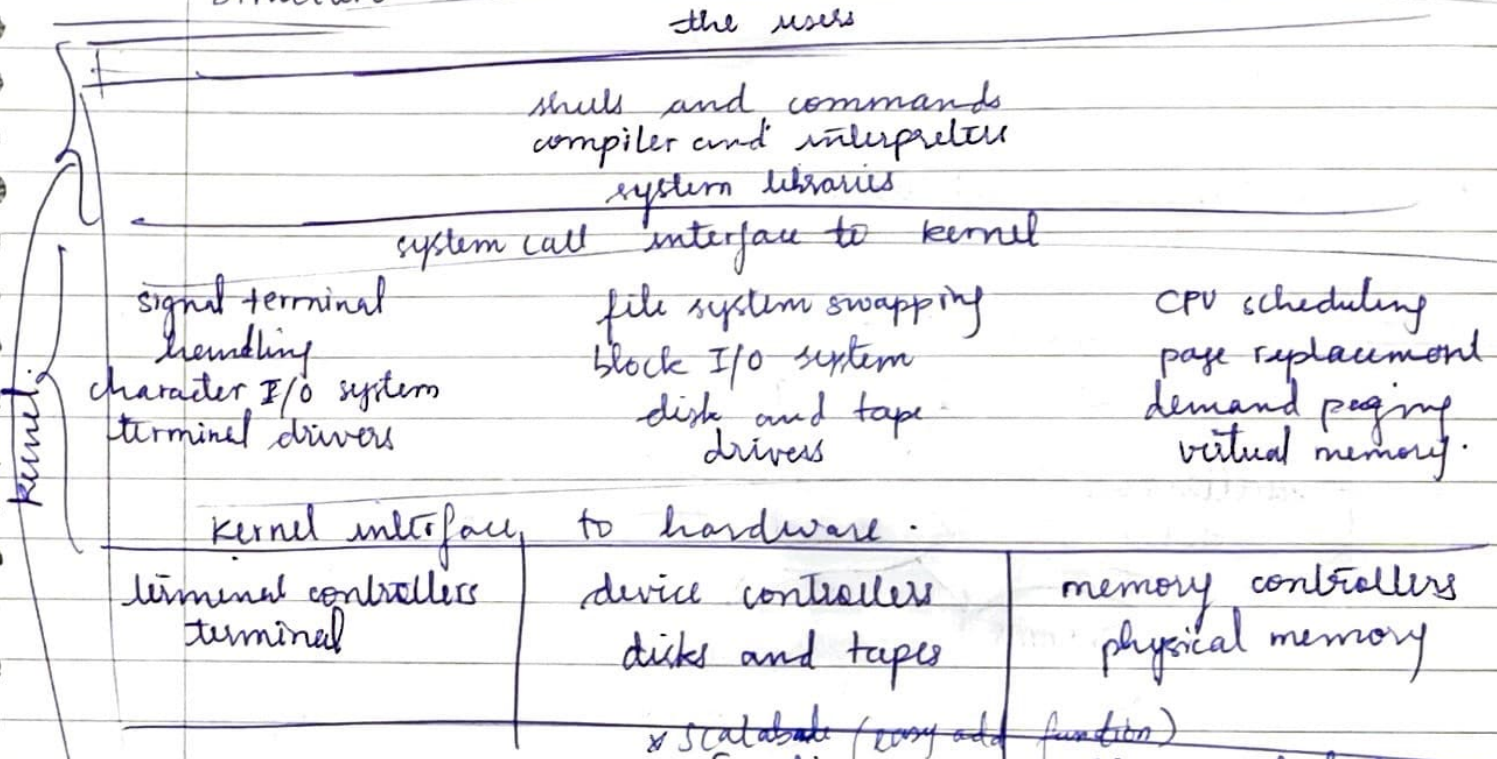
   eg: antivirus, encryption.

OS Structure (disk OS)

* app program has direct access to device driver
* MS DOS was initially used but now not used



UNIX Traditional / monolithic structure



* ~~Scalable~~ (easy add function)
* Functions are well separated
* Drivers are protected safely
* Better than MS DOS

* user mode access functions of OS through system call.

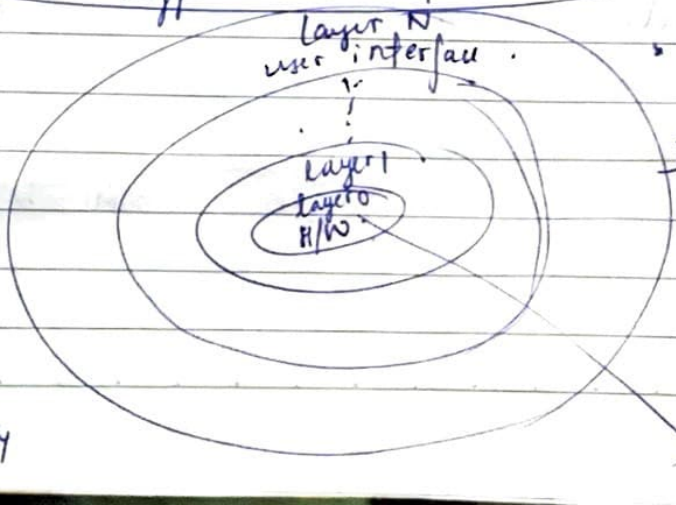
Limitation: difficult to implement

Layered approach

* each func is implemented as a layer.

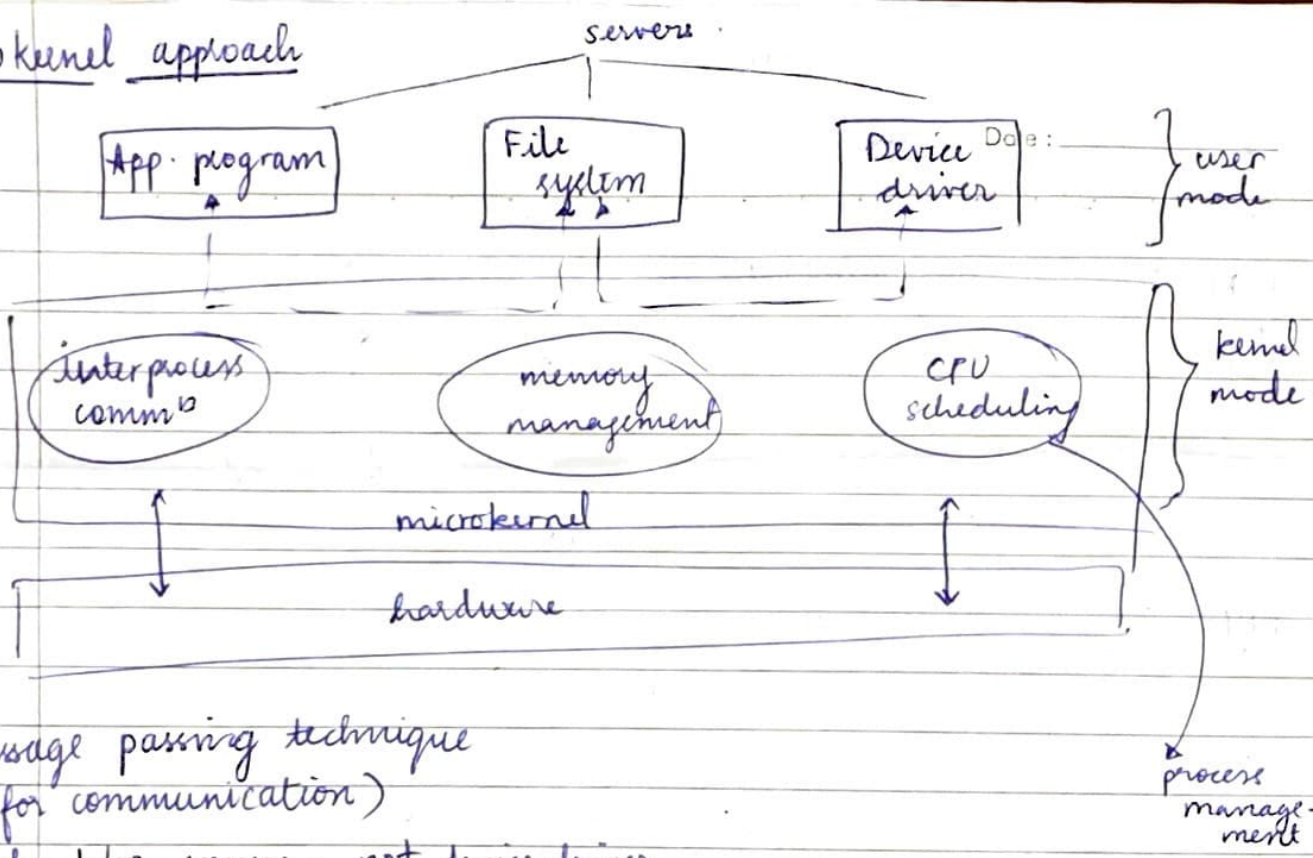
eg: under process management, each func is a separate layer (loader, CPU scheduling)

* layer provides functionality to above layer.



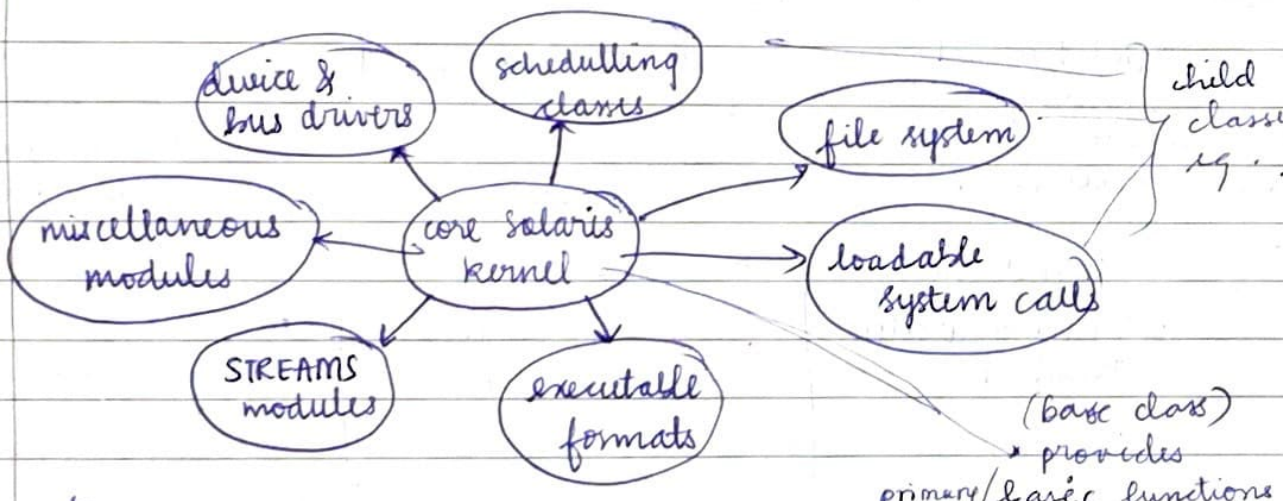
* easy to debug.
* one layer error will affect all above layers.

Microkernel approach



* message passing technique (for communication)
* only b/w servers ; not device driver and user.

Modules



* similar to inheritance.

* in previous types, if a new feature is added, the entire OS has to be restarted.
* in modules, if a new feature is added, it can be done dynamically (child class) while OS is running.
  eg: Ubuntu.
* loadable kernel modules (LKM)
  - dynamically a feature can be added/removed w/o disturbing the running OS.

Android OS

note: Def^{n} of system call with example.

(*)(*) Types of system calls (7 marks)

Date :

1. Process control
2. File management
3. Device
4. Information maintenance.
5. Communication.

(Remember few system calls under each).