- Computer System Organisation

Cs ops

Components : One / more CPU & device controllers connect thru a cmn bus

- CPU & device execute concurrently, competing for memory cycles
- I/O devices & CPU can execute concurrently
- Device Controller : each controller is in charge of a particular device type & local buffer.
- data movement : CPU moves data frm main memory to local buffers
- I/O : local buffer to controller

CPU → main mmy → local buffer → controller

Interrupt

The device controlls informs CPU that is finished its ops by sending Intrpt

Functions :-

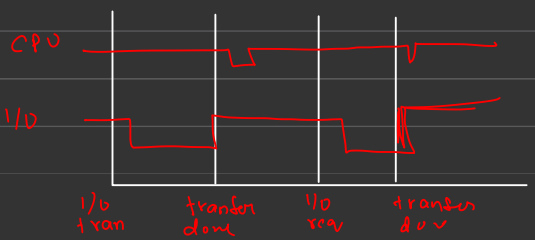
1. Control transfer : When Intrpt occurs, the system switches to intrpt service routine using addresses
2. Saving State : It saves the add of Intrpt Instruct^n
3. Trap / exception : Software gen Intrpt when error occurs
4. OS operats by responding to intrpts

Handling :-

- OS preserve the state of CPU by storing registors & pgm countr & determines type of Intr

Polling Vectored

- Seperate segments of code determine what action shld be taken



- I/O Structure

- Synchronous I/O (Blocking I/O) :-

  - I/O starts, control returns to user prog after completition of I/O, CPU waits for op to finish until next intrpt.
  - Only 1 I/O req at a time

- Asyn (Non-blocking)

  - After I/O Starts, control returns to user pgm w/o waiting for I/O completion
  - System call req OS to allow user to check I/O completion
  - device status table : into of each I/O
  - OS check table to see status

- Storage Structure

Main memory : large storge media
CPU can access directly
Random access
Volatile

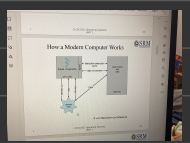
Sec Storge : large
Non-vol

Hard disks : disk controller determins the logical intraction

Solid state dks : Non-vol
Faster than hard disk

- Caching :-

Info in use is copied frm slowr to fastr storge tempo rorily
Smallr than Mmry storge
Main mry can be viewed as cache for sec memory



- DMAS

- Devce Controllr tranfrs blocks of data from buffer sforag to main mry w/o CPU
- Used for high speed I/O tranfer of data
- 1 Intrpt generated per block

- Computer System architecture
  - deals with multiprocessors

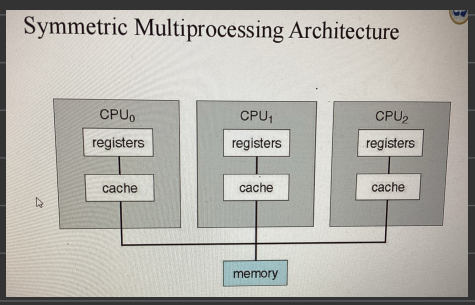
Multiprocessor systems are growing in use, they're also known as parallel systems, tightly coupled systems

- Advantages :-

1. Economy of scale
2. Increased reliability

- Two types :-

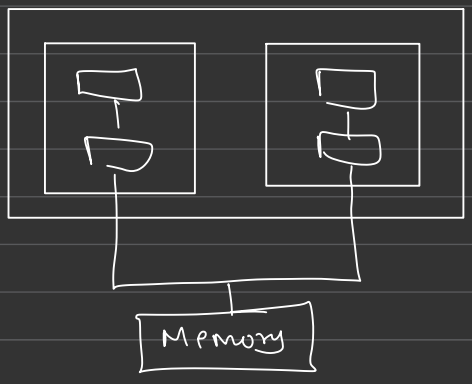
1. Asymmetric - each processor assigned a specific task
2. Symm - each proc perform all tasks



examples :-

1. A dual-core design

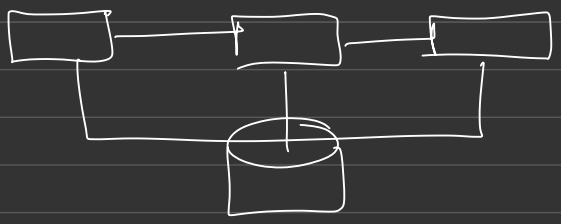
Multichip & multi-core
Single chasis containing multiple seperate systems



2. Clustered Systems

- Multiple systems working together
- Provides storage via storage-area-network
- Works in both Sym & Asym clustering depending upon nodes

eg High performance (HPC)
distributed lock manager (DLM)



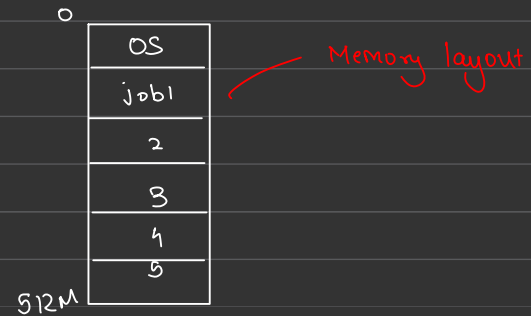
- **Operating-System Structure**

- **Multiprogramming (Batch system)**

1. Single user cannot keep CPU & I/O devices busy at all times
2. Multiprog organises jobs so CPU executes one at a time
3. A subset of total jobs is kept in memory
4. One is selected via job scheduling
5. When one job waits, OS goes to another job

- **Timesharing (Multitasking)**

1. CPU switches jobs so users can interact a running job
2. each user has atleast one program executing in memory
3. response time < 1 sec
4. CPU scheduling : several jobs run at same time
5. Swapping : If process doesn't fit in memory, swapping moves in & out
6. Virtual Memory : allows executing not completely in memory



- Operating - Structure operations
  - Interrupt driven (h/w & s/w)

- ## Process Management

  - A process is a program in execution
  - Program is passive while process is an active entity
  - Process needs resources to complete its task like CPU, I/O file, data initialisation, etc
  - Single-threaded process has one program counter
  - Multi-threaded process has one " " per thread
  - Typically a system has many processes running concurrently on one or more CPUs

OS responsibilities with process mgmt

1. Creating & deleting both user & system processs
2. Suspending & resuming
3. Provide mechanism for process synchronisat"
   comm
   for deadlock handling

- ## Memory management

1. To execute a program, its instructions must be in the memory
2. All / part of the data that is needed by the program must be in memory
3. MM determines what is memory & when
4. Optimizes CPU utilization

MM activities

1. Keeping track which part of memory are currently being used by whom
2. deciding which process & data moves into memory
3. Allocating & deallocating memory.

- **Storage managment**

OS provides uniform, logical view of into storage
- Abstracts physical props to local storage unit
- Each med. is controlled by devica (disk, drive, etc)

File system mgmt
- Files organised to dirs
- Has access control on most systems

OS activities

1. Creating & del dirs
2. Manipulate files & dirs
3. Map files
4. Backup files

Mass-storage mgmt

- It stores data that does not fit in main memory of the data that has been stored for a long period of time
- proper mgmt is of central imp

- OS activities

1. free-space mgmt
2. Storage alloc
3. disk scheduling

Migration of data "A" from disk to register

hard disk → Main memory → cache → hardware register

- I/O System

Hides peculiarities of hardware devices from the user

MM of I/O

1. Buffering - Storing data temporarily while it is being transferred
2. Caching - Storing parts of data in faster storage for performance
3. Spooling - Ovelap output of 1 job w input of other jobs

It has a general device-drive interface & has drivers for specific devices

- Protection & Security
  - Mech for controlling access of users to resources
  - defence system against internal & external attacks
    - denial of service
    - worms, viruses
    - identity theft, theft of service

User ID - includes name & user no, they determine access control of user

Group ID - Allows set of user to access control

privilage escalation - its gain of more access rights temporarily

• Computer Environments

1. Traditional

- Stand alone gen. purpose machines
- portals provide web access to internal system
- Network comps (thin clients) are like web terminals
- Mobile comps interconnect via wireless network
- every home system use firewalls to protect comps from Internet attacks

2. Mobile

- Handheld Smartphones, tabs, etc
- OS features like gps & gyroscope
- New apps like aug reality
- wireless / cellular data for connectivity
- leaders like apple iOS & Google Android

3. Distributed

- Collection of hetergeneous systems networked together
- Network is a comm. path
  1. (LAN) Local area network
  2. (WAN) wide
  3. (MAN) metro-politan
  4. (PAN) Personal
- comm. allows system to xchange mssgs
- Ilulu of single system

4. Client Server

└ dumb terminals supplanted by smart PCs

- Many system / servers respon to req gen by clients
- Compute-server system: provides interface to client req servic
- File server system: " " " to store files

5. Peer - to - Peer

- Another model of distributed system
- P2P does not distinguish clients & servers
- All nodes conn by peers
- Client for one act server for another
- Broadcast req for servic & respond for servic via discovery protocol
- eg Gnutella, Voice Over IP (VoIP)

6. Virtualisation

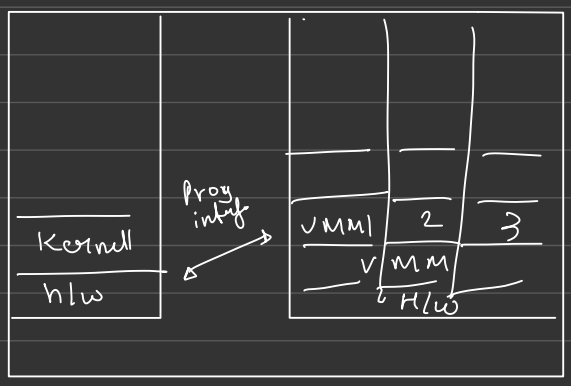
- Allows OS to run apps with other OS's
- Emulation :-
  - used when source CPU type diff frm target type
  - slowest method
  - Interprepation : Executes code line-by-line if not complied by CPU

Use cases :-

1. Running Multiple OSes : Mac OS running windows as a guest
2. App dev : develop & test apps for diff OSes

- VMM : Virtual Machine Manager

1. It creates & manages virtual machines
2. works b/w hardwar & OS
3. Runs natively if they are the host
   eg VMware, Xen Server



7. Cloud Computing

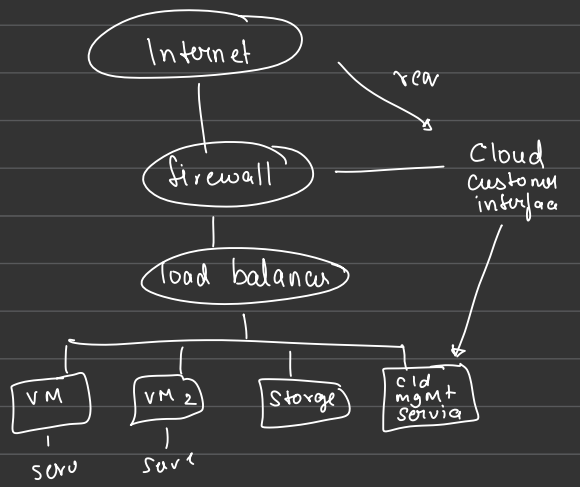
- Delivers computing resources as a service accross a network
- It uses virtualisation to provide resources
  eg Amazon EC2

Types :-

1. Public Cloud : Accessible over the internet by anyone who pays
   eg AWS , Google Cloud
2. Private : Managed by a company
3. Hybrid : Combine pub & pri
4. SaaS (Software as a Service) : Apps available over the internet
5. Paas (Platform ") : provides plat for dev & deploy apps
6. Iaas (Infra ") : provides server & storage over a internet

CC Components · tradional OSes
VMM
CC tools

Security : ensure connectivity with measures like firewalls
Load balane : distribute network traffic



8. Real-time embedded System

- Rtes are prevalent form of comps
- special purpose , ltd purpose os real time os
- Real time OS has well-defined fixed time constraints
- Processing must be done within time constraint
- Some Real time system perform task without os

- Operating System Services

1. User Interface (UI) :-

Two -type : i) CLI - Command - line
ii) GUI - Graphics user Interface

2. Program Execution :-

The system should load a prog into memory, run it and end execut^n, also handle error if any

3. I/O ops : Interacts I/o device reqs

4. File system manipulation :-

Progs need to read, write, create, delete, search files & directories also should list file info & handle permission mgmt

5. Communication :-

Thru shared memory, processes may exchnge info on same or diff comp over a network

6. Error detection :-

OS is constantly aware of possible errors may occur in CPU, hard drive or I/O device, when detected, OS takes appropriate action to ensure smooth & consistent computing

7. Resource Allocation :-

when multiple users & jobs are running simultaneously, resources are allocated to each of them

8. Accounting :-

OS keeps tracks of resources usage. Records how much CPU time, memory & other resources each user or prog uses

9. Protect^n & Security :- Pg-6

- **User OS Interface**

1. **CLI - Commnd line Interface**
   - Allows direct cmd entry
   - Fetches cmd frm user & executes it
   - Smtimes implemented in kernel or system progs
   - " cmds are built in
   - " multiple flavored implemented shells

2. **GUI -**
   - User frndly metaphor interfac
   - mouse, keyb, monitr
   - various mouse btns over obj cause
   - various actions when clicked
   - Invented at Xerox PARC

eg : Ms Windows - GUI / CUI cmd shell
Apple mac OS - " Aqua GUI, UNIX Kernel

- **System Calls**

- Prog^{mg} interface to the service of OS
- Written in high lvl lang (C/C++)
- Accessed by progms via high level App Prog Interfa (API)

- A no. is associated w each system call
- A system-call interface comm w OS Kernel to invoke SC
- OS kernel executes the prog and returns result to user
- details of OS interface is hidden from programmer by API

Passing Parameter :-

- In registors - most simple one
- Paras stores in block / table is passed as para in a register eg linux, Solaris
- Paras pushed on the stack by programmer popped off the stack by os

- Types of System calls

  - Process control
    create, terminate, end, abort, load, execute, wait event, signal event, wait for time, allocate & free memory

  - File mgmt
    Create file, delete

  - Device mgmt

- System Programs

Provide a convinient envo ~~for~~ prog develop & execution

File mgmt : create, delete, copy

• OS design and Implementation

- D & I is 'not solvable' but some methods have worked
- diff OS can have diff internal strucs
- Start by setting goals & specs, Consider H/w & type of system
- Goals :-

User : OS shld be ease to use, learn, reliable, safe & fast
System : " " to design, imple, maintain
shld be flexible, error-free, reliable, efficient

- Seperation :-

Policy : What will be done ?
Mechanism : How to do it ?

Seperating policy (decisions) from mechanism (methods) provides flexi to change policies

eg. Handle timers.

- Implementation

1. language used

Early OSes : Assembly
Mid : System prog^g lang like Algol / PL1
Modern : C & C++

2. Mix of langs

Assembly lang for low lvl OSes
C for main part of OSes
C, C++, scripting langs for system progs like PERL, Pythn

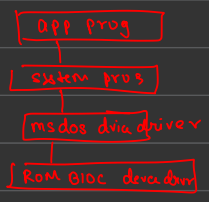
3. Speed

High-lvl langs easy to tranfer to diff h/w but slower
Emulation : Allow OS to run h/w its not designed for

• Operating System Structure

Various ways to organise an OS :-

Simple Struc : MS-DOS
More complesc : UNIX
layered Stru : Abstraction
Microkernel : Mach

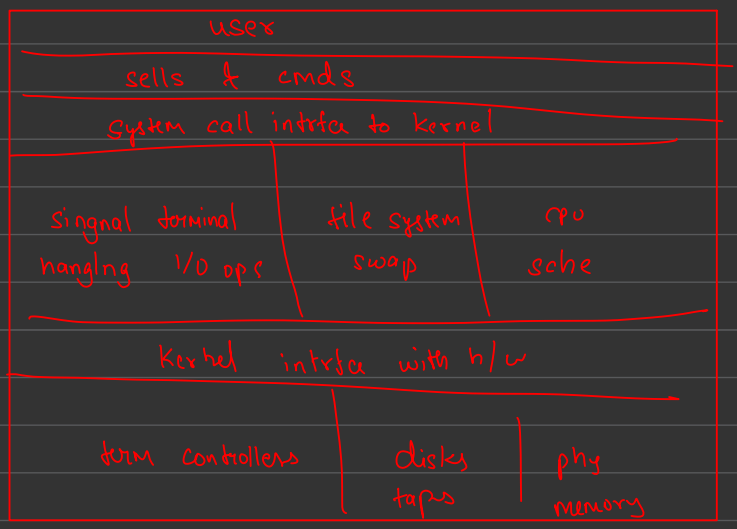


• Simple Struc - MS DOS

- designed to small & functional
- not divided into clear modules
- lacks well defined of interfaces

• Non - simple Struc - UNIX

- ltd by h/w
- 2 parts : System progs - Prog running on top of OS
  Kernel - Core part of OS, CPU scheduling, MM, etc

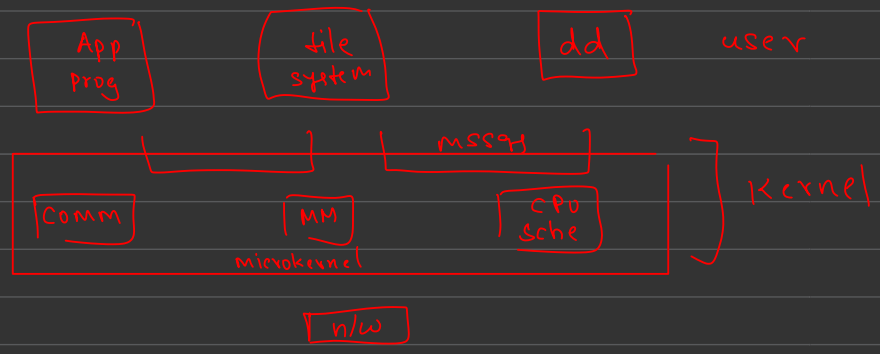


• layered approach :-

- OS div into layers
- each layer uses servica of lyr below it
- Botton - h/w, Top - user

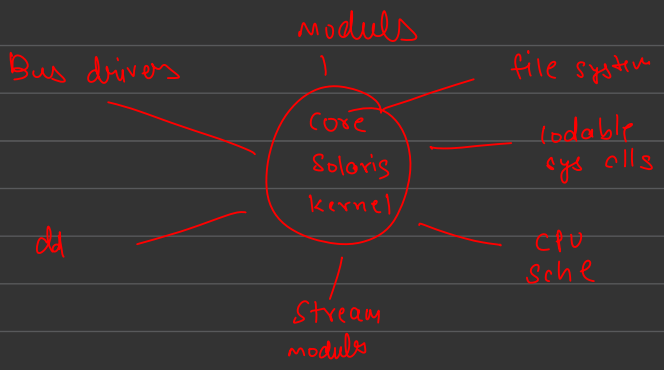
4. Microkernel Structure

- Moves many functions frm kenl to usr spce
- eg Mach (used in MAC OS)
- uses mssg passing for comm b/w modules
- **benefits** : portable, reliable & secure
- **drawbacks** : performance overhead due to comm gap



5. Modules

- Modrn OSes use loadable kernel modules
- OOP approach with seprte comp
- flexible than layers



6. Hybrid System

- Combine mul struc for better performance, security & usability

Egs :-

- linus & Solaris : Monolithic with modular compnts
- Windows : Mostly Monol with some Microk features
- Mac OS : Hybrid with layered struc & Microk featurs

- Mobile OS egs

1. iOS
   - based on mac OS
   - ARM arch, not intel
   - Features :-
     - Cocoa touch for app dev
     - Media service
     - Core service for cc
     - Core OS on Mac OS Kernl

2. Android
   - developed by Open handset alliance (google)
   - open sourc
   - Features :-
     - process, memory mgmt
     - power mgmt
     - run time environ
     - App developed in Java
     - libs for web brow, DBMS, etc

- OS debugging
  - Its finding & fixes errors / bugs
  - log files : contain error info.
  - Core dump : when app fails, it gen core dump file & structure
  - Crash dumps : when OS fails, it gens a crash dump file

- **Operating System Generation**

Purpose : OSes are designed to run on various machines for specific comp

SYSGEN : Gather info abt specific h/w config to build system - tuned setup

Efficiency : A custom kernel can be more eff than a gen on

- **System Boot**

Steps :-

1. Initialization : When system powered on, it executes from a fixed memmory locate
2. Firmware : Initial boot code is stored in firmware ROM
3. Bootstrap loader : A small piece of code, stored in EPROM, locates kernel & loads in memory
4. 2-step boot : ROM code loads boot block, then loads BS ldr
5. Commn BS : GRUB is a commn BL that allows users to select kernel
6. Kernel load : Once kernel loaded, the system begins running