Short Notes

Compiler

→ It is a software that convert program written in a high level language to a low level language
→ Moreover, it is necessary to perform this step to make the program executable
→ It analysis the entire code to check for any errors & then it run the executes
→ Generates intermediate Obj. code

Assembler

→ Translates low-level language to machine code or executable code that a CPU can understand
→ Consist of mnemonic codes that correspond directly to machine instructions
→ Generates obj. file that contains machine code

Interperters

→ Executes instruction written in a high-level language directly
→ Instead of compiling the entire program into an executable file, an interpreter process the program line by line
→ This allows for immediate execution but can result in slower performance.
→ Ex Python

Linkers

→ Tool used to combine various obj. files generated by a compiler into a single executable program.

→ it can also Handles memory allocation for different programe segment

→ Linker resolvers the refereance such as function call ensuring that function, variables, & other are properly linked across different obj file.

Loaders

→ Part od OS that Loads an executable file into memory for execution

→ Once Program is linked & an executable is created, loader takes over the memory allocation, loading the Program code

→ It also sets up the executable enviroment.

→ It ensures the program is ready to run by the CPU

Macro/Pre Prossesors

→ Processing od predefined patterns or instructions in a Program before actual compilation or assembly

→ Allows Programmers to define ~~reuable~~ reusable code spipret or commands, which can be expanded or replaced by macro Prossesor.

→ Simplifies complex tasks, improve code readability & reduces the chance od errors.

```
#ifdef macro_name
  S1
  S2
  :
  SN
#endif
```

Compiler	Interpreter
→ A compiler is a program which converts the entire src code of a Programming language into executable machine code for a CPU	→ Interpreter takes a source program & runs it run by line, translating each line as it comes to it
→ Compiler take large amt of time to analyze the entire code but the overall execution time of the program is comparatively faster	→ Interpreter takes less amt of time to analyze the code but the overall execution time of the program is slower
→ Debugging is comparatively harder as the error can be present anywhere	→ Debugging is easier as it cont. translates the code until an error is met
→ Generates intermediate Obj. code	→ No intermediate Obj. code
→ Ex. C, C++, Java	→ Ex. Python, Perl

Evolution of OS

1) Main Frame System

→ Batch system

- Early computer's were physically enormous machines run from a console.
- User did not interact with the computer system
  → User prepared the job which consists of the program, data & control info.
  → The job was in form of punch cards
- The O/p consist of
  → Result of the program, as well as a dump of the final memory
  → Register contents for debugging
- To speed up Processing
  → Operators batched together jobs with similar needs &
  → Ran them throughout the computer as a group

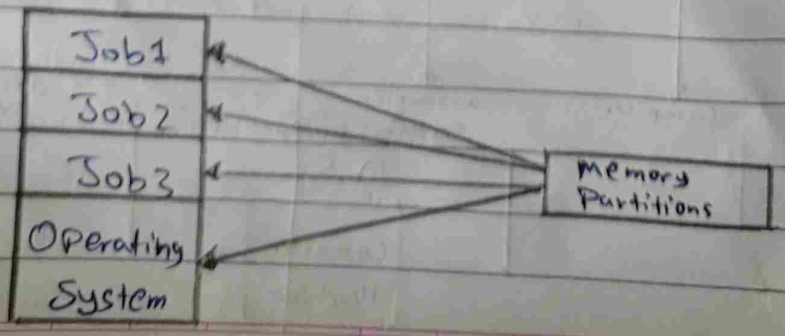
→ Multiprogramming

→ ↑ CPU utilization by organizing jobs so that the CPU always has one to execute
→ The idea is to keep multiple jobs in the main memory.
→ Advantage

- High CPU utilization ⇒ ↓ waiting time
- Useful in scenario where load on CPU is higher

→ Disadvantage

- Process scheduling is difficult
- Main memory management is required.



→ Multitasking

→ It is logical extension of multiProgramming

→ It is the ability of an OS to execute more than one task simultaneously on a CPU

→ These tasks share common resources

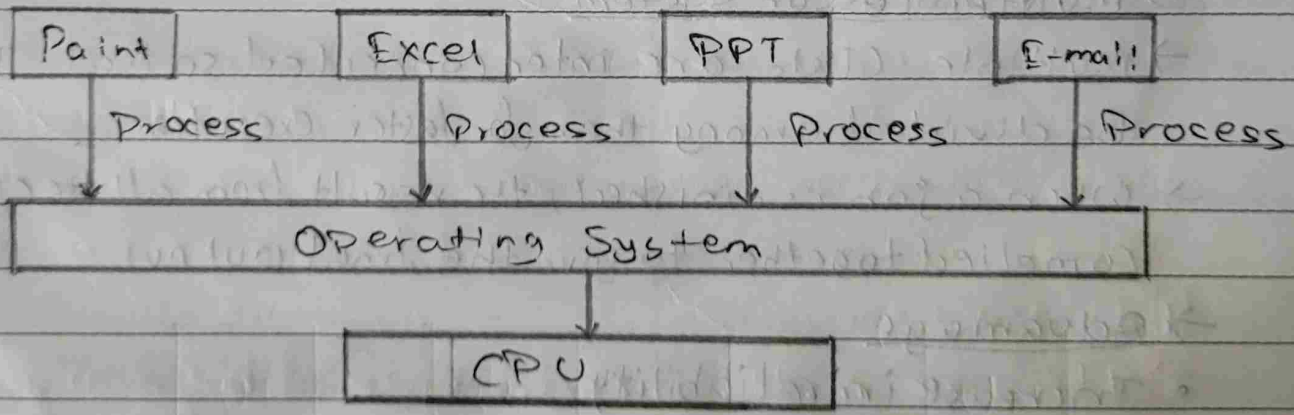
→ The CPU executes multiple jobs by switch among them using a small quatum time

→ Advantage

- The system can be used to handle multiple interactive task
- It has high CPU Utilization
- ↓ waiting time
- Useful in scenarios where current load is high

→ Disadvantage

- Process scheduling is difficult
- Main memory management is required
- Problem like memory fragmentation may occur



Multiprogramming	Multitasking / Timesharing
→ Concept of context switching	→ Concept of context switching & timesharing
→ The OS simply switches to & executes another Job when current job needs to wait	→ The processor is typically used in time sharing mode. Switching happens when allowed time expires or job needs to wait
→ ↑ CPU utilization by organizing Jobs	→ ↑ CPU utilization it also ↑ responsiveness
→ Idea is to reduce the CPU idle time for as long as possible	→ Idea is to further extend the CPU utilization concept by ↑ responsive Time sharing
→ It uses Job scheduling algo.	→ It uses Time sharing mechanism
→ Execution of Process takes more time	→ Execution of process takes less time

2) Desktop System

→ Multiprocessor System

→ Multiple CPUs are interconnected so that Job can be divided among them for faster execution

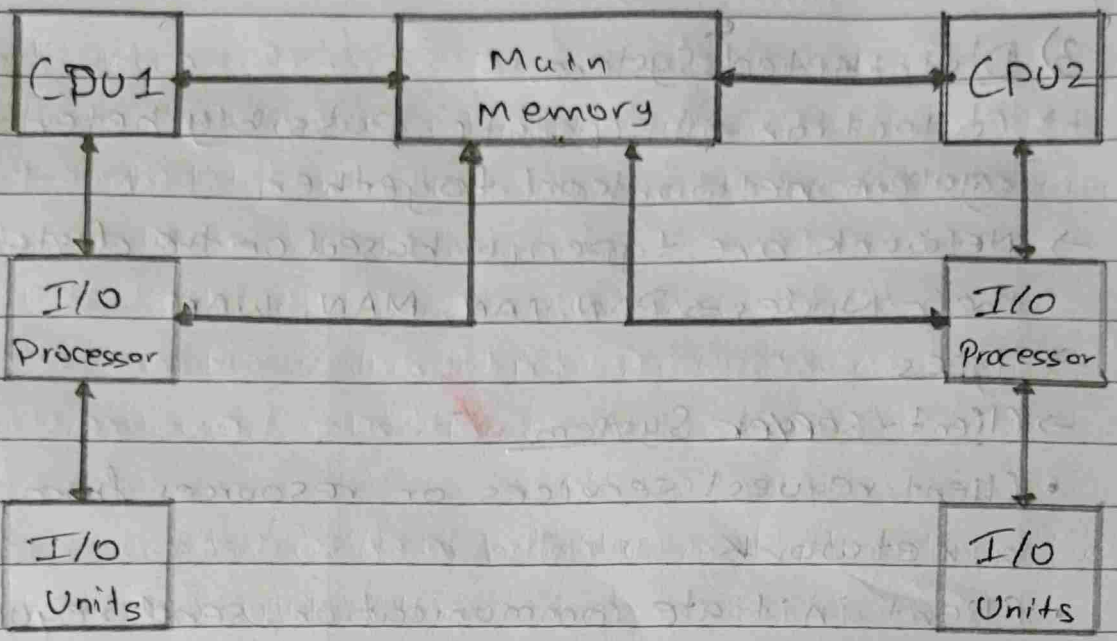
→ When a job is finished, the result from all the CPUs are complied together to give the final output

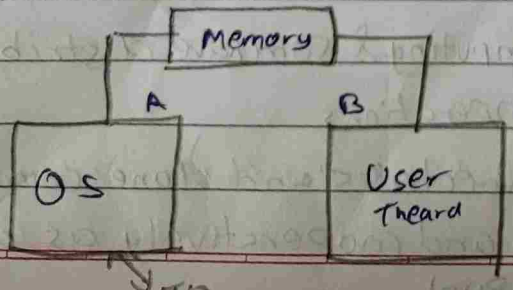
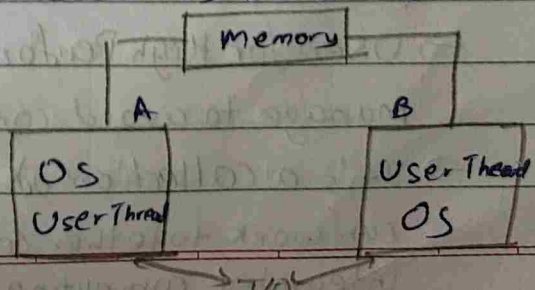
→ Advantage

- Increase in reliability
- Increased throughout
- The economy of Scale

→ Disadvantage

- It is more complex & sophisticated as it takes care of multiple CPUs at the same time.



Asymmetric	Symmetric
→ The processors are not treated equally	→ The Processors are treated equally
→ Tasks of the OS are done by master Processor	→ Task of the OS are done by individual processor
→ No communication between Processors as they are controlled by master processor	→ All processors communicate with another processor by a shared memory
→ Process are master slave	→ Process is taken from the ready Queue
→ Cheaper	→ Expensive
→ Easier to design	→ Complex to design
→ Suitable for homo or hetro - geneous cores	→ It is suitable for hetrogeneous Cores
→ All processors can exhibit different architecture	→ The architecture of each Processor is the same
→ 	→ 

3) Distributed System

→ Collection of seperate, possibly heterogenous, system networked together.

→ Network are typecast based on the distance ~~the~~ bet. their Nodes ⇒ PAN, LAN, MAN, WAN

→ Types

→ Client-server System

- Client request services or resources from servers over a network
- Client initiate communication, send requests, and handle user interfaces, while servers listen to requests, performs tasks & manage resources.
- This models allows for scalable resource utilization, efficient sharing, modular development, centralized control & fault tolerance.

→ Peer-to-Peer (P2P) System

- Interconnected nodes directly communicate & collaborate without centralized control
- Each node can act as both a client & a server, sharing resources
- The model facilitates distributed data sharing, content distribution & computing tasks

4) Clustered System

→ It's like multiprocessor systems, but multiple system working together usually sharing storage via SAN

→ Provides a high-availability service which survives failures

→ Used for High Performance Computing & somehave distributed lock manage to avoid conflicting operations

→ It's a collection of interconnected stand alone computers which can work together collectively and cooperatively as a single integrated computing resource pool

→ Deployed to improve speed and/or reliability over that provided by a single computer, while being much more cost effective than single computer of comparable speed or reliability.

→ Types

→ Asymmetric clustering

- One m/c is in hot standby mode while the other is running applications
- The hot standby host does nothing but monitor the active server
- If that server fails, the hot standby host becomes the active server
- The standby node is powered on & running some monitoring programs to communicate heart beat signals to check the status of the primary not, but is not actively running other useful workloads.

→ Symmetric Clustering

- Two or more hosts are running applications & they are monitoring each other
- More efficient, as it uses all the available hardware.

5) Real Time System

→ Used when a rigid time requirements have been placed on the processing

→ Functions correctly only if it returns the correct result within it's time constraints.

→ Types

→ Hard RTS

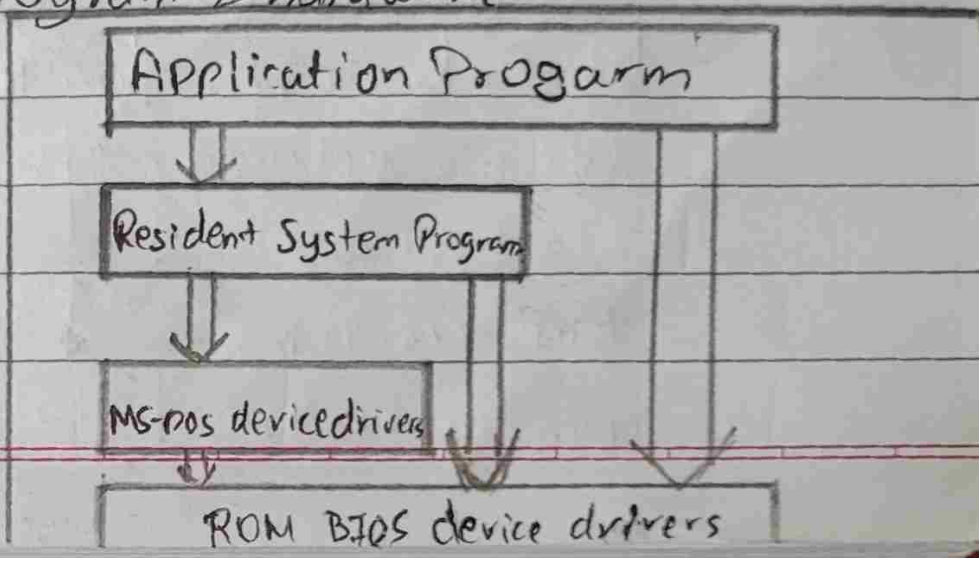
→ Soft RTS

Hard RTS	Soft RTS
→ Size of data file is Small to medium	→ Size of data file is large
→ Response time is in millisec	→ Response time is Higher
→ Peak load performance Should be predictable	→ Peak load can be Tolerated
→ System safety is critical	→ System safety is not critical
→ Very restrictive	→ Less restrictive
→ Incase of error the computation is rolled back	→ Incase of error the computation is rolled back to prev checkpoint
→ Guarantees response within specific deadline	→ Does not Guarantee response within specific deadline
→ Focus	

Operating System Structure

→ Simple / Monolithic Structure

- Such operating system do not have well defined structures & are small, simple & limited, Ex MS-DOS
- Ex. MS-DOS, application Program are able to access basic I/O routines.
- Advantage
  → It delivers better application performance because of few interfaces between application Program & hardware
  → It is easy for kernel developers.
- Disadvantage
  → Structure is complicated
  → It does not enforce data hiding in the OS



→ Micro-Kernel Structure

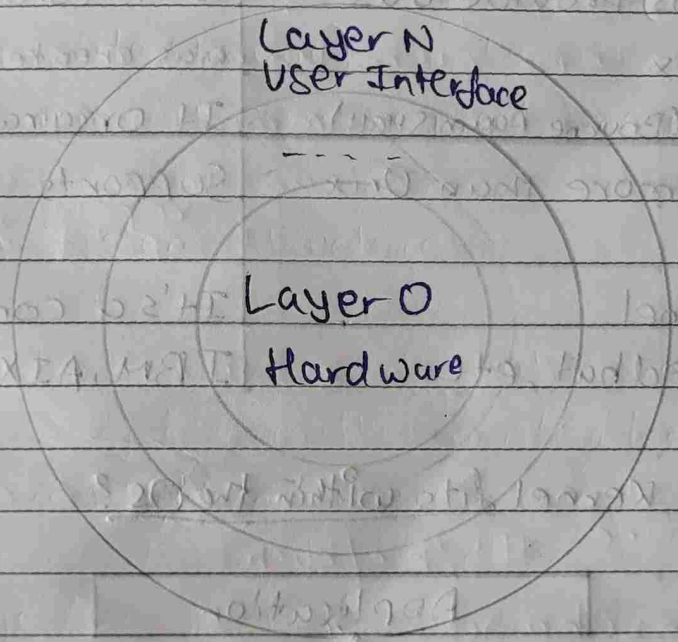
- This structure designs the OS by removing all non-essential components from the Kernel & implementing them as system & user programs.
- Advantage
  → It makes the OS portable to various platforms
  → These can be tested effectively
- Disadvantage
  → Increased level of inter module communication degrades system performance

Parameters	Micro Kernel	Monolithic Kernel
1) Address Space	User services & Kernel services are kept seperately	Both user services & Kernel services are kept together
2) Design & Implementation	Complex to design	Easy to design & implement
3) Size	Smaller in size	Larger in size
4) Functionality	Easier to add new ones	Difficult to add new ones
5) Coding	More code is required	Less code is required
6) Falivre	Falivre of one comp. does not effect the working of micro kernel	Falivre of one comp. leads to Falivre of entire system
7) Communication	To implement IPC messaging queues are used	To implement IPC messaging signals & sockets are used
8) Debugging	Simple	Difficult.

Date
Page

→ Layered Structure ~~Architecture~~

- ~~An OS can~~ In this structure, the OS is broken into no of layers.
- It is design in such a way that each layer uses the functionality of a layer at a lower level.
- This simplifies the Debugging Process
- Advantage
  - → Layers makes it easier to enhance the OS
  - → Very easy to perform debugging & system verification
- Disadvantage
  - → Application perdormance is degraded
  - → It requires careful planning for designing the layers.



→ Hybrid Structure

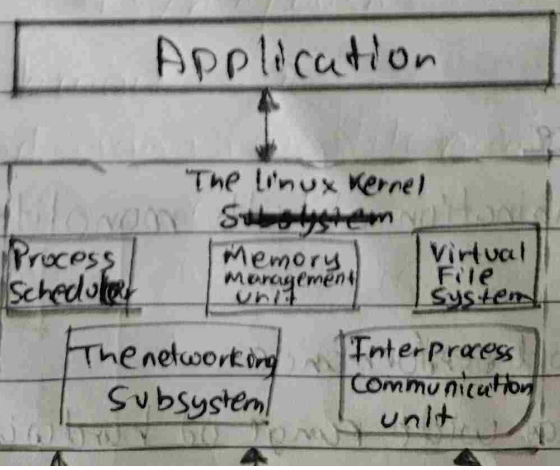
- It is just a combination of both monolithic & micro-kernel structure
- Advantage
  - → It offers good perdormance
  - → It supports a wide range of hardware & applications
  - → It provides better isolation & security

- Disadvantage
  → If ↑ overall complexity of the system
  → The layer of communication between components ↑ time complexity & ↓ performance.

Parameters	Linux	Unix
1) Definition	It is an open source OS which is freely available to everyone	It is an OS which can be used by it's copyrighters
2) Users	Anyone can use it	mainly used for servers, work & mainframes
3) Development	It's developed by sharing & collaboration of codes	was developed by AT&T Lab various commercial vendors & non Pro
4) GUI	It is command based but some distros provide GUI based Linux	Initially command based OS, later common Desktop enviro was created
) Interface	Uses Bash (Bourne Again Shell)	It originally used Bourne She
) File stem Support	Supports more than Unix	Supports Less than Linux
) OS	It's a Kernel	It's a complete package OS
Example	Ubuntu, Redhat, etc	IBM, AIX, HP-UX, etc

Where the Kernel fits within the OS?

Note:
A Kernel is a core part of OS. It's the bridge bet. Software applications & hardware of the



System Booting

The working

→ The CPU initialize itself after the power in the computer is first turned on, This done by triggering a series of clock ticks generated by system clocks

→ After this, the CPU Looks for the system's ROM BIOS to obtain the first instruction in the start-up program. It instructs ~~POST~~ the system to run POST (Power on self Test) in the memory address

→ POST

- First checks the Bios chip & then the CMOS RAM. If there is no battery failure detected ~~by~~ by POST, then it cont. to ~~check~~ initialize the CPU
- It also checks the hardware devices
- After all of the checks if all are working then the BIOS start Bout Strap Program

Master Boot Record

- Located at the very beginning on the hard disk & it has critical details for starting up
- It holds information about GRUB

Bios (Integrity check) → MBR (Executes GRUB) → GRUB (Executes Kernel)

Kernal (Executes init.d) → init (Initiate Runlevel) → Runlevels (Execute Runlevel)

- Last step of Booting Process
- With Runlevels of OS:

0 - System Halt

1 - Single user Mode

2 - Multiuser without NSF

3 - Full multiuser Mode (default)

4 - Unused

5 - Full multiuser mode with network & X11

6 - Reboot

Failure during Boot

→ If the computer cannot Boot we will get the Boot failure error
→ If could be caused by computer not passing POST or Hard-ware failure
→ You may hear a beep if it is a hardware failure
→ You may see a error message or blue screen ~~due~~ if it is due to corruption or not found.

Type of Boot

1) Cold => means to startup a computer that is turned off
2) Warm => means to restarting a computer once on (Ctrl+Alt+Del)

Feature	Multiprocessor	Multicore
1) Definition	A system with multiple CPUs that can process programs simultaneously	A single processor containing multiple core (Independent processing unit)
2) Execution	Can Run multiple programs faster	Can Run a single program faster
3) Reliability	More reliable. If one CPU fails others cont. working	Less reliable. If one core fails in can affect the entire processor
4) Memory Structure	Can use shared or distributed memory systems	Cores share memory within the same chip
5) Traffic	Higher traffic due to multiple CPU's accessing shared resources	Less traffic as multiple cores are integrated on a single chip.
6) Config	Requires complex config	Easier to configure
7) Parallel Processing	Achieved by using Multiple Processors	Achieved by using multiple cores within one processor
8) Energy Efficiency	Consumes more energy	Consumes less energy
9) Use Case	Suitable for large systems needing to run multiple applications simultaneously	Ideal for general purpose computing with parallel tasks in a single program.

System Calls

→ Allows programs to request services from the OS like accessing hardware, managing files, or controlling processes.

→ How does it work

- Request: Program makes a request to system for resources like memory or I/O
- System call Execution: made using predefined instructions
- Switch to kernel mode: to execute the requested service
- Operation Performed: Performs the Requested task
- Then Returns control

→ Service Provided

- Process Management
- Memory Management
- File Management
- Device Management
- Communication

→ Examole

Process	Windows	UNIX
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	Fork() Exit() Wait()
File Manipulation	CreateFile() ReadFile() WriteFile()	Open() Read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() Read() write()
Information Management	GetCurrentProcessID() SetTimer() Sleep()	Getpid() Alarm() Sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	Pipe() Shmget() Mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorgroup()	Chmod() Umask() Chown()

→ Advantage

- Access to hardware resources
- Memory management
- Process Management
- Controls access to privileged resource
- Provides consistent interface across different Platform

→ Disadvantage

- Context switching can slow down performance
- Security Risks
- Error Handling Complexity
- Compatibility issues.

MODY

1] File management ~~System~~

→ Structure Terms

i) Field
- basic element of data
- Contains a single value
- Characterized by it's length & data type
- Fixed & variable length

ii) Record
- Collection of related fields that can be treated as a unit by some application program
- One field of the record is the key
- Depending on design, records may be of fixed length or variable length.

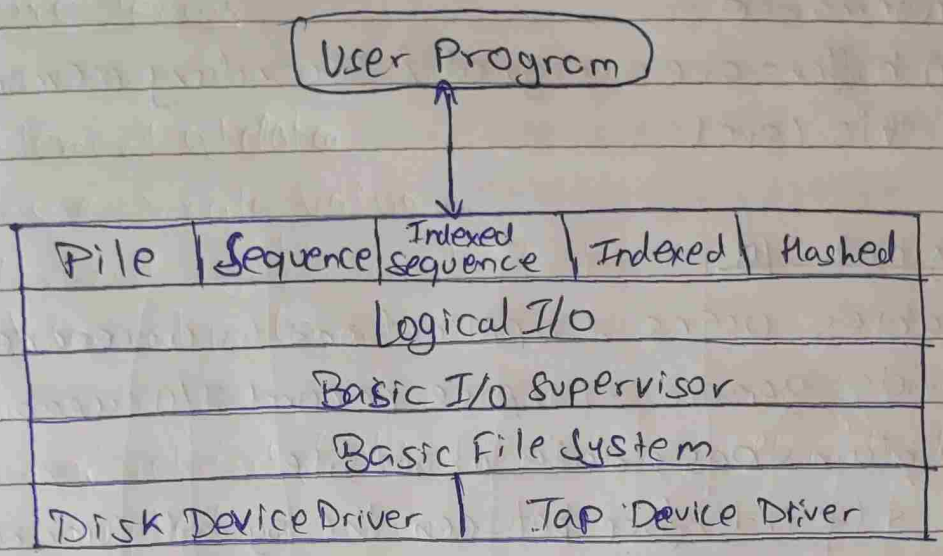
iii) File
- Collection of similar records
- treated as a single entity by user & application.
- access control restrictions usually apply at the file level.

iv) Database
- Consists of one or more types of file
- Collection of related data
- Relationships among elements of data are explicit
- Designed for use by a no° of different applications

→ Objectives

- Provides services to users & applications in the use of files
- Typically, the only way that a user or application may access files is through the file management system.
- Meet the data management needs of the user
- Guarantee that the data in the file are valid
- Optimize performance
- Provide I/O support for a variety of storage device types
- Minimize the potential for lost or destroyed data
- Provide a standardized set of I/O interface routines to user Processes
- Provide I/O support for multiple users in the case of multiple user systems.

→ Architecture



Device Drivers

→ Lowest level, communicates directly with peripheral devices.
→ Responsible for starting I/O operations on a device
→ Processes the completion of an I/O request
→ Considered to be part of the operating system.

Basic-File system

→ Also referred to as the physical I/O level
→ Primary interface with the enviroment outside the computer system.
→ Deals with blocks of data that are exchanged with disk or other mass storage devices.
→ Considered to be part of the operating System.

Basic
~~Logical~~ I/O Supervisor

→ Responsible for all file I/O initiation & termination.
→ Control structures that deal with devices I/O, scheduling & file status are maintained.
→ Selects the device on which I/O is to be performed.

→ Concerned with scheduling disk & tape accesses to enhance performance
→ I/O buffers are assigned & secondary memory is allocated at this level

Logical I/O

→ Enables users & applications to access records
→ Provides general-purpose record I/O capability
→ Maintains basic data about file
→ It is the interface between the logical commands issued by a program & the physical details required by the disk

File Organization

→ Is the logical structure of records as determined by the way in which they are accessed.
→ Criteria's
- Short access time
- Ease of update
- Economy of Storage
- Simple maintenance
- reliability

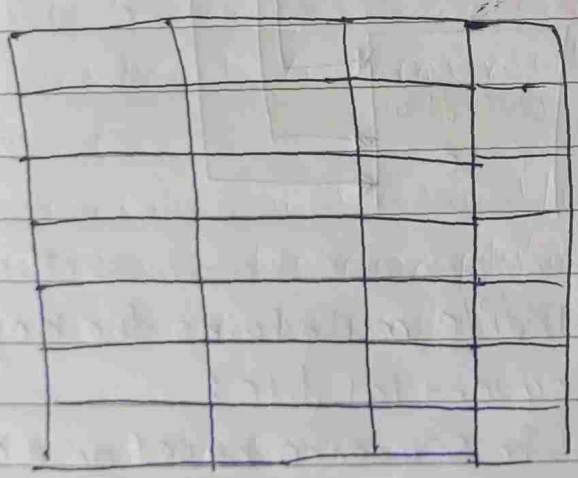
→ Five common file organizations:-

1) The pile

→ The least complicated form of file organization
→ Data are collected in the order they arrive
→ Purpose is simply to accumulate the mass of data and save it
→ Each record consists of one burst of data

→ Record may have different fields, or similar fields in different orders.
→ because there is no structure to the pile file, record access is by exhaustive search
→ That is, if we wish to find a record that contains a particular field with a particular value
→ It is necessary to examine each record in the pile until the desired record is found or the entire file has been searched

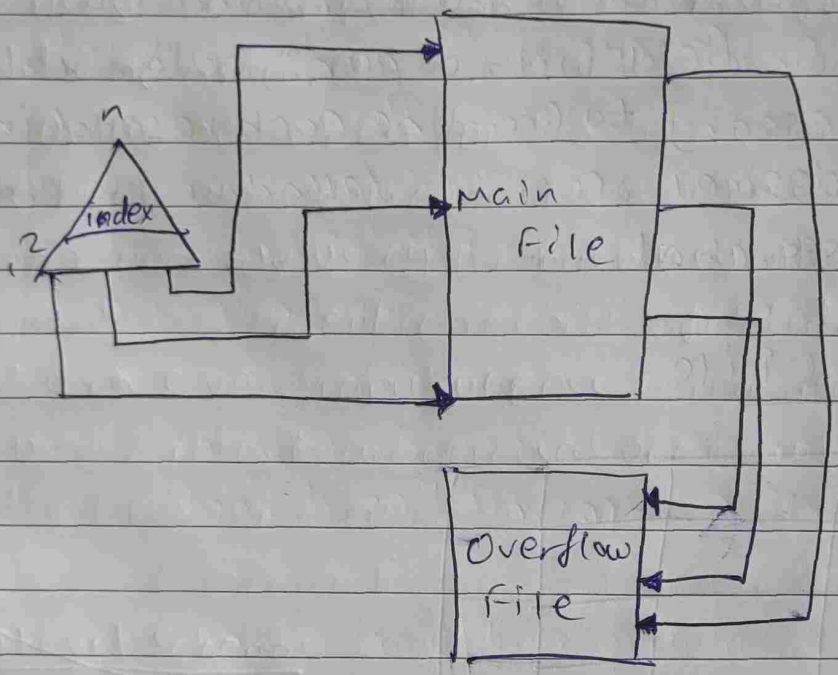
2) Sequential File



→ Most common form of file structure
→ A fixed format is used the records
→ All records are of the same length
→ Records consist of same no. of fixed-length fields in a particular order.
→ One particular field, usually the first field in each record, is referred to as the key field.
(uniquely identifies the record.)
→ Typically used in batch application if they involve the processing of all the records

→ Easily stored on tape as well as disk.

3) Indexed Sequential File



→ The indexed sequential file maintains the key characteristic of the sequential file:
- Records are organized in sequence based on a key field

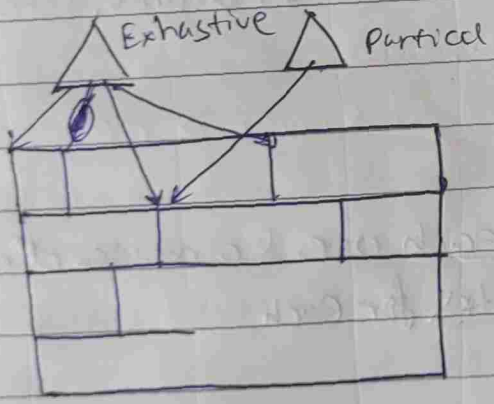
→ Two features are added:
- Adds an index to the file to support random access
- An overflow file.

→ Each records in the index file consists two fields:
- A key field, which is the same as the key field in the main file
- And a pointer into the main file.

→ To find a specific field, the index is searched to find the highest key value that is equal to or precedes the desired key value.

→ The search continues in the main file at the location indicated by the pointer
→ Additions to the file are handled in the following manner:
- Each record in the main file contains an additional field not visible to the application, which is a pointer to the overflow file.
- When a new record is to be inserted into the file, it is added to the overflow file.

4) Indexed File



→ Records are accessed only through their indexes
→ The result is that there is now no restriction on the placement records as long as pointer in at least one index refers to that record.
→ Variable length records can be employed.
→ Two types of indexes:
- Exhaustive index contains one entry for every record in the main file
- Partial index contains entries to records where the field of interest exists

→ Some records will not contain all fields.
→ Used where timeliness is critical & where data are rarely processed exhaustively.

→ When a new record is added to the main file, all of the index file must be updated.

5) Direct or Hashed File

→ It uses the computation of the hash function on some fields of a record.

→ The output of the hash function defines the position of the disc block where the records will be stored.

→ When requested using the hash key columns, an address is generated, and the entire record is fetched.

→ Used where very rapid access is required, fixed-length record are used & records are accessed one at a time.

One Level Schema

→ There is one directory for each user & a master directory

→ master directory has an entry for each

File directories

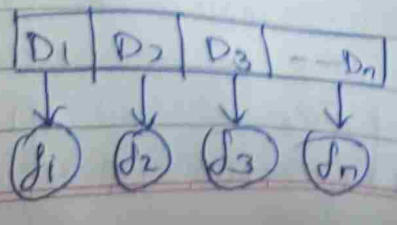
→ A directory is a container that is used to contain folders & files.

→ It organizes files & folders in a hierarchical manner.

→ Types

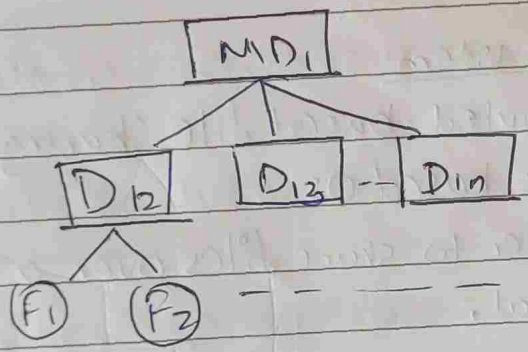
1) Single level

- It is the simplest directory structure
- In it all files are contained in the same directory which makes it easy to support & understand



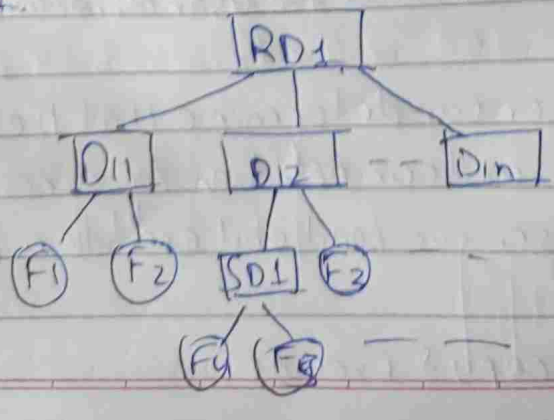
2) Two-level

- There is a one directory for each user & a mastery directory
- master directory has an entry for each user directory providing address & access control info.
- Each user directory is simple list of the files of that user
- Names must be unique only within the collection of files of a single user
- File system can easily enforce access restriction on directories



3) Tree Structure

- User can create files & subdirectories.
- The directory structure resembles a real trees upside down, where the root directory is at the peak
- The root contains all the directories for each user.
- The users can create subdirectories & even store files in their directory
- A user does not have access to the root directory data & cannot modify it.



File Sharing

→ It denotes how information & files are shared between different users, computers or devices over the network.

→ Ways to achieve File sharing

1) Server message block
   → Is like a network based file sharing protocol mainly used in windows OS
   → It allows our computer to share files ~~folders~~ on a network

2) Network File System
   → It is a distributed based file sharing Protocol mainly used in Linux / Unix based OS.
   → It allows computer to share files over a network as if they were based on local.

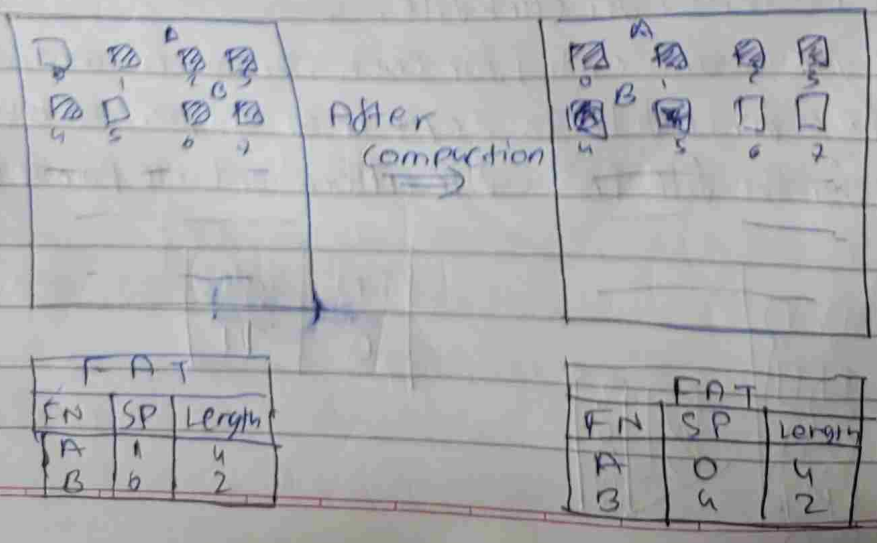
3) File transfer Protocol

4) Cloud based file sharing
   → It involves the famous way of using online services like google drives, etc.
   → Any user can store files over these cloud services & they can share that with others & providing access from many users

→ Access Rights
1) None : Not allowed to read the user directory that includes the file
2) Knowledge : the user can determine that the file exists & who its owner is and can petition for access rights
3) Execution : The user can load and execute a program but cannot copy
4) Reading : Can read, copy & Execute

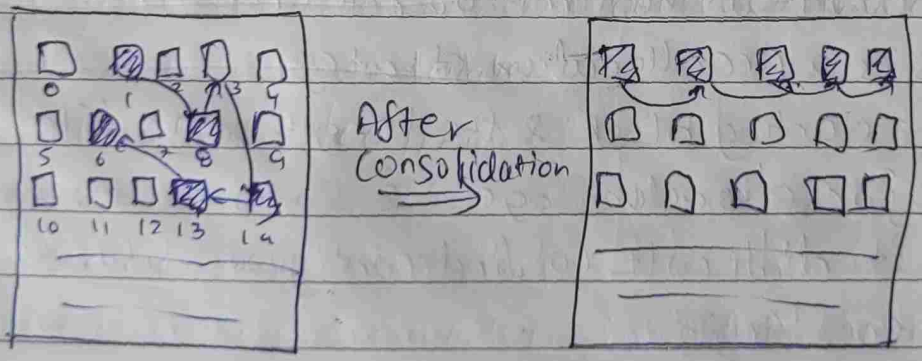
5) Appending : Can add to the file, but not modify or delete
6) Updating : Can add, modify & delete
7) Changing Protection : Change access rights granted to others
8) Deletion : Can delete

~~File allocation~~
Secondary Storage ~~Stage~~ management
1) File allocation
a) Contiguous Allocation
→ In this schema each files occupies a contigoous set of blocks on the disk.
→ A single contiguous set of blocks is allocated to a file at the time of file at the time of file creation
→ Thus this is a preallocation strategy.
→ Shows the starting block & the length of the file
→ But a major disadvantage is External Fragmentation, making it difficult to find contiguous blocks of space of sufficent length
→ The sol^n is from time to time it will be necessary to perform a compaction algorithm to free up additional space on the disk.



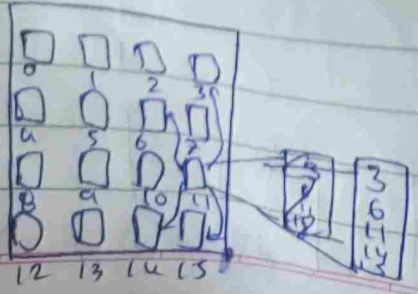
b) Chained Allocation

→ Each block contains a pointer to the next block in the chain
→ It needs just a single entry for each file
→ Showing the starting block & the length of the file
→ Although preallocation is possible, it is more common to simply allocate blocks as needed
→ There is no external fragmentation to worry about because only one block at a time is needed
→ Better for sequential files.
→ One disadvantage of chaining, as described so far, is that there is no accomodation of principle of locality
→ To overcome this systems periodically consolidate files



c) Indexed Allocation

→ In this case the file allocation table contains a seperate one-level Index for each file.
→ The index has one entry for each portion allocation to the file
→ File index for a file is kept in a seperate block, and the entry for the file in the file allocation table points to that block



FN	Index block
B	14

2) Free space mangement.

→ It is a critical aspect of OS as it involves managing the available Storage space.

→ It uses various Techniques:

a) Linked allocation

- Each file is represented by a linked list of disk blocks.
- When file is created, the OS finds enough free space on the disk & links the blocks of file to form chain.

b) Contiguous Allocation

- Allocates contiguous disk blocks for each file
- The entire file is stored in a single contiguous block of space
- Efficient but prone to external fragmentation.

c) Indexed Allocation

- Maintains an index block for each file
- The index contains address for all the disk blocks belonging to the file.

d) File Allocation Table

- It is a table that tracks file locations by maintaining pointers to blocks in sequential list.
- When a file is created, the OS updates the Table

e) Volume shadow copy

- Used to create backup copies of files or entire volumes.
- When a file is modified, the OS creates a shadow copy of the file & stores it in a separate location.

→ Methods to track free disk

a) Bit vector

- This method uses a vector containing one bit for each block on the disk
- Each entry of 0 corresponds to a free block & 1 as a used block
- Requires very little space, as no complex disk allocation table is maintained.

- Works well with all types of file allocation methods.

b) Chaining Free portion

- In this the free blocks on the ~~block~~ disk are chained together by using a pointer & length value.
- Negligible space overhead because there is no need for a disk allocation table, merely a pointer to the beginning of the chain and the length of the first portion.
- If allocation is block by block, simply choose the free block at the head of the chain & adjust the first pointer or length value.
- If allocation is variable length, the header from the portion are fetched one at a time to determine the next suitable free portion in the chain.
- ~~Negligible space overhead because there is no need for a disk allocation table~~
- ~~If~~ leads to fragmentation

c) Indexing

- For efficiency, the index should be on basis of variable size portions rather than blocks
- This approach provides efficient support for all of the file allocation methods

d) Free block list

- Each block is assigned a no° sequentially
- Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block no°
- As a free block list is too large if is stored in the disk rather than in main memory
- 2 effective techniques for storing small part of the list in main memory :

i) Stack
- When a new block is allocated, it is popped from the top of the stack which is in the main memory
- Similarly, when a block is deallocated, it is pushed onto the stack.
- There only has to be transfer between disk & main memory, when it's either full or empty

ii) Queue
- A block is allocated by taking first entry from the head of the queue
- Deallocated by adding it to the end of the tail of the queue
- There only has to be transfer between disk & main memory, when either head becomes empty or tail is full

Unix File Management

1) Ordinary Files
- Used to store your info, such as some text you have written or an image you have drawn.
- Always located within / under a directory file
- Do not contain other files

2) Directories
- Stores both special & ordinary files
- Used to organize groups of files
- All files are descendants of the root directory.

3) Special Files
- Used to represent a real physical device such as printer, tape drive or terminal
- Used for I/O operations
- They appear in a file system just like an ordinary file or directory

4) Pipes

- Allows linking of commands using pipes
- It acts as a temporary file which only exists to hold data from one command until it is read by another.
- Provides a one way flow of data
- The output or result of first command sequence is used as the input to the second command sequence.

5) Sockets

- It is a special file which allows for advance interprocess communication
- It is used in a client-server application framework.

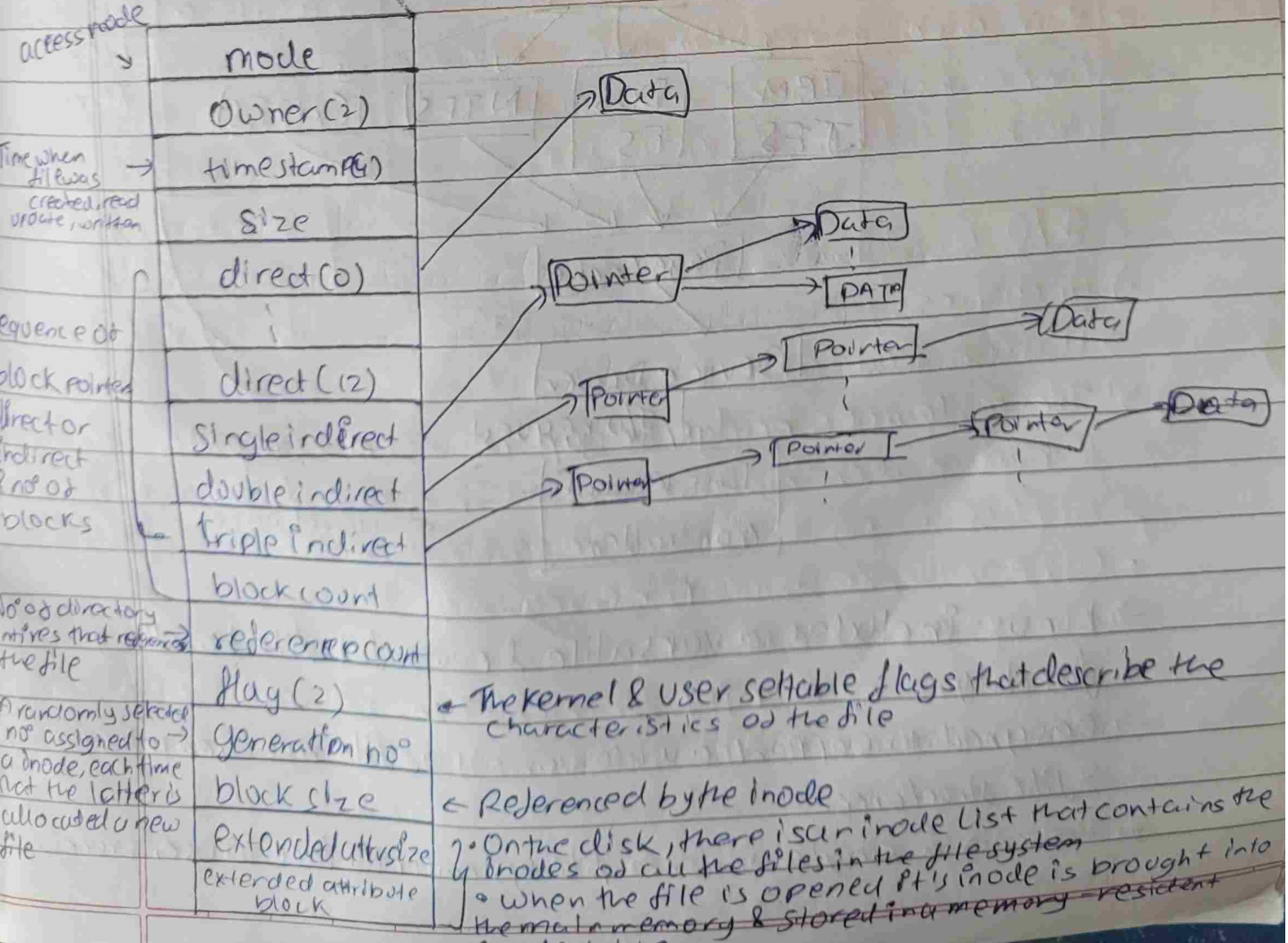
6) Symbolic Link

- It is used for referencing some other file of a file system.
- Also known as soft link
- It contains a text form of the path to the file it references
- To end user, symbolic link will appear to have its own name, but when you try reading or writing data to ~~files~~ this file, it will instead reference these operations to the file it point to.
- If we delete the soft link itself, the data file would still be there. If we delete the src file or move it to a different location, symbolic file will not function properly.

→ Inodes

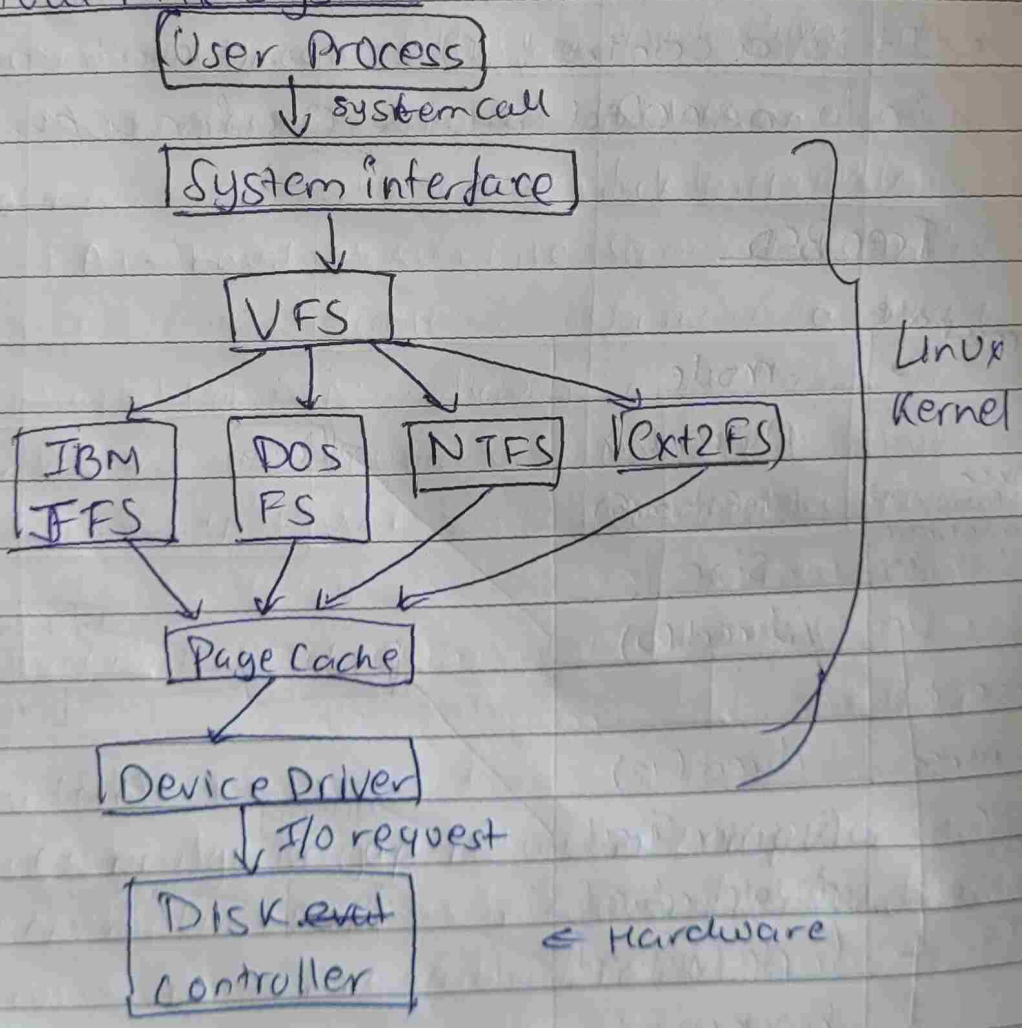
- Modern UNIX OS support multiple file system.
- but map all of these into a uniform underlying system for supporting file system and allocating disk space to files
- It is a data structure in the Unix-style file system that describes a filesystem object.
- Each inode stores the attributes & disk block locations of the object's data.
- It is a control structure that contains the key info. needed by the OS for a particular file.

FreeBSD



→ FreeBSD allows administrator to assign a list of Unix user IDs and group to a file
→ Any no° of users & groups can be associated with a file, each ~~three~~ with 3 protection bits (read, write, execute).
→ It includes an additional protection bit that indicates whether the file has an external Access control List.

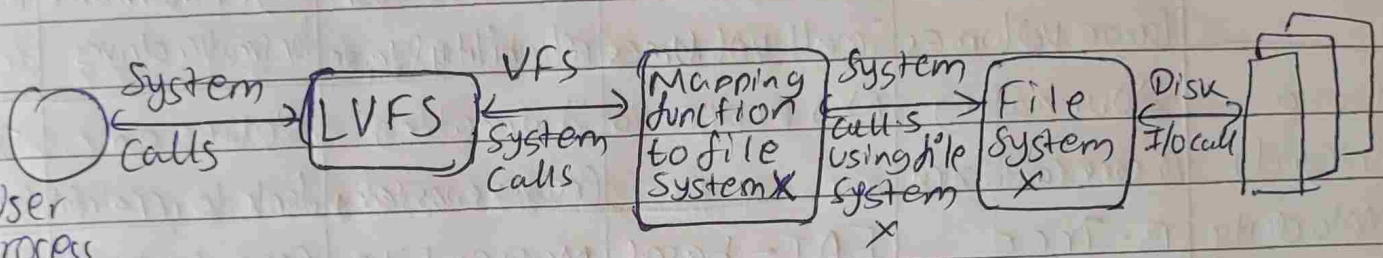
Linux Virtual File System



→ Linux includes a versatile & powerful file handling facility
→ Designed to support a wide variety of file management systems & file structures.
→ Presents a single uniform file system interface to user processes

→ Defines a common file model that is capable of representing any conceivable file system's general feature & behavior.
→ Assumes files are objects that share basic properties regardless of the target file system or the underlying processor hardware.

Role of VFS within the Kernel



→ A user process issues a file system call using VFS file schema
→ The VFS converts this into an internal C to the kernel file system call that is passed to a mapping function for specific file system
→ The mapping function is simply a mapping of file system functional calls from one schema to another
→ Primary Object Types in VFS

- Superblock Object: represents a specific mounted file system
- Dentry Object: represents a specific directory entry
- Inode Object: " " " file
- File Object: " an open file association with a process.

Features	IBM JFS	DOS FS	NTFS	ext2 FS	ext4 FS
Supported OS	Linux	MS-DOS	Windows	Linux	Linux
File Vol size limit	4PB	2GB	16 TB	2TB	16TB
Journaling Support	Yes (metadata & optional data)	NO	Yes (metadata & optional data)	NO	Yes (metadata & optional data)
Cluster/ Block size	512 bytes to 4KB	Fixed to 512 bytes	512 bytes to 64 KB	1KB to 4KB	1KB to 64KB
Performance	Optimal for large volumes	Limited to Small volumes	Optimized for reliability & large Vol	High performance for smaller drives	Better performance for larger drives
Data Integrity	Journaling ensure consistency	NO protection	Journaling ensures consistency	Relies on fsck to repair data	Journaling ensures consistency
Metadata Storage	B-Tree based	FAT-based	Master File Table	Seperate blocks for metadata	Inodes & extent based metadata
Advantage	High scalability	Simple & Lightweight	Advanced security	Simple & efficient	Backward compatible
Disadvantage	Complex, higher Overhead	Limited Scalability	Requires Windows	Not suited for large disks	Higher overhead compared to ext2

I/O Management

1) Categories

→ Human Readable Devices

- These devices bridge the gap between computer system & human users by taking inputs in a user friendly format or displaying outputs in readable formate
- Eg: monitor, keybords, printer, etc

→ Machine Readable Devices

- These are primarily for interactions between the system & other hardware or electronics. They are integral to storing, retrieving or processing data

Eg: Disk drivers, controllers, etc

→ Communication Devices
- Focused on enabling remote communication, these devices often manage data transfer over networks or between systems
- eg: modems, NICs, etc

2) Differences in I/O Devices
→ Data Transfer Rate
- Fast Devices: such as SSD & network cards
- Slow Devices such as keyboard & printers
→ Application ~~Based~~ specific behavior
- File Storage such as hard disks or SSDs storing application file require file-management software to ensure files are stored & retrieved properly
- Virtual Memory such as Disks use paging to store memory page temporarily, required virtual memory hardware to manage page swaps efficiently
→ Complexity Control
- Simpler devices such as keyboard may only send signals on keypresses
- Complex devices such as robot arms require real-time processing & control for precise operations
→ Unit of Transfer
→ Data Representation
→ Error Handling

3) Design Issues
→ Efficiency
- I/O devices are typically slower than main memory & the CPU, which can cause bottlenecks in system performance
- To optimize efficiency, especially Disk & network I/O, OS often uses techniques like multiprogramming.

- Additionally, systems may use swapping

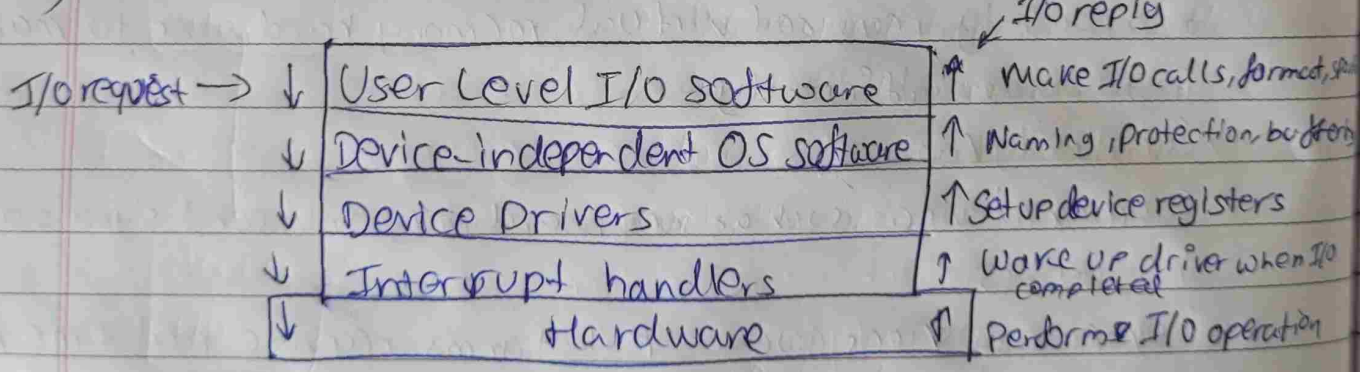
→ Generality & Uniformity

- The diversity of I/O devices often means that generality can compromise efficiency.
- To address this, lower-level routines are used to hide specific details of device operations.

→ Interrupt handling

- Efficient I/O management involves handling interrupts efficiently, as they notify the CPU about I/O events.
- Interrupt handlers are generally design to unblock drivers waiting for I/O completion & often operate in context of the currently running process to avoid costly context switches

4) I/O Software Layers



5) I/O Technique

→ Programmed I/O

- The processor issues an I/O command, on behalf of a process to an I/O module
- that process then busy waits for the operation to be completed before proceeding.

→ Interrupt driven I/O

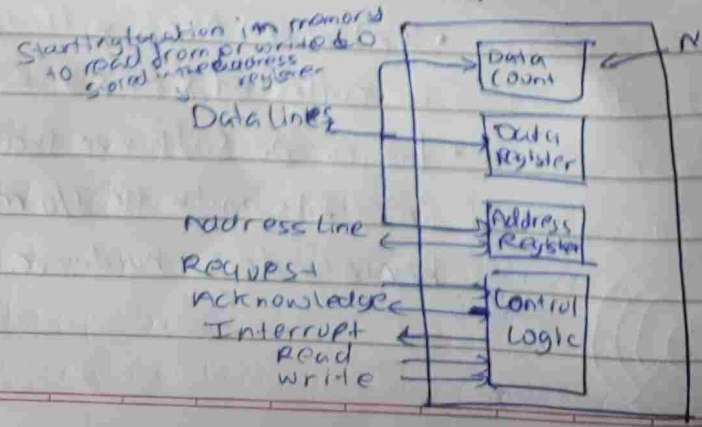
- The processor issues an I/O command, on behalf of an process
- There are 2 possibility:-
- If I/O instruction from the process is nonblocking, the

processor continues to execute instructions from the process that issue the I/O command

- If the I/O instruction is blocking, then the next instruction that the processor executes from the OS which will put the current process in a block state, interrupt the current process & schedule another process

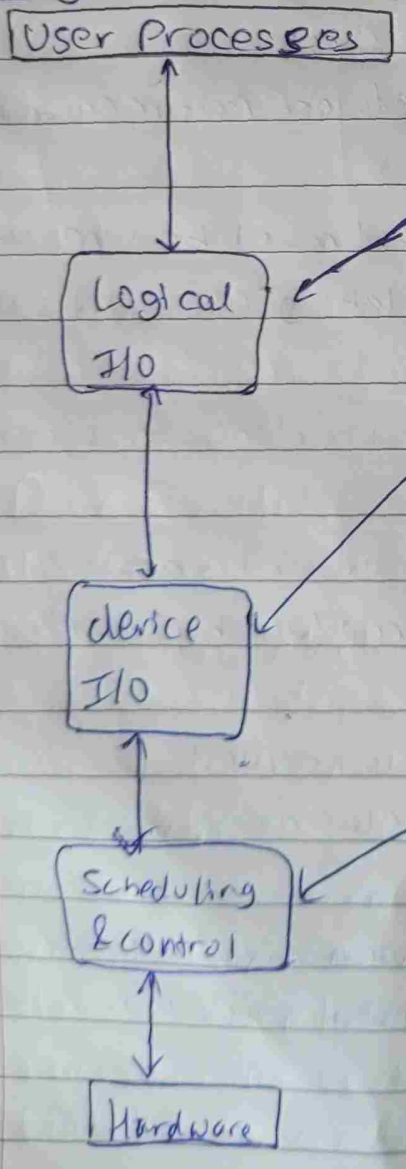
→ Direct Memory Access

- The processor sends a request for the transfer of a block of data to the DMA module & is interrupted only after the entire block has been transferred.
- It controls the exchange of data between main memory & the I/O module
- It is the dominant form of transfer that must be supported by the OS
- Capable of mimicking the processor, taking control of system bus just like a processor.
- The transfer of data to & from memory is done over system bus
- Working:
  - When the processor wishes to read or write a block of data, it issues a command to the DMA module by sending to the DMA module some info
  - The info is whether read or write is requested
  - Using the read or write control line between the processor & the DMA module
  - The address of the I/O device involved, communicated on the data lines



- The processor cont. with other work.
- The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the processor
- When it's over the DMA sends an interrupt signal to the processor
- ~~DMA~~ Processor involved only at the beginning & end of the transfer.

6) Logical Structure



Concerned with managing general I/O function on behalf of user processes

The requested operations & data are converted into appropriate sequences of I/O instructions, channel commands & controller order

- Actual queueing & scheduling of I/O operations occurs at this layer, as well as the control of operations
- Interrupt are handled, I/O status is collected & reported.
- Layer of software that actually interacts with the I/O module & hence the device hardware.

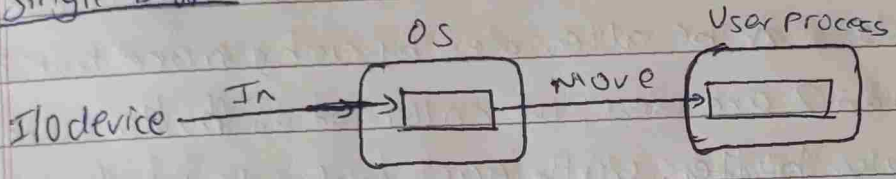
7) I/O Buffering

→ It is a technique used in computer systems to improve the efficiency of input & output operations.

→ It involves temporary storage of data in a buffer, which is reserved area of memory

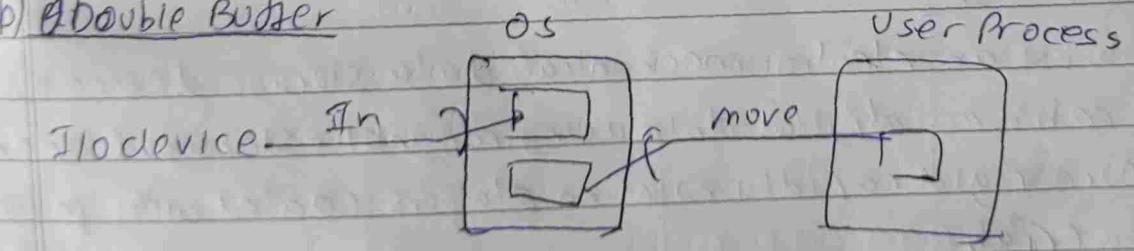
→ Types

a) Single buffer



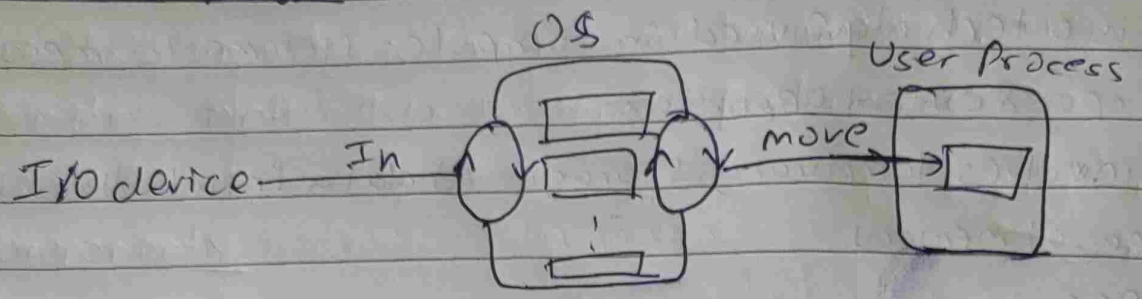
- Simplest type of support that the OS can provide
- When user process issues an I/O request, the OS assigns a buffer in the system portion of main memory to the operation.
- Input transfers are made to the system buffer
- When completed, the process moves the block into user space & immediately request another block

b) Double Buffer



- Assigning two system buffers to the operation:
  - A process now transfers data to or from one buffer
  - while the OS empties or fills the other
- this is known as buffer swapping

c) Circular Buffer



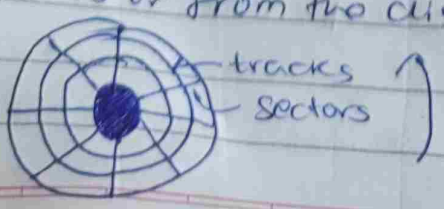
- If the process performs rapid burst of I/o, in this case the problem can often be alleviated by using more than 2 buffers.
- When more than 2 are used, the collection of buffer is referred to as circular buffer, with each individual buffer being one unit in the circular buffer.

→ It helps to smooth out peaks in I/O demand.
→ It can ↑ the efficiency of the OS & the performance of individual processes.

Disk scheduling

→ Hard Disk

- It is an electro-mechanical data storage device that stores & retrieves digital data using magnetic storage & one or more rigid rapidly rotating platters coated with magnetic material.
- The platters are paired with magnetic ~~disk~~ heads, usually arranged on a moving actuator arm, which read & write data to the platter surfaces
- Each surface is divided into concentric circles, creating tracks
- Each tracks are further divided into sectors (is the smallest unit of data transfer to or from the disk)



→ Some imp terms

a) Seek time
- It is the time taken to move a disk arm to a specific track containing the desired sector where the data is to be read or written.

b) Rotational Latency
- It is the time taken by the desired sector of disk to rotate to the disk header, ie. into a position so that it can access the read/writer heads.

c) Disk bandwidth
- Total no° of bytes transferred, divided by the total time between the first request for service & the completion of last transfer.

d) Transfer time
- Time to transfer data, depends on the rotating speed.

e) Disk access Time = Seek Time + Rotational Latency + Transfer time

f) Disk response Time
- It is the avg of time spent by request waiting to perform it's I/O operation.
- Variance RT is measure of how individual request are served with respect to avg response time.

→ Disk management

Techniques
a) Partitioning : This involves dividing a single physical disk into multiple logical partitions. Each partition can be treated as a seperate storage device.

b) Formatting : This involves preparing a disk to use by creating a file system on it

c) File system management : This involves managing the file systems used by the OS to store & access data on the disk

d) Disk space allocation : This involves ~~managing~~ allocating space on the disk for storing files & directories.

e) Disk defragmentation: Over time, as files are created & deleted the data on a disk can become fragmented. It involves rearranging the data on the disk to improve performance

Components

a) Disk formate

i) Low-level Formate

- Dividing a disk into sectors that the disk controller can read & write
- Each sector can hold, header info, data & error correction code
- Conducted in 2 stages
  - Dividing the disk into cylinder groups
  - Logical format: The OS creates the file system, storing structure for free & allocated space.

ii) Logical Format

- OS organize the data into file systems
- File system groups blocks into clusters for efficiency.

b) Booting

- The bootstrap loader program initializes the systems & starts the OS
- Stored in ROM
- Full bootstrap program, stored in the boot block of the disk & handles loading of the OS kernel into memory.

c) Bad Blocks

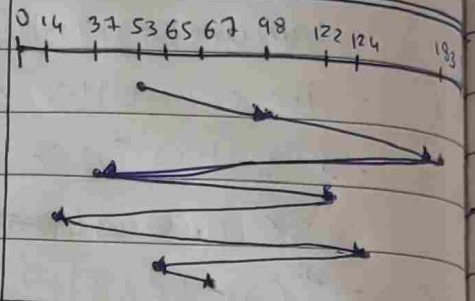
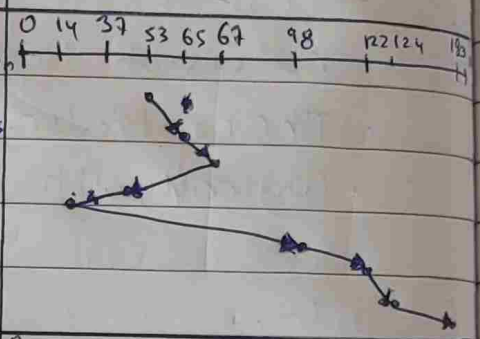
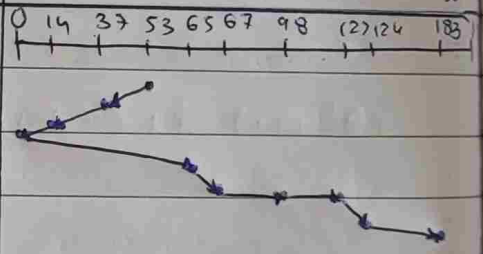
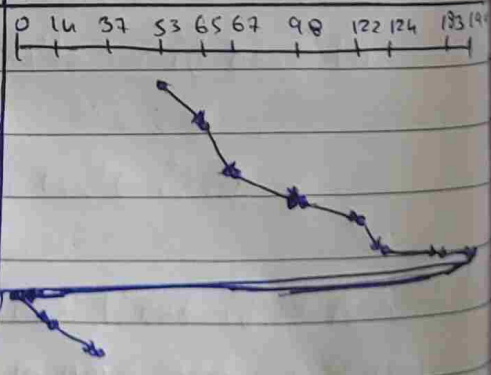
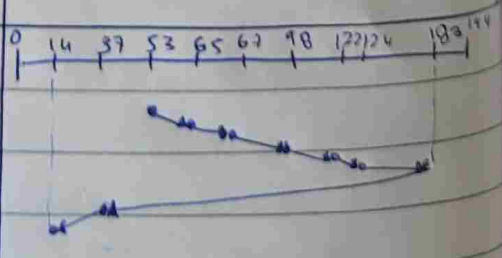
- methods such as sector sparing used to handle bad blocks
- Because disk have moving parts & small tolerance.
- They are prone to failure.
- If failure is complete, disk need to be replaced & it's contents restored.

- Simple Disks, such as in IDE controllers they are handled manually.
- For more sophisticated disk, used in high-end PC's are smarter about bad block recory.
  - The controller maintains a list of bad blocks on the disk
  - The list is initialized during the low-level formatting at the factory & is updated over the life of the disk
  - Low-level formatting also sets aside spare sectors not visible to the OS
  - The controller can be told to replace each bad sector logically with one of the spare sectors. Known as sector sparing

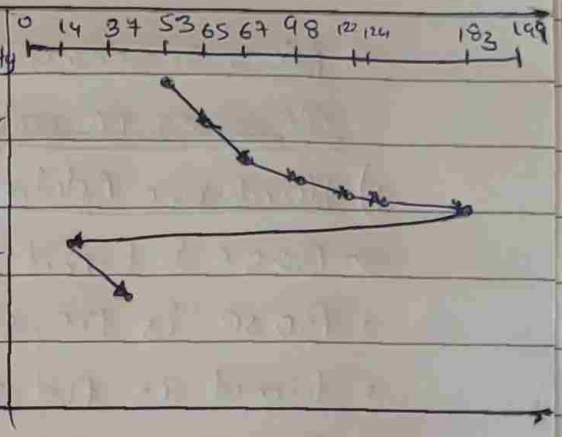
Date
Page

Algos

98, 183, 37, 122, 14, 124, 65, 67 header = 53

Name	Description	Advantage	Disadvatage	Sum
FCFS (First Come First Serve)	Processes request in the order they arrive	- Simple to implement - Fair to all requests	- May cause long wait time - Not efficient for high workload	
SSTF (Shortest seek time)	Picks the request closest to the current head position	- Reduce total seek time compared to FCFS - Improves throughput	- Can cause starvation for far off requests - May Not be fair	
SCAN (Elevator Algo)	Moves the head back & forth Serving request in One direction at a time	- Reduce Variance in Response Time - Ensures No Starvation	- Longer seek time for requests - Not optimal for uniformly distributed request	
C-SCAN (Circular Scan)	Similar to Scan but only process request in one direction, returning to start without servicing requests	- Provides more uniform wait times than SCAN - Avoids Starvation	- Longer seek time do requests at the far ends - Increase travel distance compared to SCAN	
LOOK	An optimized version of SCAN that stops moving the head when there is no request in a direction	- Reduce Unnecessary movement - Efficient for dense workload	- Can still have longer wait times compared to SSTF	

LOOK	Circular version of LOOK, head only moves in one direction & jumps back to start	- Reduce Head movement compared to C-scan - Ensures fairness & uniform response	- Increase complexity - Can still suffer from seek time inefficiencies for certain patterns



Which to Select?

1) Efficiency
   - → SSTF, SCAN & LOOK are more efficient than FCFS in min seek time
   - → C-SCAN & C-LOOK provide more uniform wait times than SCAN & LOOK
2) Fairness
   - → FCFS & C-SCAN ensures fairness but may sacrifices efficiency
   - → SSTF can lead to starvation
3) Complexity
   - → FCFS is simplest
   - → LOOK & C-LOOK have more efficient head movements but are slightly more complex.

MODS

~~Main Memory~~

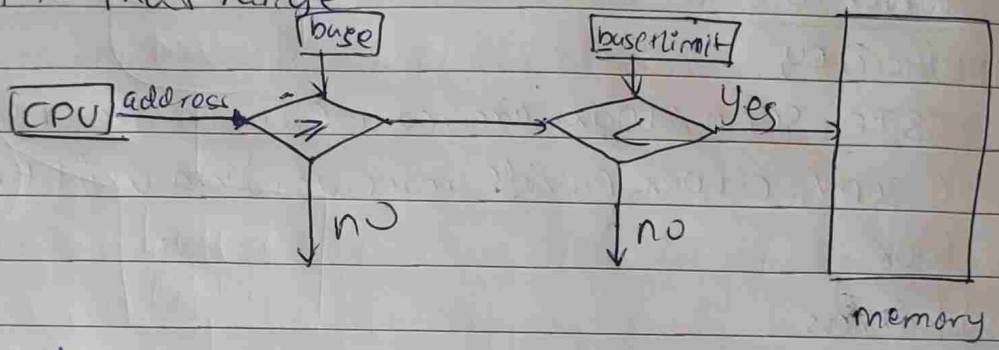
~~Memory Management~~

1) Hardware Address Protection

→ Base & Limit registers define the logical address space

- Base is the smallest legal address
- Limit is the size of the range

→ CPU must check every user-mode memory access to ensure it is that range



2) Address binding

→ A program on disk are brough into memory to create running Process

→ It happen at 3 different points:

- Compile time: If memory location known as priori, absolute code can be generated, requires recompilation if base location changes
- Load time: Need to generate relocation code if memory location is not known at ~~compile~~ compile time.
- Execution time: Binding Delay until run time if the process can be moved during it's execution from one memory segment to another

3) Logical Vs Physical Address

→ Logical Address

- Also known as virtual address
- Generated by the CPU during program execution
- Used within the program.

→ Physical Address

- Actual address in main memory
- Computed by adding the base address to the logical address

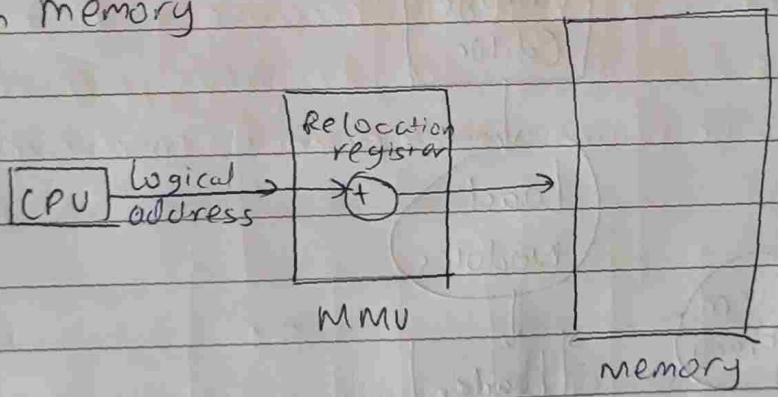
4) Memory Management Unit

→ Hardware that maps logical to physical at run time

→ Conceptually simple schema ⇒ replace base register with relocation register

→ Add the value in the relocation register to every address generated by a user process at the time it is sent to memory

→ Execution time binding occurs when reference is made to location in memory



5) Dynamic Linking & Loading

→ Linking combines different Object code modules to create program

- Static Linking - all libraries and program code combined into the binary program image
- Dynamic Linking - Postpone Linking to execution time

→ It is useful for system or shared libraries

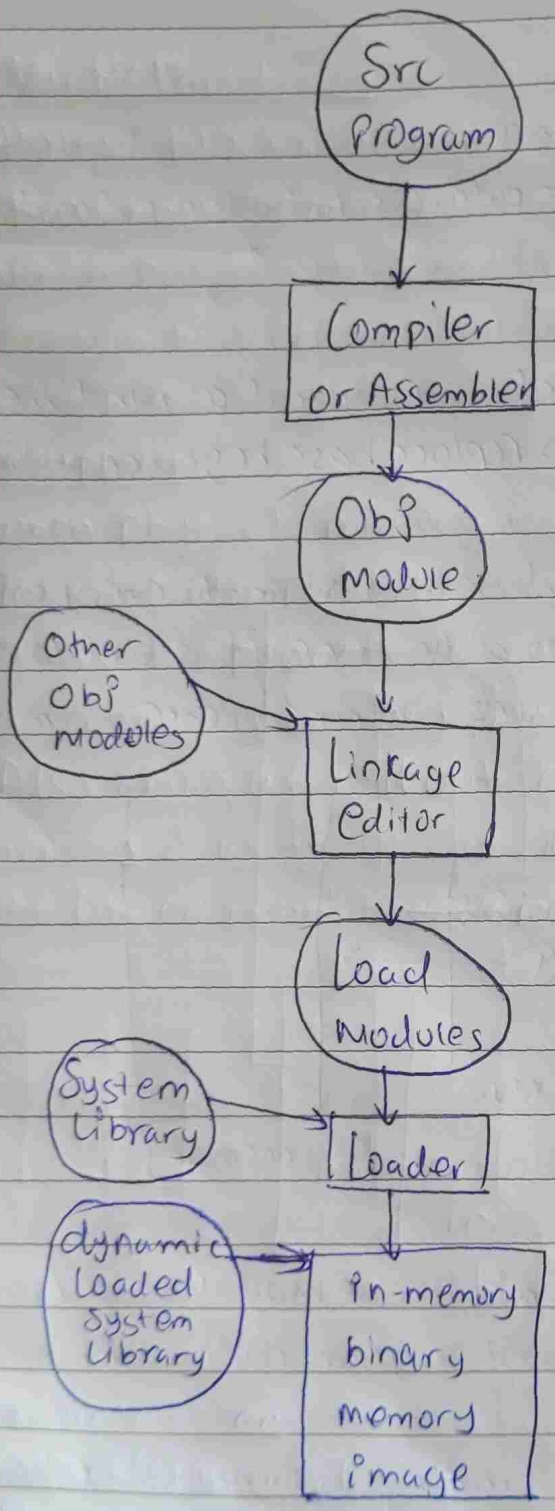
→ Calls are replaced with stubs - a small piece of code to locate the appropriate in memory routine

→ Stub ~~itself~~ replaces itself with the address of the routine, & executes the routine.

→ Dynamic loading avoids loading routines until they're called

- Better memory usage

→ OS can help by providing libraries to implement dynamic loading


Src Program
Compiler or Assembler
Obj Module
Other Obj Modules
Linkage Editor
Load Modules
System Library
Loader
dynamic Loaded System Library
In-memory binary memory image

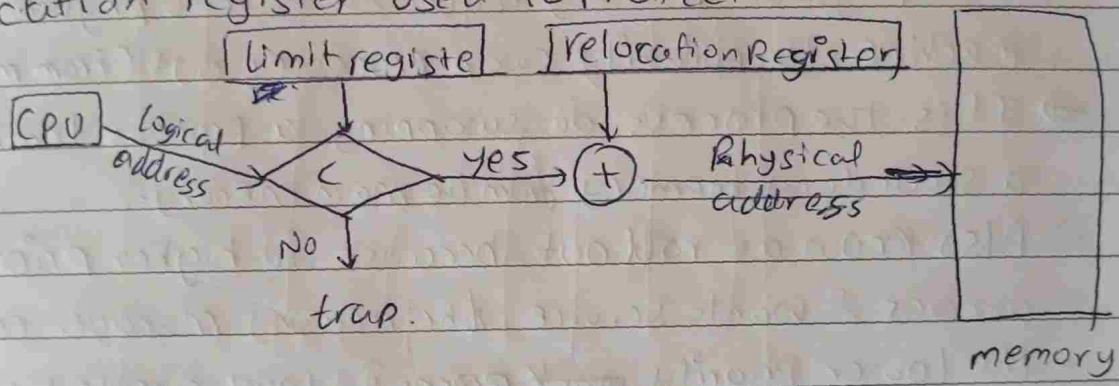
6) Memory Allocation

→ Main memory must support both kernel & user processes, limited resource, must allocate efficiently

→ Contiguous allocation is early method putting each process in one chuck of memory

→ Main memory usually partitioned into 3 Resident kernel (low memory) & User Process (high memory)

→ Relocation register used to protect

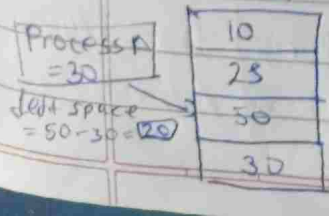


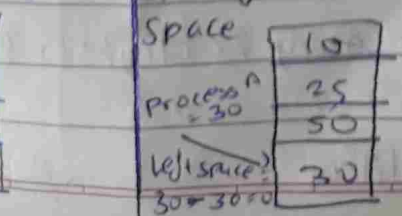
→ Chucks are determined in ~~two ways~~

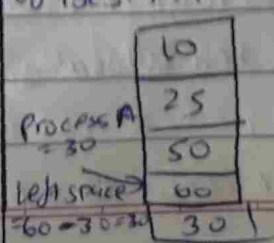
i) Fixed Partitioning

- In this method, OS maintains a table that indicates which part of memory are available & which are occupied by processes
- Initially all the memory are available ~~& which are occupied~~ for user process & is considered 1 large block of available memory (known as Hole)
- While allocating memory sometimes dynamic storage allocation problem occurs, which concerns how to satisfy a request of size n from a list of free holes.
- The soln:

First Fit	Best Fit	Worst Fit	Next Fit
→ Chooses the first available space	→ It searches the list & finds the best space	→ It is opposite of Best Fit	→ It is similar to first fit but it will search for the first sufficient partition from the last allocation point







ii) Varible Partitioning

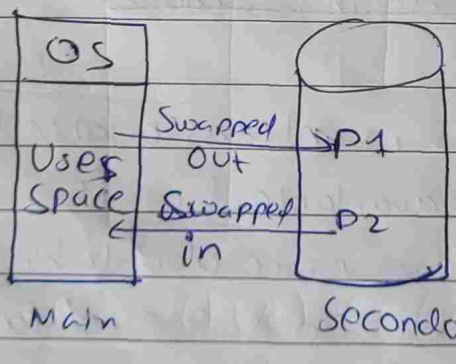
- Holes, blocks of available memory of variable size are scattered throughout the memory
- When process arrives, it is allocated memory from hole large enough to accomodate it

7) Swapping

→ When physical memory requested exceeds physical memory in machine, temporarily swap processes out (From main to storage)

- If is the process of swapping a process temporarily into a secondary memory from the main memory.

→ Also know as roll out, because if higher priority process arrives & wants service, the memory manager can swap out the lower priority ~~work~~ process & load, execute the higher process



8) Fragmentation

→ External Fragmentation

- Occurs when free memory exists to satisfy a request but it is not contiguous
- Can eventually ~~lead to~~ result in blocking as insufficient contiguous memory to swap any process in

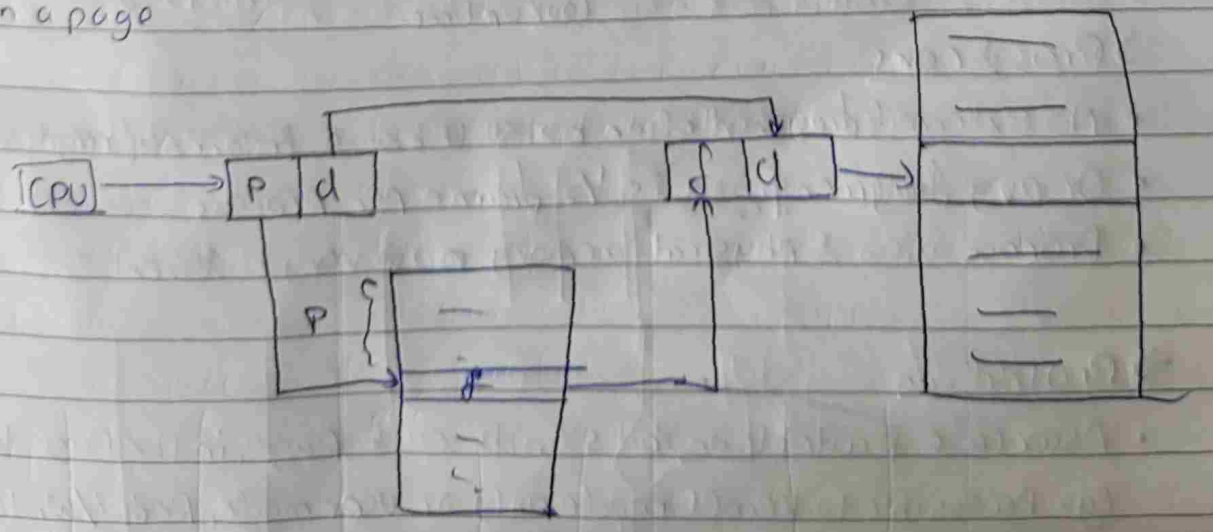
→ Internal Fragmentation

- Occurs when allocation memory is slightly larger than requested memory
- Memory internal to a partition

→ To reduce external fragmentation we can use the concepts of compaction
→ Basically shuffling memory contents to place all free memory together in one large block
→ It is only possible if, relocation is dynamic & done at execution Time

9) Paging
→ Physical address space of a process can be Noncontiguous process is allocated in physical memory whenever Latter is available
→ Divide physical memory into fixed-sized blocks, are called frames
→ Divide logical memory " " " " of same size are are called pages.
→ This approach prevents fragmentation issues by keeping memory allocation uniform.
→ Each program has a page table, which the OS uses to keep track of where each page is stored in physical memory.
→ Page no°(p): no° of bits required to represent the page in logical address
→ Page offset(d): no° of bits required to represent a particular word in a page

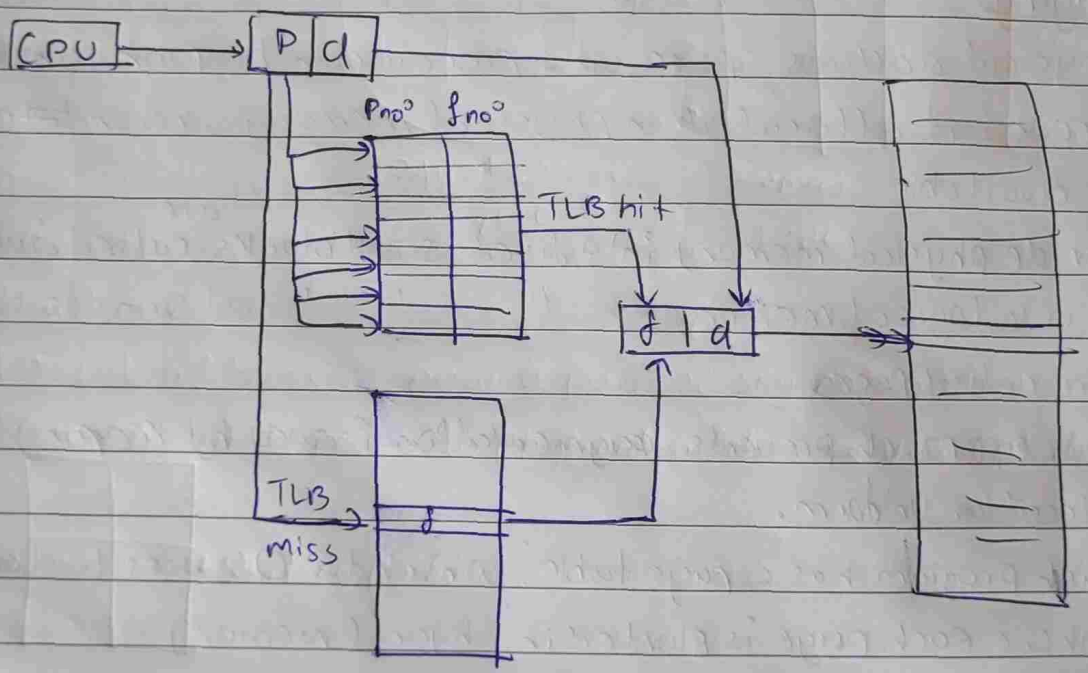
Logical address



→ In this schema every data/instruction access requires two memory association.

→ The two memory ~~association~~ access problem can be solved by using a special fast-lookup hardware cache called translation look aside buffer (TLB)

→ On a TLB miss, a value is loaded into TLB for faster access next time



→ Effective access time = $(1+\varepsilon)$ ~~α~~ $+ (2+\varepsilon)(1-\alpha)$

↑ Associative lookup time ↑ Hit ratio

→ Pros & cons

- No external fragmentation but still have internal fragmentation
- On avg fragmentation is 1/2 frame per process
- Process view & physical memory now very different

→ Protection

- Associate protection bits with each page, in the Page table entry
  eg. Accessible to kernel mode only or user mode, Read/Write/Executed
  Valid/invalid.

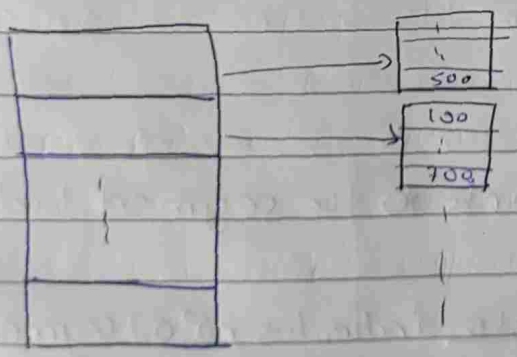
- As the address goes through the page hardware, protection bits are checked
- Attempts to violate protection cause a hardware trap to the OS

→ Sharing

- Shared code
  - Keep just one copy of read-only code shared among processes
  - Similar to multiple threads sharing the same process space
- Private code
  - Each process keeps its own copy of private code & data
  - Pages for which can appear anywhere in the logical address space

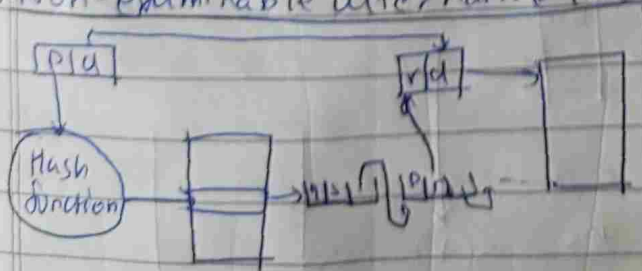
→ Page table structure

- Page table can get huge using straigh forward methods
- Instead, split the page table into multiple levels.

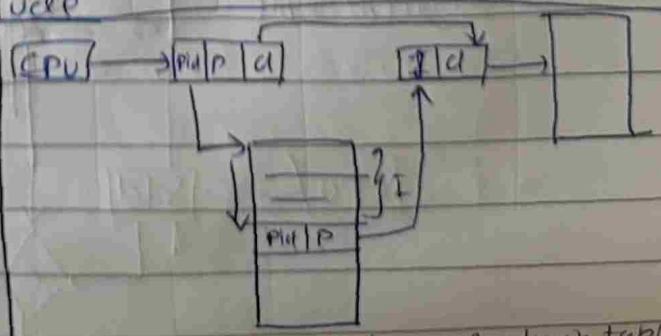


→ For larger address spaces simple hierarchy is impractical

→ Non-examinable alternative include



Page no° is hashed into a table & chained followed until the specific entry is found



With an entry for each frame & a hash table used to limit the search to one or few entries, trading size for lookup latency

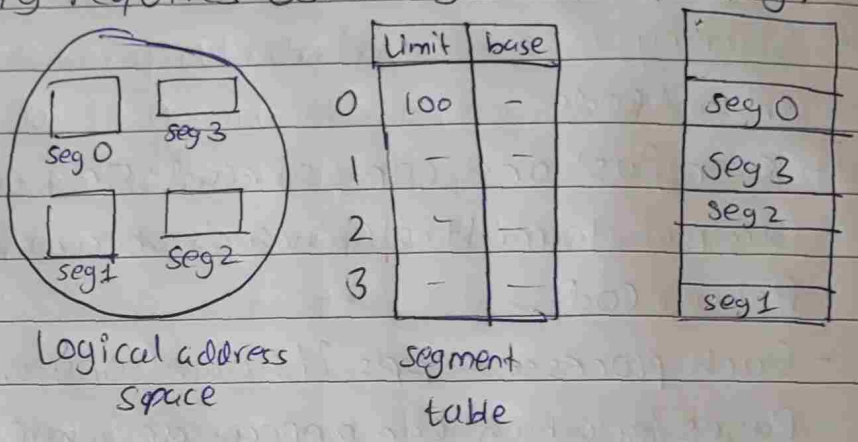
10) Segmentation

→ It is a memory management schema supporting user view of memory

→ View a program as a collection of segments, logical program unit

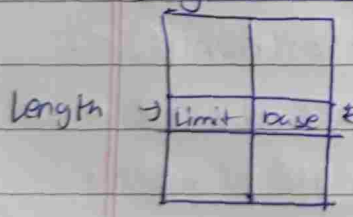
→ Accessing memory requires usr program to specify:

- Segment name
- Offset



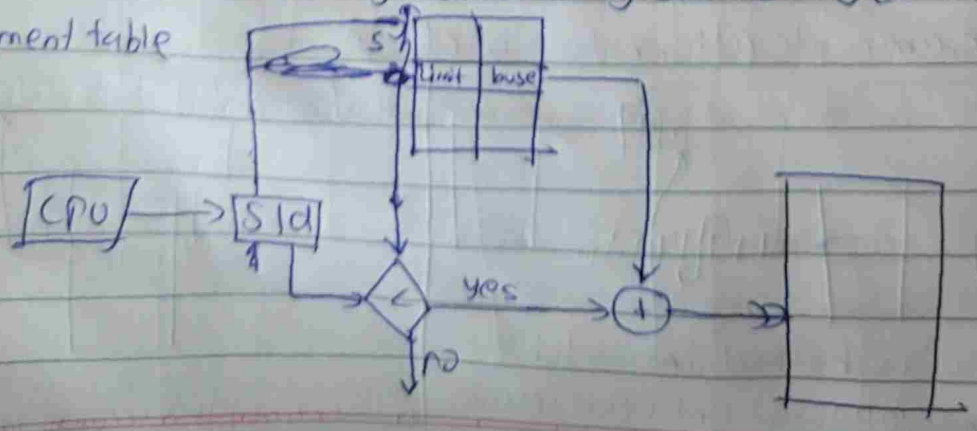
→ Hardware

- Logical address is now a pair of seg no°, offset
- Segment table maps to physical addresses via entires having:



- Segment table base register points to the segment table's location in memory
- Segment table limit register indicates no° of segments used by a program

→ Protection provided by associating with ~~entry~~ each entry in the segment table



11) Virtual Memory

→ It is a method that computers use to manage storage space to keep system running

→ It allows the system to compensate for physical memory shortages, enabling larger applications to run on system with less RAM

→ Benefits:

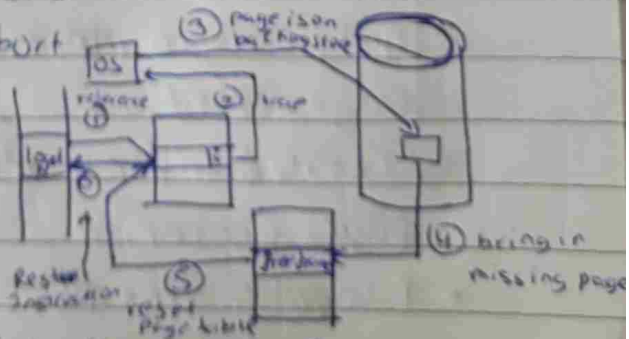
- Portability: Program works regardless of how much physical memory. Can be larger than physical memory
- Convenience: Less program needs to be in the memory at once
- Efficiency: No need to waste memory on code or data which isn't used

→ Address Space

- Gives the logical view of how process is stored in memory
- Usually stack starts at maximum logical address & grows 'down' while heap grows 'up'
- NO physical memory needed until heap or stack grows to a new page
- System lib shared via mapping into virtual address space.

→ Page Fault

- when an invalid page is referenced it causes a trap to the OS
- The OS handles the trap by examining another table:
  - If invalid memory reference, then abort
  - If valid but not resident, find free frame & swap the page in
  - Entry is now marked valid as page is in memory



- After which restart the instructions that caused the fault

→ Demand Paging

- It is a technique used in Virtual memory systems where pages enter main memory only when requested or needed by the CPU
- In this the OS system loads only the necessary pages of a program into memory at runtime
- Due to this Page fault occurs when a program needs to current use a page that is not in the memory
- Lazy Swapper never swaps a page into memory unless page will be needed.
- Performance?

- Can improve system performance by reducing the memory need for programs & allow multiple programs to run simultaneou
- However if not implemented properly it can cause performance issues.

→ Optimization

- Swap space I/O can be faster than file system I/O even on same device
- Demand page program from binary on disk, discard when freeing unmodified frame.
- Copy-on-write (COW)

- Both parent & child processes initially share same pages in memory
- Only when a process actually modifies a shared page is the page copied
- It allow for more efficient process creation as only modified pages are copied.

- Allocate free pages from pool of zero fill on demand pages

→ Paging in from disk requires a free frame, but physical memory is limited

→ To combat this we can discard the ~~used~~ unused pages or we swap it using page replacement Algo.

Algo	Replacement Policy	Advantage	Disadvantage	Use Case	Analysis
FIFO	Replaces the oldest page in the memory	- Simple to implement - Requires minimal computation	- Suffers from Belady anomaly where ↑ frame can ↑ page fault	∴ Suitable when simplicity & low computational are priorities	- Low efficiency in term of page faults Performs poorly when frequent page reuse occurs as it does not account for recency or future use
Optimal Page Replacement	Replaces the page that will not be used for the longest time in the future	- Produces the least no° of Page Fault - Sets a benchmark for other algo	- Not implementable in practice as future reference are unknown	Useful as a benchmark for comparing other algo	Highest efficiency as it min. page faults by perfect future knowledge.
Least Recently Used	Replaces the page that has not been used for the longest time	- Takes past behavior is account improving efficiency over FIFO	- Higher Computational overhead requires tracking usage history	Effective when recent usage patterns are predictive of future needs	High efficiency in practice, as it uses past data to predict future needs. Performs poorly if access patterns are random or irregular.

→ ~~Thrashing~~

→ Frame allocation

- Determines how physical memory frames are distributed among processes
- Reserves memory for OS code & data before distributing remainging memory to processes
- Objectives
  - Fairness : Allocate memory equally among processes & based on process size.
  - Minimize system wide page fault rate
  - Maximize multiprogramming level.

→ Global vs Local Allocation

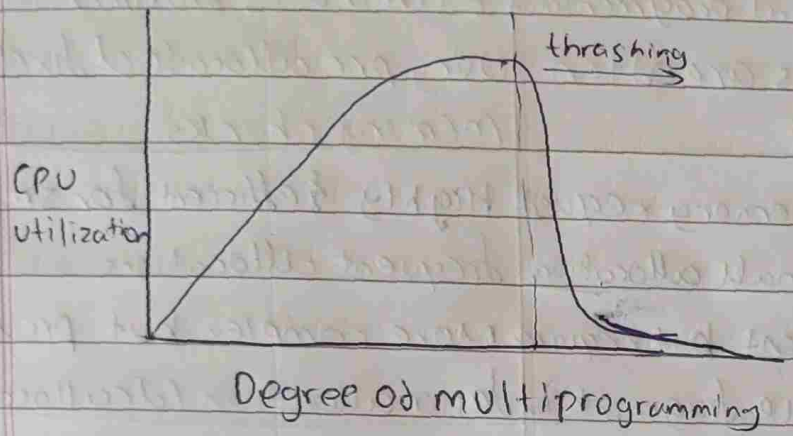
- Global : Any page in the system can be selected as a victim for replacement, results in higher throughput due to dynamic memory usage, it is more flexible but can cause performance variations across processes.
- Local : A process can only replace pages within it's allocated frame ensures consistent performance for each process but may underutilize memory provides predictability but at the cost of potentially wasting memory.

→ Thrashing

- A state where a process experiences frequent page faults, causing system performance to degrade significantly
- Cause ⇒ A process doesn't have "enough" pages to support it's working set
- Symptoms ⇒ Frequent page replacements for ~~failure~~ frames that are immediately needed again, wasted CPU cycles & low system utilization OS mistakenly increases multiprogramming by adding more processes worsening memory.

→ Result =>

- Prevention => use load or priority page replacement policies, monitor the working set to ensure processes have enough frames.



Working Set

- Avoid thrashing by considering the working set
  - Those pages required at the same time for process to make progress
  - Varies between processes & during execution
  - Assume process shifts phases but gets locality of reference in each phase.
- Demand $D = \sum_i (\text{Working Set})_i$, approximation of locality
  - Thrashing occurs if $D > m$ (no. of frames), in which case suspend/swap out a process
  - Prepaging: bring in working set pages when (re-) starting a process

12) Allocating Kernel Memory

Aspect	Buddy System	Slab System
Purpose	Used for variable sizes	Used for Fixed size
Fragmentation	May lead to internal fragmentation as allocated blocks are often larger than required	Eliminates fragmentation by using pre allocated fixed size memory chunks
Efficiency	Efficient for Large memory request but not ideal for small allocations	Highly efficient for small & frequent allocations
Complexity	Simpler to implement but requires coalescing to handle fragmentation	More complex but provides faster allocation / deallocation.
Coalescing	Combines adjacent free blocks to larger blocks when memory is freed	Not required as memory chunks are reused from chunks
State management	Tracks memory as free or allocated blocks	Tracks slab states are full, empty or partially for Optimal allocation
Allocation speed	Moderate, as it may require searching & splitting blocks	Faster, as objects are pre-allocated & reused for caches
Diagram