Session 3&4: (4T+2L)

Introduction to memory management

o Logical Address

- Also called virtual address.
- Does not refer to an actual location in physical memory.
- Generated by the CPU during program execution.
- Translated to physical address using Memory Management Unit (MMU).
- Visible to the user/programmer.
- Used in user-space applications.

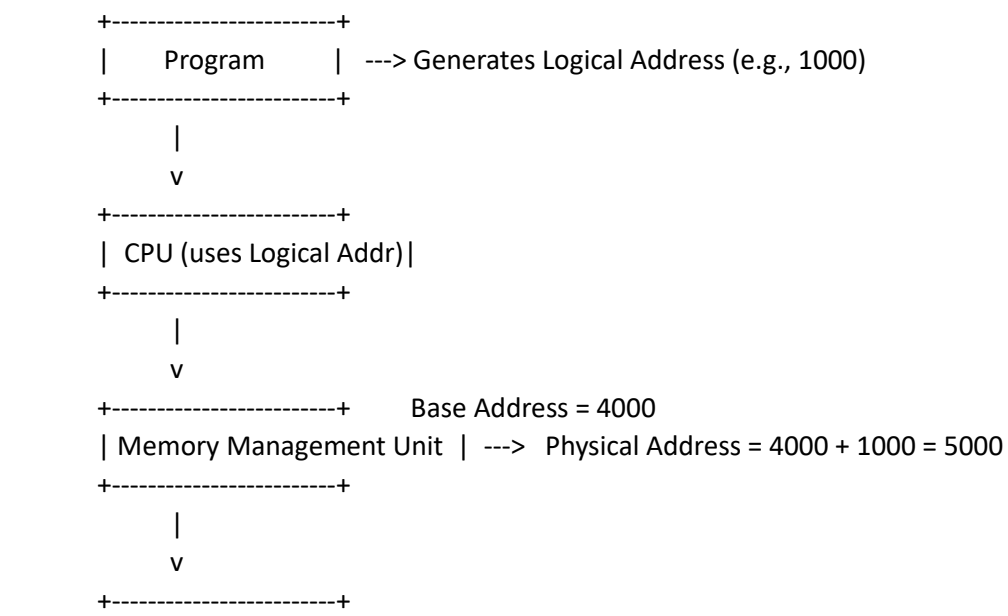
o Physical Address

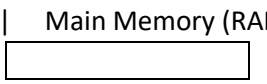
- Actual location in **main memory (RAM)**.
- Used by the **memory unit** to access data.
- **Not visible to the user**; only the OS handles it.
- Obtained after **address translation** from logical address.
- Example: If logical address = 1000, base = 4000 → physical = 5000.

o Dynamic linking

- Linking of libraries **at runtime**, not during compile time.
- Reduces program size; **shared libraries** used by multiple programs.
- Library code is **loaded into memory only when needed**.
- Provides **flexibility** and **modularity**.
- Example: DLLs in Windows, .so files in Linux.

1. Logical vs Physical Address Mapping Diagram

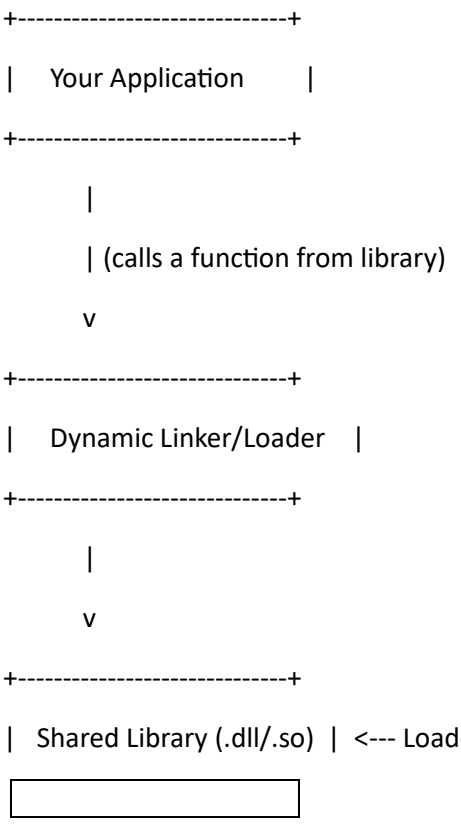




Logical Address

- It's the address seen by the process (or programmer).
- Think of it like an apartment number. You give this to someone, but internally the building has actual physical coordinates.
- Generated by the CPU when the program accesses a variable or instruction.
  - Physical Address
- The real address in RAM where data is stored.
- This is calculated based on a mapping scheme (like base + offset).
- Managed by OS + MMU (Memory Management Unit).
  - Real-Life Analogy:
- Logical Address: Your home's apartment number (e.g., Flat 203).
- Physical Address: The GPS location (e.g., Latitude 52.379, Longitude 4.900).

Dynamic Linking Diagram



Dynamic Linking - More Info

- Instead of copying the library code during compilation, a reference is added.
- At runtime, the dynamic linker loads the actual library into memory.
- Saves space and enables updates (update one .dll, all programs benefit).
  - Real-Time Example:
- On Windows, when an app uses kernel32.dll, it doesn't include its code.
- The DLL is loaded dynamically when the app runs.

Memory Management

- In multi-programming OS, multiple programs are loaded in memory.
- RAM memory should be divided for multiple processes running concurrently.
- Memory Mgmt scheme used by any OS depends on the MMU hardware used in the machine.
- There are three memory management schemes are available (as per MMU - Memory Management Unit hardware).

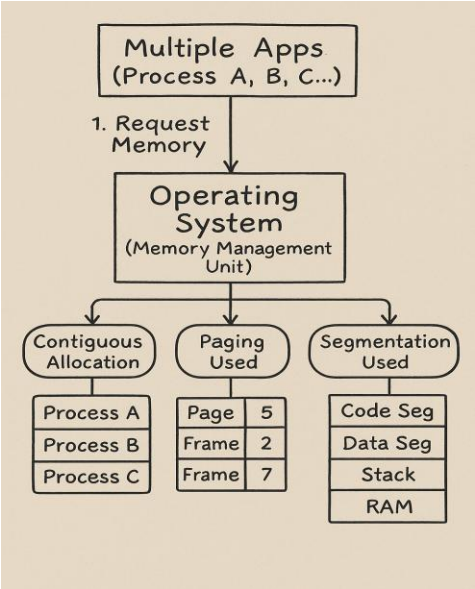
1. Contiguous Allocation
2. Segmentation
3. Paging

Big Picture: What's Happening in Memory?

When **multiple processes run concurrently**, the **Operating System (OS)** manages memory using **memory management techniques** to:

- Load processes into RAM
- Prevent overlapping
- Efficiently utilize available space
- Translate logical to physical addresses

Daigram:-



Step-by-Step OS Memory Management

1. **Process Creation:**
    - A user/application starts a program → OS creates a process.
2. **Memory Request:**
    - The process asks the OS for memory (code, data, stack).
3. **OS Allocates Memory Based on Scheme:**
    - If **Contiguous Allocation** → Entire block is given continuously.
    - If **Paging** → Process divided into fixed pages → mapped to available frames.
    - If **Segmentation** → Process divided into segments (code, data, stack) → placed in memory.
4. **Address Translation:**
    - OS translates **Logical Address → Physical Address** using **Page Table** or **Segment Table**.
5. **Process Execution:**
    - The CPU uses these mappings to access memory safely and efficiently.

<u>1.Contiguous Allocation</u>

Fixed Partition

- RAM is divided into fixed sized partitions.
- This method is easy to implement.
- Number of processes are limited to number of partitions.
- Size of process is limited to size of partition.
- If process is not utilizing entire partition allocated to it, the remaining memory is wasted. This is called as "internal fragmentation".

Dynamic Partition

- Memory is allocated to each process as per its availability in the RAM. After allocation and deallocation of few processes, RAM will have few used slots and few free slots.
- OS keep track of free slots in form of a table.
- For any new process, OS use one of the following mechanism to allocate the free slot.
  - First Fit: Allocate first free slot which can accommodate the process.
  - Best Fit: Allocate that free slot to the process in which minimum free space will remain.
  - Worst Fit: Allocate that free slot to the process in which maximum free space will remain.
- Statistically it is proven that First fit is faster algo; while best fit provides better memory utilization.
- Memory info (physical base address and size) of each process is stored in its PCB and will be loaded into MMU registers (base & limit) during context switch.
- CPU request virtual address (address of the process) and is converted into physical address by MMU as shown in diag.
- If invalid virtual address is requested by the CPU, process will be terminated.
- If amount of memory required for a process is available but not contiguous, then it is called as "external fragmentation".
- To resolve this problem, processes in memory can be shifted/moved so that max contiguous free space will be available. This is called as "compaction".

<u>2.Paging</u>

- RAM is divided into small equal sized partitions called as "frames" / "physical pages".
- Process is divided into small equal sized parts called as "pages" or "logical/virtual pages".
- page size = frame size.
- One page is allocated to one empty frame.
- OS keep track of free frames in form of a linked list.
- Each PCB is associated with a table storing mapping of page address to frame address. This table is called as "page table".
- During context switch this table is loaded into MMU.
- CPU requests a virtual address in form of page address and offset address. It will be converted into physical address as shown in diag.

- MMU also contains a PTBR, which keeps address of page table in RAM.
- If a page is not utilizing entire frame allocated to it (i.e. page contents are less than frame size), then it is called as "internal fragmentation".
- Frame size can be configured in the hardware. It can be 1KB, 2KB or 4KB, ...
- Typical Linux and Windows OS use page size = 4KB.

3.Segmentation

- Instead of allocating contiguous memory for the whole process, contiguous memory for each segment can be allocated. This scheme is known as "segmentation".
- Since process does not need contiguous memory for entire process, external fragmentation will be
- reduced.
- In this scheme, PCB is associated with a segment table which contains base and limit (size) of each segment of the process.
- During context switch these values will be loaded into MMU segment table.
- CPU request virtual address in form of segment address and offset address.
- Based on segment address appropriate base-limit pair from MMU is used to calculate physical address
- as shown in diag.
- MMU also contains STBR register which contains address of current process's segment table in the
- RAM.

    - **Demand Segmentation**
        - If virtual memory concept is used along with segmentation scheme, in case low memory, OS may swap out a segment of inactive process.
        - When that process again start executing and ask for same segment (swapped out), the segment will be loaded back in the RAM. This is called as "demand segmentation".
        - Each entry of the segment table contains base & limit of a segment. It also contains additional bits like segment permissions, valid bit, dirty bit, etc
        - If segment is present in main memory, its entry in seg table is said to be valid (v=1). If segment is swapped out, its entry in segment table is said to be invalid (v=0).

4.Segmentation with paging

What is “Segmentation with Paging”?

It is a hybrid memory management scheme that:

- Divides a program into segments (like code, data, stack).
- Then, each segment is divided into fixed-size pages.
- Pages are mapped to physical memory using a page table for each segment.

Why Combine Segmentation and Paging?

Segmentation Alone	Paging Alone	Combined (Segmentation + Paging)
Logical division of program (modules)	Eliminates external fragmentation	Combines logical view + efficient memory use
Suffers from external fragmentation	Suffers from internal fragmentation	Reduces both
Each segment has variable size	Fixed-size pages	Segments of pages

Structure Overview

1. Logical Address = Segment Number + Page Number + Offset
2. Each segment has its own page table
3. Address translation is done in 3 steps

Advantages

- Logical organization (via segments)
- Avoids external fragmentation (due to paging)
- Better protection and sharing between processes
- Efficient memory usage

Disadvantages

- More complex hardware and address translation
- Needs **both** segment and page tables → more overhead
- Slightly slower due to multi-level address translation

Real-World Use

This technique is used in systems like **Multics**, and early versions of **Intel x86 architecture** used segmentation with paging (e.g., 80286, 80386 processors).

Virtual Memory

- Virtual memory is a memory management technique where the system executes processes that may not be completely in main memory (RAM). It gives an illusion that each process has a large continuous memory space, even more than physical memory.
- The portion of the hard disk which is used by OS as an extension of RAM, is called as "virtual memory".

- If sufficient RAM is not available to execute a new program or grow existing process, then some of the inactive process is shifted from main memory (RAM), so that new program can execute in RAM (or existing process can grow). It is also called as "swap area" or "swap space".
- Shifting a process from RAM to swap area is called as "swap out" and shifting a process from swap to RAM is called as "swap in".
- In few OS, swap area is created in form of a partition. E.g. UNIX, Linux, ...
- In few OS, swap area is created in form of a file E.g. Windows (pagefile.sys), ...
- Virtual memory advantages:
  -Can execute more number of programs.
  -Can execute bigger sized programs.

Key Features:

- Uses disk space (swap space) as an extension of RAM.
- Enables multiprogramming and larger program execution.
- Improves CPU utilization and system throughput.

o Swapping

Swapping is the process of **moving an entire process in or out of main memory**.

- **How It Works:**
  - If RAM is full, an **idle or lower-priority process** is moved to disk (swap space).
  - When needed again, it is brought back into RAM.
- **Pros:**
  - Allows more processes to run concurrently.
- **Cons:**
  - Can be slow due to disk I/O (called **thrashing** if overused).

o Demand Paging

- When virtual memory is used with paging memory management, pages can be swapped out in case of low memory.
- The pages will be loaded into main memory, when they are requested by the CPU. This is called as "demand paging".
  - Swapped out pages
  - Pages from program image (executable file on disk)
  - Dynamically allocated pages

- **How It Works:**

1. Page not in memory → **Page fault** occurs.
2. OS fetches the required page from disk.

3. Updates the page table and resumes process.

- **Benefits:**
    - **Saves memory**.
    - Allows **larger programs** to run.
    - Reduces startup time.

o Page fault exception handler

- Each page table entry contains frame address, permissions, dirty bit, valid bit, etc.
- If page is present in main memory its page table entry is valid (valid bit = 1).
- If page is not present in main memory, its page table entry is not valid (valid bit = 0).
- This is possible due to one of the following reasons:
  -Page address is not valid (dangling pointer).
  -Page is on disk/swapped out.
  -Page is not yet allocated.
- If CPU requests a page that is not present in main memory (i.e. page table entry valid bit=0), then "page fault" occurs.
- Then OS's page fault exception handler is invoked, which handles page faults as follows:
    - Check virtual address due to which page fault occured. If it is not valid (i.e. dangling pointer), terminate the process (sending SEGV signal). (Validity fault).
    - Check if read-write operation is permitted on the address. If not, terminate the process (sending SEGV signal). (Protection fault).
    - If virtual address is valid (i.e. page is swapped out), then locate one empty frame in the RAM.
    - If page is on swap device or hard disk, swap in the page in that frame.
    - Update page table entry i.e. add new frame address and valid bit = 1 into PTE.
    - Restart the instruction for which page fault occurred.

o Page-Replacement Algorithms

- While handling page fault if no empty frame found (step 3), then some page of any process need to be swapped out. This page is called as "victim" page.
- The algorithm used to decide the victim page is called as "page replacement algorithm".
- There are three important page replacement algorithms.
    - FIFO
    - Optimal
    - LRU
- **FIFO**
    - The page brought in memory first, will be swapped out first.
    - Sometimes in this algorithm, if number of frames are increased, number of page faults also increase.
    - This abnormal behaviour is called as "Belady's Anomaly".

- **OPTIMAL**
  - The page not required in near future is swapped out.
  - This algorithm gives minimum number of page faults.
  - This algorithm is not practically implementable.
- **LRU**
  - The page which not used for longer duration will be swapped out.
  - This algorithm is used in most OS like Linux, Windows, ...
  - LRU mechanism is implemented using "stack based approach" or "counter based approach".
  - This makes algorithm implementation slower.
  - Approximate LRU algorithm close to LRU, however is much faster.
- **Dirty Bit**
  - Each entry in page table has a dirty bit.
  - When page is swapped in, dirty bit is set to 0.
  - When write operation is performed on any page, its dirty bit is set to 1. It indicate that copy of the page in RAM differ from the copy in swap area.
  - When such page need to be swapped out again, OS check its dirty bit. If bit=0 (page is not modified) actual disk IO is skipped and improves performance of paging operation.
  - If bit=1 (page is modified), page is physically overwritten on its older copy in the swap area

o Thrashing

What is Thrashing?

Thrashing occurs when the system **spends more time swapping pages in and out of memory than executing actual processes.**

It results in:

- Extremely **high page fault rate**
- **Drastic performance drop**
- Processes making **little or no progress**

Why Does Thrashing Happen?

Common causes:

1. **Overcommitment of memory**:
   - Too many processes, not enough RAM.
2. **Large working sets**:
   - Each process needs more pages than available.
3. **Poor page replacement**:

- Frequently used pages are evicted.

Example:

Imagine three processes each needing 5 pages to function well, but only 10 frames available:

- OS keeps swapping in pages for one process and swapping out pages for another.
- Each access causes a **page fault**, and useful pages are **constantly removed**.
- CPU is **idle**, waiting for I/O → system becomes **unresponsive**.

Symptoms of Thrashing:

- High **CPU utilization** but little useful work done.
- Very high **page fault rate**.
- Disk I/O (swap activity) is constantly busy.

How to Prevent or Handle Thrashing:

Solution	Explanation
Working Set Model	Keep only pages a process actively uses in RAM
Page Fault Frequency (PFF)	Monitor fault rate; if too high, reduce multiprogramming
Adjust degree of multiprogramming	Limit number of active processes
Use better replacement algorithms	E.g., LRU or Clock to avoid removing needed pages
Increase RAM	Hardware solution for more capacity

Session 5: (2T)

File System Interface and Implementation

Files and Directories Operations

File Operations

Files are the logical storage units. Common operations include:

- **Create:** Allocates space for a new file in the file system.
- **Write:** Data is written into the file from the process memory.

- **Read:** Data is read from the file to memory.
- **Reposition (seek):** Move the file pointer to a desired location.
- **Delete:** Removes the file and frees up space.
- **Truncate:** Erases content but keeps file metadata.

Each file has:

- Name
- Type (e.g., .txt, .pdf)
- Location on disk
- Size
- Permissions
- Timestamps (creation, access, modification)

◆ **Directory Operations**

Directories hold files and possibly subdirectories. Operations include:

- **Create:** Makes a new directory.
- **Delete:** Removes an empty directory.
- **List:** Shows contents.
- **Rename:** Changes the name of a directory.
- **Search:** Locates a file/directory.
- **Traverse:** Goes through all entries (used in recursive operations).

File System mounting and protection methods

Mounting

Mounting means **making a file system accessible** to the OS.

- Example: Attaching a USB to /mnt/usb in Linux.
- **Mount Point:** Location where the file system is attached.
- Steps in Mounting:
    1. OS verifies the file system (e.g., via magic number).
    2. Registers it into the directory hierarchy.
    3. Makes it accessible via paths.

Auto-mounting: OS automatically mounts when the device is plugged in.

◆ **Protection Methods**

To prevent unauthorized access:

- **Access Control List (ACL):** Specifies users and their permissions (read, write, execute).
- **User IDs / Group IDs:** UNIX-style protection.
    - Owner
    - Group
    - Others
- **Encryption:** Data and file names are encrypted.
- **Read-Only Mount:** Used to prevent changes to the file system.

File System Structures and Implementation

- **Key Structures in a File System**
    - **Boot Control Block:** Contains boot loader code.
    - **Volume Control Block (Superblock):** Information about volume like number of blocks, block size, free space, etc.
    - **Directory Structure:** Organizes file names and their metadata.
    - **File Control Block (FCB):** Contains details like permissions, size, timestamps, disk block pointers.
- **Implementation Concepts**
    - **Contiguous Allocation:** Stores files in consecutive blocks. Fast, but may cause fragmentation.
    - **Linked Allocation:** Each file is a linked list of blocks. No fragmentation, but random access is slow.
    - **Indexed Allocation:** Uses an index block for each file to keep track of all disk block pointers.
- **Free Space Management**
    - **Bitmap:** 0 for free, 1 for allocated.
    - **Linked List of Free Blocks**
    - **Counting:** Store number of free blocks and starting address.

<u>Secondary Storage Structure</u>

Secondary storage refers to **non-volatile storage** used to store data permanently. Examples include **hard disks, SSDs, USB drives**. Among these, **disks (HDDs)** are the most commonly used form of secondary storage in OS-level design.

Disk Structure

- **Physical Structure**

A traditional **hard disk drive (HDD)** consists of:

- **Platters:** Round, magnetized disks stacked vertically.
- **Tracks:** Concentric circles on a platter.
- **Sectors:** Subdivision of a track; typically 512 bytes or 4 KB per sector.
- **Cylinders:** Same track number across all platters.
- **Read/Write Head:** Moves across platters to access data.
- **Disk Arm:** Moves the head to different tracks.
- **Rotational Speed:** Measured in RPM (e.g., 5400, 7200).

- **Logical Structure**

From the OS point of view:

- Disks are seen as **array of logical blocks** (block numbers starting from 0).
- OS maps **logical blocks to physical sectors**.
- Disk formatting (low-level and high-level) prepares disks for use.

o Disk Scheduling and Management

Why Scheduling?

Disk I/O is **slow** compared to memory access. Optimizing the order of read/write requests can **improve performance**.

Disk Scheduling Algorithms

Algorithm	Description	Pros	Cons
FCFS (First Come First Serve)	Services requests in the order they arrive	Simple	Inefficient head movement
SSTF (Shortest Seek Time First)	Selects request closest to current head	Better performance than FCFS	May cause starvation
SCAN (Elevator)	Head moves from one end to another, serving requests along the way	Fairer than SSTF	Might wait longer for edge requests

Algorithm	Description	Pros	Cons
LOOK	Like SCAN, but reverses direction after last request	Avoids unnecessary travel	Slightly better than SCAN
C-SCAN (Circular SCAN)	Head moves in one direction only and jumps to start	Uniform wait time	Skips requests behind head
C-LOOK	Like C-SCAN, but jumps to closest request instead of start	Improved performance	May still have jumps

Disk Management

Includes:

- **Partitioning**: Divides disk into logical units (partitions).
- **Formatting**: Prepares partition with file system (e.g., NTFS, ext4).
- **Boot Block Management**: Stores bootloader info for OS booting.
- **Bad Block Management**: Isolates damaged sectors.

Swap-Space Management

What is Swap Space?

Swap space is a special area on disk used as **virtual memory** when **RAM is full**.

- The OS may move inactive memory pages from RAM to swap (called **paging**).
- Also used for **process suspension**, **hibernation**, and **overcommitment** of memory.

◆ **Swap-Space Allocation Methods**

Method	Description
Preallocated	A fixed-size area reserved during OS install
Dynamic	Created/expanded on demand, more flexible

Can be a **dedicated partition** (faster) or **swap file** (flexible).

- **Swap-Space vs File System**
    - Swap space avoids overhead of file system (faster).
    - OS bypasses file structure and reads/writes directly to blocks.
- **Swap-Space Management Tasks**
    - Keeping track of which pages are in swap
    - Handling page-in and page-out requests
    - Protecting against simultaneous access
    - Clearing out unused or stale pages

Here , detail explanation on topic File System Interface and Implementation , Secondary Storage Structure :-

https://harshrb2424.github.io/Jntuh-R22-Notes/public/resources/2nd%20Year/OS-Unit5.pdf

Session 6: (2T+6L)

Linux History and Operation

o The Evolution of Linux

Background

- In the 1980s, **UNIX** was a popular operating system in universities and companies, but it was **proprietary**.
- In 1983, **Richard Stallman** launched the **GNU Project** to create a free UNIX-like OS.
- By the early 1990s, GNU had many components, but it lacked a **working kernel**.

Birth of Linux

- In 1991, **Linus Torvalds**, a Finnish computer science student, started developing a **new kernel**.
- He released **Linux 0.01** in 1991 under his own license, and shortly after under the **GNU General Public License (GPL)**.
- Combined with GNU tools, it became a **complete free operating system**.

Major Milestones

Year	Event
1991	Linux 0.01 released (first version)
1992	Licensed under GNU GPL
1994	Linux 1.0 released (first stable release)
1996	Linux 2.0 introduced support for multiprocessor systems
2003	Linux 2.6 released with improved scalability
2011	Linux 3.x kernel series started
2020+	Linux 5.x series continues with massive performance and hardware support

- **Linux Today**
    - Powers **servers**, **Android phones**, **cloud systems**, **supercomputers**, **IoT devices**.
    - Key part of **DevOps**, **machine learning**, and **embedded systems**.

o Linux Operations as a Server

Linux is widely used in **server environments** due to stability, security, and open-source nature.

- **Why Linux for Servers?**
    - **Open-source:** Free to use and modify.
    - **Stability:** Can run for years without reboot.
    - **Security:** Strong permissions, firewall (iptables), SELinux.
    - **Scalability:** Handles light to enterprise-level workloads.
    - **Support for major server technologies:** Apache, Nginx, MySQL, Docker, Kubernetes, etc.

Server Roles Linux Handles

Server Type	Description
Web Server	Hosts websites using Apache, Nginx
Database Server	Runs MySQL, PostgreSQL, MongoDB

Server Type	Description
File Server	Shares files via NFS, Samba
Email Server	Manages emails using Postfix, Sendmail
DNS Server	Resolves domain names using BIND
Virtualization Host	Runs VMs via KVM, QEMU
Container Host	Uses Docker, Podman, Kubernetes

The Architecture and Structure of Linux

Linux follows a **modular architecture** based on the traditional UNIX model.

- **Main Components**

1. Kernel Space

- **Monolithic kernel**: Includes file system, memory management, device drivers, networking.
- **Modular Design**: Kernel modules can be added/removed dynamically (insmod, rmmod).

2. User Space

- Programs and system libraries running outside the kernel.
- Communicates with kernel via **system calls**.

3. System Libraries

- Provide common functions (e.g., libc) used by applications.
- Abstract kernel functionalities.

4. System Utilities

- Basic tools and commands (ls, cp, ps) for managing the OS.

5. Shell

- Command-line interface (e.g., bash, zsh) where users interact with the OS.
- Acts as an intermediary between the user and kernel.

6. File System Hierarchy

- Everything in Linux is a **file**.
- Organized under a single root (/) with standard directories:

/bin – essential binaries

/etc – config files

/home – user home directories

/var – logs, variable data

/tmp – temporary files

/usr – user programs and libraries

/dev – device files

/proc – kernel and process info (virtual fs)

Basic Commands

(ls, cp, mv, sort, grep, cat,head,tail, man, locate, find, diff, file, rm, mkdir, rmdir,
cd, pwd, ln and ln – s, gzip and gunzip, zip and unzip, tar an its variants,
touch, echo, who, whoami, ps, kill,makefile,etc.)

File and Directory Commands

Command	Description	Example
ls	Lists files and directories	ls -l (long format), ls -a (includes hidden)
cp	Copies files or directories	cp file1 file2 (copy file1 to file2), cp -r dir1 dir2
mv	Moves or renames files/directories	mv file1 dir/ or mv oldname newname
rm	Removes files or directories	rm file.txt, rm -r dir/
mkdir	Creates a new directory	mkdir newfolder
rmdir	Removes an empty directory	rmdir emptyfolder
cd	Changes directory	cd /home/user
pwd	Prints current working directory	pwd shows e.g. /home/user
touch	Creates empty file or updates timestamp	touch new.txt
echo	Prints text to console or file	echo "Hello" > file.txt

File Viewing and Content Handling

Command	Description	Example
cat	Displays content of files	cat file.txt
head	Shows first lines of a file	head -n 5 file.txt
tail	Shows last lines of a file	tail -n 5 file.txt
sort	Sorts lines in a file	sort file.txt
grep	Searches text using patterns	grep "error" logfile.txt
diff	Compares two files line by line	diff file1.txt file2.txt
file	Tells file type	file filename (e.g., ASCII text, shell script)

Help and Documentation

Command	Description	Example
man	Opens manual for a command	man ls
locate	Finds files quickly using a database	locate file.txt
find	Searches files in real-time	find /home -name file.txt

Links and Shortcuts

Command	Description	Example
ln	Creates a **hard link**	ln file1 file2
ln -s	Creates a **symbolic (soft) link**	ln -s /path/to/file linkname

Compression Utilities

Command	Description	Example
gzip	Compresses files	gzip file.txt → creates file.txt.gz
gunzip	Decompresses .gz files	gunzip file.txt.gz
zip	Compresses into .zip format	zip archive.zip file1 file2
unzip	Extracts .zip files	unzip archive.zip

Command	Description	Example
tar	Archive utility. Often used with gzip or bzip2	
tar -cvf	Create archive	tar -cvf files.tar file1 file2
tar -xvf	Extract archive	tar -xvf files.tar
tar -czvf	Create and compress	tar -czvf files.tar.gz dir/
tar -xzvf	Extract .tar.gz	tar -xzvf archive.tar.gz

User and Process Management

Command	Description	Example
who	Shows users currently logged in	who
whoami	Displays current username	whoami
ps	Shows running processes	ps aux, ps -ef
kill	Terminates a process	kill 1234 (PID), kill -9 1234 (force kill)

Other Useful Commands

Command	Description	Example
makefile	Used to build software from source code via make utility	Contains rules for compiling (e.g., C programs)
make	Executes instructions from a Makefile	make (in source code directory)

Access control list and chmod command

What is ACL?

ACLs provide a **fine-grained permission mechanism** in Linux. They allow you to set **specific permissions** (read, write, execute) for **multiple users or groups** on a **single file or directory**, beyond the standard **owner/group/others** model.

Why Use ACL?

- Traditional Linux permissions allow only 3 entities: Owner, Group, Others.
- ACLs allow **specific access** to multiple users or groups **independently**.

How to Use ACL?

Enable ACL (If not already enabled):

mount -o remount,acl /mount_point

View ACL of a file:

getfacl filename

Add ACL to a file:

setfacl -m u:username:rwx filename # For a user

setfacl -m g:groupname:rw filename # For a group

Example:

touch demo.txt

setfacl -m u:john:rw demo.txt

getfacl demo.txt

Output:

file: demo.txt

owner: root

group: root

user::rw-

user:john:rw-

group::r--

mask::rw-

other::r—

chown and chgrp commands

Purpose:

chown is used to change the ownership of a file or directory to a different user or group.

- Syntax:

  chown [OPTIONS] new_owner[:new_group] file

- Examples:

  chown mahima file.txt

  Changes the owner of file.txt to mahima.

  chown mahima:developers file.txt

  Changes the owner to mahima and the group to developers.

chown -R mahima:users /home/mahima

Changes ownership of all files and subdirectories.

Commands like telnet, ftp, ssh, and sftp

Telnet (TELecommunication NETwork)

- **Purpose:**
  Used to remotely access another computer over a TCP/IP network (like the Internet).
  Port: 23 (default)

  Command Example:
  telnet hostname port

  Use Case:
  - Accessing remote servers or network devices (e.g., routers)
  - Testing if a port is open (e.g., telnet google.com 80)

  Limitations:
  - Not secure: Data (including passwords) is transmitted in **plain text**
  - Mostly replaced by SSH

FTP (File Transfer Protocol)

- **Purpose:**

Used to transfer files between client and server.

Port:

- 21 (control)
- 20 (data, in active mode)

Command Example:

ftp ftp.example.com

Commands Inside FTP:

```
ls       # List files
cd dir   # Change directory
get file # Download file
put file # Upload file
bye      # Exit
```

Limitations:

- Not secure (data and credentials sent in plain text)
- Can be secured using FTPS (FTP over SSL)

SSH (Secure Shell)

- **Purpose:**
  Secure remote login and command execution.

Port: 22 (default)

Command Example:

ssh username@hostname

Features:

- Encrypts all data
- Can tunnel ports, forward X11, copy files (using scp)

Use Case:

- Managing servers securely
- Admin tasks and file transfer via scp or rsync

SFTP (SSH File Transfer Protocol)

- **Purpose:**
  Securely transfer files over an encrypted SSH connection

Port: 22 (same as SSH)

Command Example:

sftp username@hostname

Commands Inside SFTP:

```
ls        # List files
cd dir    # Change directory
get file  # Download
put file  # Upload
exit      # Quit
```

Advantages over FTP:

- Encrypted
- More firewall-friendly (uses single port)

Basics of I/O System in Linux

What is an I/O System?

I/O (Input/Output) system in Linux is responsible for:

- Communicating with hardware devices (disks, keyboards, printers, USB, etc.)
- Reading and writing data from/to storage
- Providing a file-system abstraction for devices

Everything is a File in Linux

In Linux:

- Devices are treated as files and are found under the /dev directory
  e.g., /dev/sda → first hard disk, /dev/tty → terminal

Mount and Unmount in Linux

Linux does not automatically assign a drive letter (like Windows).
Instead, it mounts storage devices (USB, partitions, CD-ROMs) to a directory in the file system tree.

What is mount?

mount command attaches a filesystem (like a USB drive or partition) to a directory (called a *mount point*).

Syntax :- sudo mount [device] [mount_point]

Example :- sudo mount /dev/sdb1 /mnt/usb

- /dev/sdb1 is the USB device
- /mnt/usb is the mount point

Viewing Mounted Devices:

mount

What is umount?

umount detaches the filesystem from the directory tree.

Syntax: sudo umount [device or mount_point]

Example :- sudo umount /mnt/usb

Note: You **must not** be inside the mount directory while unmounting. If you are, you'll get a "device is busy" error.

Important Mount-Related Files

File	Purpose
/etc/fstab	Lists filesystems that should be auto-mounted at boot
/proc/mounts	Shows currently mounted filesystems

Example Workflow: Mounting a USB Drive Manually

```
# Create a mount point
sudo mkdir /mnt/usb

# Mount the USB drive
sudo mount /dev/sdb1 /mnt/usb

# Access files
cd /mnt/usb

# After you're done
cd ~
sudo umount /mnt/usb
```

- vi editor

vi Editor Overview

vi (short for **Visual Editor**) is a powerful, lightweight **text editor** available by default in almost all Unix/Linux systems.

Features of vi Editor

- Available on all Unix/Linux systems

- Lightweight and fast
- Works in terminal
- Powerful with many built-in commands
- Supports syntax highlighting (in vim version)
- Can edit large files efficiently

o Editing using vi editor

Modes in vi Editor

vi works in **three primary modes**:

Mode	**Purpose**
1. Normal Mode	Default mode (navigate, delete, copy, paste, etc.)
2. Insert Mode	For inserting/editing text
3. Command Mode	For running commands (save, search, replace, etc.)

- Switching Between Modes:

Press Key	**Action**
i	Enter Insert mode (before cursor)
a	Insert after cursor
Esc	Return to Normal mode
:	Enter Command mode

Opening and Editing a File

Open a file:

vi filename.txt

Basic Editing:

Command	**Meaning**
i	Insert before cursor
a	Append after cursor
o	Open a new line below

Command	Meaning
Esc	Exit Insert mode
:w	Save (Write) file
:q	Quit
:wq or ZZ	Save and quit
:q!	Quit without saving

- **Find and Replace in vi**

  Syntax:

  sql

  :[range]s/old/new/[flags]

  Examples:

  - Replace **first occurrence** of "cat" with "dog" in the current line:

    :s/cat/dog/

  - Replace **all occurrences** in the current line:

    :s/cat/dog/g

  - Replace in the **entire file**:

    :%s/cat/dog/g

    :%s/cat/dog/gi

- **Cut, Copy, and Paste in vi**

Action	Command
Cut a line	dd (delete line)
Copy a line	yy (yank line)
Paste below	p
Paste above	P
Delete word	dw
Delete char	x

Action	Command
Undo	u
Redo	Ctrl + r

You must be in **Normal mode** (Esc) to use these.

- **The :set Command**

  The :set command in vi/vim is used to change editor settings.

  Common :set options:

Command	Description
:set number	Show line numbers
:set nonumber	Hide line numbers
:set ignorecase	Search ignoring case
:set hlsearch	Highlight search results
:set tabstop=4	Set tab width to 4 spaces
:set autoindent	Auto-indents new lines
:set paste	Enables paste mode (avoids auto-indent mess)

Show current settings:

```
:set all
```

Introduction to Users and Groups in Linux

Linux is a multi-user operating system, meaning multiple users can access and use the system resources independently at the same time. To manage this, Linux uses users and groups for:

- Access control
- File ownership
- Permission management

Users in Linux

Types of Users:

User Type	Description
Root	Superuser with full system access (UID 0)
System Users	Used for background processes and services (like mysql, www-data)
Normal Users	Created by the administrator or users themselves

User Files:

File	Purpose
/etc/passwd	Stores user account information
/etc/shadow	Stores encrypted passwords
/etc/login.defs	Default settings for new users

View Current User: whoami

Switch User: su username

Create New User: sudo adduser newusername

Delete User: sudo deluser username

Groups in Linux

Groups are collections of users. They allow the administrator to assign permissions to **multiple users at once**.

Types of Groups:

Group Type	Description
Primary Group	Assigned to each user (default group)
Secondary Group	Additional groups a user belongs to

Group Files:

File	Purpose
/etc/group	Stores group info (group name, GID, members)

File	Purpose
/etc/gshadow	Encrypted group passwords

Create Group: sudo groupadd groupname

Add User to Group: sudo usermod -aG groupname username

View Groups: groups username

User & Group IDs

Term	Description
UID	User ID (each user has one)
GID	Group ID (each group has one)

System users usually have **UIDs below 1000**, and normal users have **UIDs 1000+**.

Permissions Based on Users and Groups

Every file and directory in Linux has 3 types of ownership:

Ownership Type	Description
User (u)	The owner of the file
Group (g)	The group assigned to the file
Others (o)	Everyone else

Use ls -l to see ownership and permissions: ls -l file.txt

Example : -rw-r--r-- 1 mahima devs 1234 Jun 20 test.txt

- **Owner:** mahima
- **Group:** devs
- **Permissions:** Owner can read/write, Group & Others can only read

Changing Ownership & Groups

Change Owner: sudo chown newuser file.txt

Change Group: sudo chgrp newgroup file.txt

Change Both: sudo chown newuser:newgroup file.txt