5. **Start System Services**
    - Executes all **startup scripts** in the appropriate order:
        - Network services
        - Daemons (background services)
        - Logging services
        - GUI services (if graphical runlevel)
6. **Monitor Processes**
    - init continuously monitors **critical system processes**.
    - If a process dies and is configured to restart, init respawns it.
7. **Handle Runlevel Changes**
    - If the system receives a request to change runlevel (via telinit or init command):
        - Executes shutdown scripts for the current runlevel
        - Executes startup scripts for the new runlevel
        - Switches to the new runlevel
8. **Shutdown / Reboot**
    - If runlevel 0 (halt) or 6 (reboot) is selected:
        - Stops all services in the correct order
        - Unmounts filesystems
        - Powers off or reboots the system

UNIT-IV Programming With Shell

16 Marks

1. Describe Shell Script in UNIX.

- A **Shell Script** is a text file containing a sequence of UNIX commands, programming constructs (like variables, loops, conditionals), and instructions that the **shell interpreter** executes line by line.
- Instead of typing commands repeatedly in the terminal, a shell script automates the task.

Shebang (#!)

- Every shell script usually starts with a line like:
- `#!/bin/bash`
- This tells the operating system which shell should interpret the script.
- Common shells: **sh(**Bourne Shell), **bash(**Bourne Again Shell), **csh(**C Shell), **ksh(**Korn Shell), **zsh(**Z Shell).

Features of Shells

1. **Automates repetitive tasks**
   - Shells allow users to write **scripts** that perform multiple commands automatically.
   - This is useful for tasks like **backups, file management, or system monitoring**, which would be tedious if done manually.
   - Example: A script that deletes temporary files every night saves time and ensures consistency.
2. **Easy to write and execute**
   - Shell scripts are **simple text files** containing commands.
   - No compilation is required; they can be executed directly in the shell using `sh scriptname.sh` or `bash scriptname.sh`.
   - Even beginners can quickly learn to write useful scripts.
3. **Can use all UNIX commands inside it**
   - A shell acts as an **interface between the user and the OS**.
   - Any command that works in the terminal (like `ls`, `grep`, `awk`, `sed`) can be used inside a script.
   - This allows combining multiple commands for complex tasks efficiently.
4. **Provides programming features (variables, loops, conditions, functions)**
   - Shells support **variables** to store data, **loops** for repetitive operations, **conditional statements** (`if`, `case`) for decision-making, and **functions** to organize code.
   - Example: Using a `for` loop to process all files in a directory automatically.
5. **Portable (runs on most UNIX/Linux systems)**
   - Shell scripts are **highly portable**; the same script often runs on different UNIX/Linux systems without changes.
   - Only minor adjustments may be needed if system-specific commands are used.
6. **Reduces manual errors**
   - By automating tasks, shell scripts **minimize human mistakes**.
   - Example: A script for bulk renaming files ensures all names follow a specific pattern without manual typing errors.

Example:

Variables in Shell Script

Variables in shell scripting are used to **store data** that can be reused throughout the script. Unlike other programming languages, **shell variables do not require explicit declaration of type**; they are **loosely typed**.

```
#!/bin/bash

name="John"

age=22

echo "My name is $name and I am $age years old."
```

OUTPUT:

My name is Anjalin and I am 22 years old.

Control statements

Control statements in a shell script are used to **control the flow of execution** of commands based on certain conditions or loops. They allow the script to make decisions, repeat tasks, or execute specific sections of code depending on conditions.

If-Else Example:

```
#!/bin/bash

echo "Enter a number:"

read num

if [ $num -gt 0 ]

then

  echo "Positive"

else

  echo "Non-Positive"

fi
```

```
OUTPUT:

Enter a number:

5

Positive

Enter a number:

-3

Non-Positive
```

Loop Example:

```
#!/bin/bash

for i in 1 2 3 4 5

do

  echo "Number: $i"

done
```

```
OUTPUT:

Number: 1

Number: 2

Number: 3

Number: 4

Number: 5
```

2. Command Line Arguments and positional parameters in Shell Script

Command Line Arguments

- **Command Line Arguments** are parameters passed to a shell script when it is executed from the command line.
- These allow users to provide input to the script **without using read statements**.
- Example of execution: **./script.sh arg1 arg2 arg3**

- Here, arg1, arg2, and arg3 are command line arguments.

Positional Parameters

- **Positional Parameters** are special variables in a shell script that hold the **command-line arguments** passed to the script.
- They are called *positional* because each variable represents an argument based on its **position**.

Variable	Description
`$0`	Name of the script itself
`$1`	First argument
`$2`	Second argument
`$3…$9`	Third to ninth argument
`$#`	Number of arguments passed
`$@`	All arguments as a list (`"$1" "$2" ...`)
`$*`	All arguments as a single string
`$$`	Process ID of the script
`$?`	Exit status of last command

Example 1 – Simple Addition

```
#!/bin/bash
num1=$1
num2=$2
sum=$((num1 + num2))
echo "Sum of $num1 and $num2 is: $sum"
```

Execution:

./add.sh 10 20

Output:

Sum of 10 and 20 is: 30

Example 2 – Display Arguments

```
#!/bin/bash
echo "Script name: $0"
echo "First argument: $1"
echo "Second argument: $2"
echo "All arguments: $@"
echo "Number of arguments: $#"
```

Execution:

./example.sh apple banana cherry

Output:

Script name: ./example.sh

First argument: apple

Second argument: banana

All arguments: apple banana cherry

Number of arguments: 3

Shifting Arguments

- **shift** command is used to move arguments to the left.
- $1 takes the value of $2, $2 takes the value of $3, and so on.
- Useful for processing an unknown number of arguments.

```
#!/bin/bash

while [ $# -gt 0 ]

do

  echo "Argument: $1"

  shift

done
```

Execution:

./example.sh a b c d

Output:

```
Argument: a

Argument: b

Argument: c

Argument: d
```

Advantages:

- Allows **dynamic input** without hardcoding.
- Supports **automation** and batch processing.
- **Reusable** for different inputs.
- **Efficient**, faster than interactive input.

Disadvantages:

- **Errors** if arguments are missing or in wrong order.
- **Limited** to positional numbers (beyond `$9` needs extra handling).
- Requires **validation** for correct input.
- Less **user-friendly** for beginners.

3.Explain File System Architecture with example.

- A **File System** is a way the operating system **stores, organizes, manages, and retrieves files** on storage devices.
- It provides users a **logical view of data** while managing the **physical storage details** internally.
- In UNIX, the file system is **hierarchical**, allowing files and directories to be organized like a tree.

Objectives of a File System

- Efficient storage and retrieval of files.
- Support for multiple users and processes simultaneously.
- Maintenance of **data integrity**, preventing accidental loss.
- Security and access control through **permissions and ownership**.
- Efficient management of **free space** and disk blocks.
- Ease of backup, recovery, and system administration.

Components of UNIX File System:

(A) File

- Collection of related data stored on disk.
- Types in UNIX:
    1. **Regular files** – text, binary, executable.
    2. **Directory files** – contain references to other files.
    3. **Special files** – device files:
        - Character devices (/dev/tty)
        - Block devices (/dev/sda1)

(B) Directory

- Organizes files in a **hierarchical structure**.
- Root directory / is the top-level.
- **Directories can contain subdirectories and files.**

(C) File Control Block (FCB) / Inode

- **Inode** stores **metadata** about a file:
    - File type
    - Owner and permissions
    - File size
    - Timestamps (created, modified, accessed)
    - Pointers to data blocks

(D) Data Blocks

- Actual **physical storage** of file contents on disk.
- Managed by the **kernel**, allocated as needed.

(E) Superblock

- Stores information about the **file system structure**:
    - Total size, free blocks, number of inodes
    - File system state (clean/dirty)

(F) File System Interface

- Allows **user programs** to interact with files using system calls:
  - open(), read(), write(), close()

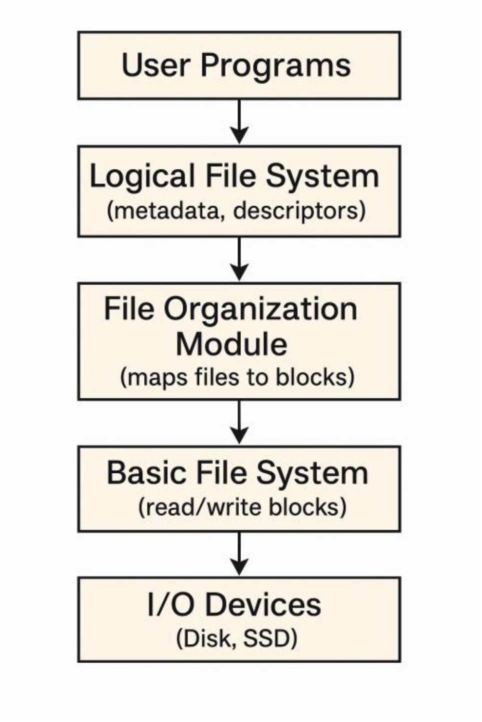
(G) Kernel File System / I/O Control

- Handles **disk operations, buffering, caching**, and ensures efficient access.

Layered File System Architecture:

UNIX uses a **layered approach** to separate **logical view** from **physical storage**:

1. **User Level**
   - Users access files via commands (ls, cat, cp) or programs.
2. **Logical File System (LFS)**
   - Maintains **metadata** and file descriptors.
   - Provides **logical view** to user programs.
3. **File Organization Module (FOM)**
   - Maps logical files to **physical blocks**.
   - Handles allocation strategies (contiguous, linked, indexed).
4. **Basic File System (BFS)**
   - Manages **low-level read/write** operations on disk.
   - Communicates directly with the disk driver.
5. **I/O Hardware**
   - Physical storage devices (HDD, SSD, USB)



File Access Methods

1. **Sequential Access** – Read data **in order**, like text files.
2. **Direct / Random Access** – Jump to any position, used in databases.
3. **Indexed Access** – Uses an **index table** for faster access to blocks.

Advantages

- Efficient storage and retrieval of files.
- Supports multiuser and multitasking.
- **Hierarchical directory structure** simplifies file organization.
- Flexible allocation strategies for data blocks.
- Layered design allows **modularity and portability**.
- Provides **security** with permissions and ownership.

Disadvantages / Limitations

- Managing very large file systems can be **complex**.
- Performance may degrade if inode tables or directories are too large.
- Requires careful **disk management and backup strategies**.