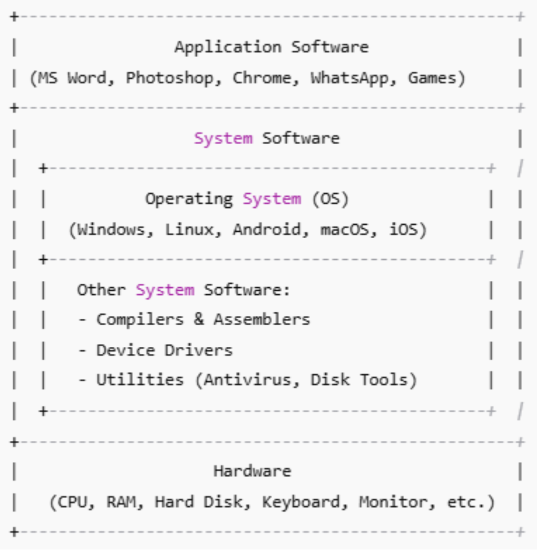
1. **Explain the term *Operating System* in your own words. How does it differ from system software and application software? Give suitable examples to support your answer.**



```
+------------------------------------------------------+
|                 Application Software                 |
| (MS Word, Photoshop, Chrome, WhatsApp, Games)        |
+------------------------------------------------------+
|                   System Software                    |
|  +------------------------------------------------+  |
|  |            Operating System (OS)               |  |
|  |  (Windows, Linux, Android, macOS, iOS)         |  |
|  +------------------------------------------------+  |
|  |   Other System Software:                       |  |
|  |   - Compilers & Assemblers                     |  |
|  |   - Device Drivers                             |  |
|  |   - Utilities (Antivirus, Disk Tools)          |  |
|  +------------------------------------------------+  |
+------------------------------------------------------+
|                       Hardware                       |
|  (CPU, RAM, Hard Disk, Keyboard, Monitor, etc.)      |
+------------------------------------------------------+
```

Ans. "An Operating System is the most important type of System Software that manages hardware resources such as CPU, memory, and devices, and also provides a platform for application software, while other system software (like compilers, drivers, utilities) support its functioning."

- All Operating System are system software but not all system software is OS.
- OS is the interface b/w User and Hardware.
- Examples of System Software are Compilers, Assemblers, Device Drivers, Tools etc.

It is different from other system software and application software in the following ways:

Aspect	Operating System (OS)	Other System Software	Application Software
Meaning	A system program that controls hardware and acts as an interface between user and computer.	Supporting programs that help in controlling, translating, or maintaining system operations.	Programs developed to perform specific tasks for the user.
Role	Manages CPU, memory, devices, and files.	Provides tools or services to support OS and applications.	Provides solutions to user needs like writing, browsing, or editing.
User Interaction	Indirectly interacts with user through interface.	Does not directly interact with user (works in background).	Directly interacts with user.
Examples	Windows, Linux, macOS, Android.	Compilers, Assemblers, Device Drivers, Utilities (Antivirus).	MS Word, Photoshop, Chrome, WhatsApp, Games.

For example, **Windows and Linux** are operating systems, **compilers and antivirus tools** are system software, and **MS Word, Photoshop, and Chrome** are application software.

2. Consider a mobile phone and a desktop computer. Discuss how the role of the operating system differs in these two environments. Give at least two examples of features unique to each environment.

Ans.

Aspect	Mobile Phone OS	Desktop Computer OS
Primary Role	The OS manages **touchscreen input**, **battery usage**, **mobile sensors** (GPS, camera, accelerometer), and **mobile network connectivity**.	The OS manages **keyboard/mouse input**, **larger memory/CPU**, multiple peripherals, and supports a wider range of software.
Optimization Focus	It is optimized for **power efficiency**, multitasking with limited resources, and running apps from app stores.	It is optimized for **high-performance multitasking** and handling heavy applications.
Unique Feature 1	Power/Battery Management – limits background apps to save battery.	File System & Disk Management – manages hard drives and external storage
Unique Feature 2	Mobile Network & SIM Management – handles calls, SMS, 4G/5G connectivity	Peripheral Management – handles printers, scanners, external drives, multiple monitors

3. An operating system is often described as a *resource manager*. Analyse this statement by discussing how an OS manages CPU, memory, and process when multiple applications are running simultaneously. Use a real-world analogy to strengthen your explanation.

Ans.

An **Operating System (OS)** is often called a **resource manager** because it efficiently allocates and controls the computer's resources—**CPU, memory, and processes**—so that multiple applications can run smoothly at the same time.

a) CPU Management

- The OS decides which program gets to use the CPU and for how long.
- It uses methods like **round-robin** (taking turns) or **priority scheduling** (important tasks first).
- This makes sure all programs run smoothly and no program block the CPU forever.

b) Memory Management

- The OS provides each program its own memory space in memory (RAM), so programs do not interfere with each other while running.
- If RAM is full, the OS can **move data temporarily to disk (virtual memory)** and bring it back when needed.
- This ensures **programs have enough memory to run** without crashing.

c) Process Management

- The OS keeps a **list of all programs (processes) running** and their status: ready, running, or waiting.
- It handles **creation, execution, and termination** of processes, ensuring smooth multitasking without crashes.

Real-World Analogy: Restaurant Kitchen

- **CPU = Chef** → Prepares dishes one at a time but quickly switches between them to serve multiple orders efficiently.
- **Memory = Kitchen counter/storage** → Holds ingredients for each dish, so the chef can access them quickly when needed.
- **Processes = Orders** → Each dish represents a program/process; the kitchen manager (OS) ensures all orders are prepared correctly and served on time without mix-ups.

4. A computer system is more than just hardware. Explain its essential characteristics and how system programs enhance its functionality.

Ans.

A computer system is **more than just hardware**; it also includes software, data, and users. Its essential characteristics are:

A) Integration of Hardware and Software

- Hardware provides physical components like CPU, memory, and storage.
- Software (system programs and applications) enables the hardware to perform useful tasks.

B) Automation

- Computers execute instructions automatically once a program is provided, reducing the need for constant human intervention.

C) Speed and Accuracy

- Computers process data and perform calculations much faster than humans, with high precision.

D) Storage and Retrieval

- They can store large amounts of data and quickly retrieve it when needed.

E) Multitasking

- Computers can run multiple programs simultaneously by efficiently managing CPU, memory, and processes.

Role of System Programs

System programs (or system software) **enhance the functionality of the computer** by acting as intermediaries between the hardware and user applications:

- **Operating System (OS):** Manages hardware resources and allows applications to run.
- **Compilers and Interpreters:** Translate user-written programs into machine language.
- **Utilities:** Perform tasks like file management, virus scanning, and disk cleanup.

In short: System programs make the hardware **usable, efficient, and safe**, enabling users and applications to interact with the computer effectively.

5. Differentiate between system programs and application programs with examples. Why are system programs considered the "foundation" of software execution?

Ans.

Feature	System Programs	Application Programs
Definition	Software that manages, supports, or controls computer hardware and system operations.	Software designed to help the user perform specific tasks.
Purpose	To make the computer hardware usable and efficient.	To solve user-specific problems or provide entertainment/productivity.
Interaction	Works mostly **behind the scenes**, indirectly with the user.	Directly interacts with the user.
Dependency	Applications rely on system programs to function.	Runs on top of system programs (cannot work without them).
Examples	Operating System (Windows, Linux), Compilers, Device Drivers, Utilities (Antivirus, Disk Tools)	MS Word, Photoshop, Chrome, WhatsApp, Games

Why System Programs are the "Foundation" of Software Execution

- System programs, especially the **Operating System**, provide the **basic environment** for running all other software.
- They **manage hardware resources** (CPU, memory, storage, devices) so that applications can run efficiently and safely.
- Without system programs, **applications cannot execute**, because they would have no way to communicate with hardware or perform tasks.

6. **If utility programs (backup, compression, virus scanners) were absent, how would system reliability and usability be affected?**

Ans.

Utility Program	Role	Impact if Absent
Backup	Saves and restores data	Data loss becomes permanent; system reliability drops
Compression	Reduces file size for storage/transfer	Disk space fills quickly; file transfer slows
Virus Scanner	Protects against malware and attacks	System vulnerable to viruses; performance and security decrease

- ➢ Utility programs **enhance system reliability, security, and efficiency**.
- ➢ Without them, computers would be **less safe, slower, and harder to use**, even if the OS and applications are working.

7. **Modern computing cannot function efficiently without an operating system. Justify this by discussing its role in resource sharing and hardware management.**

Ans.

Modern Computing Cannot Function Efficiently Without an OS because:

A) **Resource Sharing**

- Modern computers run **multiple applications simultaneously**.
- The OS ensures **fair and efficient allocation of CPU, memory, and storage**, so each program gets the resources it needs.
- It prevents conflicts between programs and enables **smooth multitasking**.

B) **Hardware Management**

- The OS acts as a **bridge between software and hardware**, managing all devices efficiently.
- It handles **CPU scheduling, memory allocation, and input/output operations** for keyboards, printers, storage, and networks.
- Without the OS, programs would need to control hardware directly, which is **complex, error-prone, and inefficient**.

8. **Discuss why kernel runs in privileged mode, while shell runs in user mode. What problems might occur if this distinction did not exist?**

Ans.

- **Kernel**: The core part of the Operating System. It directly controls hardware (CPU, memory, disks, input/output devices).
- **Shell**: A command interpreter (like a bridge between user and kernel). It takes user commands (like ls, mkdir, etc.) and asks the kernel to execute them.

❖ **Why Kernel runs in Privileged Mode –**

- The **kernel must have full control** of hardware resources.
- Privileged mode (also called **kernel mode**) allows it to:
  - Access hardware directly (CPU instructions, memory, disk, etc.)
  - Manage processes, memory, file system.
  - Ensure security and stability of the system.

❖ **Why Shell runs in User Mode –**

- The shell is just a program run by the user.
- It doesn't directly access hardware; instead, it **requests the kernel** to perform tasks.
- Running in user mode ensures that if the shell crashes, it won't harm the whole system.
- **Example:** If your terminal freezes, you can just close it — your computer still runs fine.

❖ **What if both ran in the same mode –**

1) **Accidental Damage**
   - A small typing mistake could delete system files or overwrite memory.
   - Example: Wrong command might crash the whole OS.
2) **Security Risk**
   - Any user program (like shell) could directly access hardware and kernel memory.
   - Hackers could steal passwords, files, or take full system control.
3) **System Instability**
   - If the shell (or any user program) crashes, the whole operating system would also crash.
   - No protection between user errors and kernel core.
4) **No Resource Protection**
   - Any program could directly control CPU, memory, or I/O devices.
   - One program could use up all resources and block others.
5) **Data Loss**
   - Wrong disk command could format the hard drive or corrupt files permanently.
   - No safeguard layer to prevent damage.
6) **Difficult to Maintain**
   - Debugging and fixing errors would be very hard because every program would have full control.
   - No clear separation of responsibilities.

9. The kernel is the "core" of the OS. Explain its role in process management, memory allocation, and device handling.

Ans.

- The **kernel** is the most important part of the operating system.
- It runs in **privileged mode** and acts as a **bridge between hardware and software**.
- Its main roles are in **Process Management, Memory Management, and Device Handling**.
- The **kernel** is like the **manager of the whole computer**. It directly controls CPU, memory, and devices. Without the kernel, the system cannot manage CPU, memory, and devices.

A) Process Management:

- Creates and terminates processes.
- Schedules processes to decide which one gets CPU time.
- Supports multitasking so multiple programs run together.
- Performs context switching to quickly switch between processes.
- Provides inter-process communication (IPC) for safe data sharing.
- Handles deadlocks to avoid processes blocking each other.

B) Memory Allocation:

- Allocates and deallocates RAM to processes.
- Protects memory so one process cannot access another's space.
- Uses virtual memory to extend RAM using disk space.
- Efficiently uses memory through paging and segmentation.
- Translates logical addresses into physical addresses.

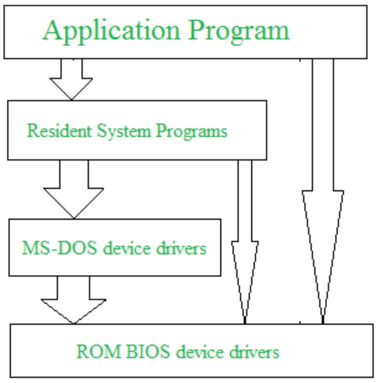
C) Device Handling:

- Uses device drivers to communicate with hardware like keyboard, mouse, printer, and disk.
- Performs buffering and caching to speed up I/O.
- Schedules device access when many processes request the same device.
- Provides device independence so applications don't need to know hardware details.
- Handles device errors like printer jams or disk read failures.

10. A monolithic OS has all functions tightly integrated. Discuss advantages and drawbacks of this structure with an example.

Ans.

- A **monolithic OS** is a type of operating system where *all functions (process management, memory, device drivers, file system, etc.) are tightly integrated into a single large kernel.*
- All components can directly communicate with each other inside the kernel.
- **Example: UNIX** and **MS-DOS** are classic monolithic operating systems.



❖ **Advantages of Monolithic OS –**

- **Fast performance** – since all services run in the same kernel space, communication between them is quick.
- **Simple design** – easier to design initially because everything is combined into one kernel.
- **Efficient resource usage** – no overhead of switching between user and kernel modules frequently.
- **Direct communication** – system calls are handled directly inside the kernel without extra layers.
- **Better hardware control** – since all device drivers are part of the kernel, hardware access is faster.

❖ **Drawbacks of Monolithic OS –**

- **Lack of modularity** – everything is tightly linked, so changing one part may affect the whole system.
- **Difficult to maintain** – fixing bugs or adding new features requires modifying the large kernel.
- **Less secure** – a single bug in any part (e.g., device driver) can crash the whole system.
- **Poor reliability** – if one service fails, the entire OS can fail (system crash or kernel panic).
- **Not flexible** – hard to extend for modern needs like networking, security, or new devices.

11. Modular OS structures improve maintainability and scalability. Explain how layers or modules simplify OS design.

Ans.

- A **modular (or layered) OS** organizes the operating system into separate independent layers or modules, each handling a specific function.
- Examples of modules: process management, memory management, file system, device drivers, networking, etc.

❖ **How layers/modules simplify OS design**

- **Separation of concerns** – each module is responsible for one function only (e.g., file system handles files, memory module handles RAM). This makes design cleaner.
- **Easier to maintain** – if one module has a bug (like device driver), you can fix or replace it without touching other modules.
- **Better scalability** – new features (like new file system, new device driver) can be added without redesigning the whole OS.
- **Improved security** – one faulty module cannot easily crash the entire OS because modules are isolated.
- **Testing is simpler** – you can test each module separately before integrating it.
- **Flexibility** – different modules can be upgraded or replaced independently (e.g., adding a new network protocol).
- **Reduced complexity** – programmers can focus on one module instead of the entire system.
- **Reliability** – failure in one layer (e.g., printer driver) won't bring down the whole OS.

12. Compare monolithic and modular OS structures in terms of reliability, flexibility, and performance.

Ans.

Feature	**Monolithic OS**	**Modular OS**
Reliability	Less reliable because a bug in one part (like a device driver) can crash the entire system	More reliable because each module is isolated; failure in one module usually does not affect others
Flexibility	Less flexible because all functions are tightly integrated; adding or updating features requires modifying the kernel	More flexible because modules can be added, removed, or upgraded independently
Performance	High performance because all functions run in the same kernel space; communication between components is fast	Slightly lower performance due to overhead of module interfaces and communication between separate modules, but still efficient

13. **Why do modern operating systems (like Windows, Linux) prefer modular or hybrid structures? Support your answer.**

Ans.

- Modern operating systems like **Windows and Linux** prefer **modular or hybrid structures** instead of purely monolithic designs.
- Modules are independent, so bugs in one module can be fixed without affecting the rest of the system, making the OS easier to maintain and upgrade.
- New features, drivers, or services can be added without modifying the whole OS.
  Example: Adding support for a new printer or network protocol only requires updating the relevant module.
- Isolated modules prevent a failure in one part from crashing the entire system, and security is improved because modules can be restricted to specific access levels.
- The OS can scale to support modern hardware, multiple processors, and complex applications by adding or upgrading modules independently.
- Each module can be tested separately before integration, making development more efficient and errors easier to locate.
- Hybrid approaches allow core functions to run in kernel mode for speed, while less critical services run as separate modules, balancing performance and safety.
  Example:
  - **Linux** uses loadable kernel modules: you can add a device driver without restarting the system.
  - **Windows NT/10** uses a hybrid structure: core kernel handles essential functions, and other services are modular.

14. **The Windows Executive is a major layer inside Kernel Mode. Discuss its components and how they support process, memory, and I/O management.**

Ans.

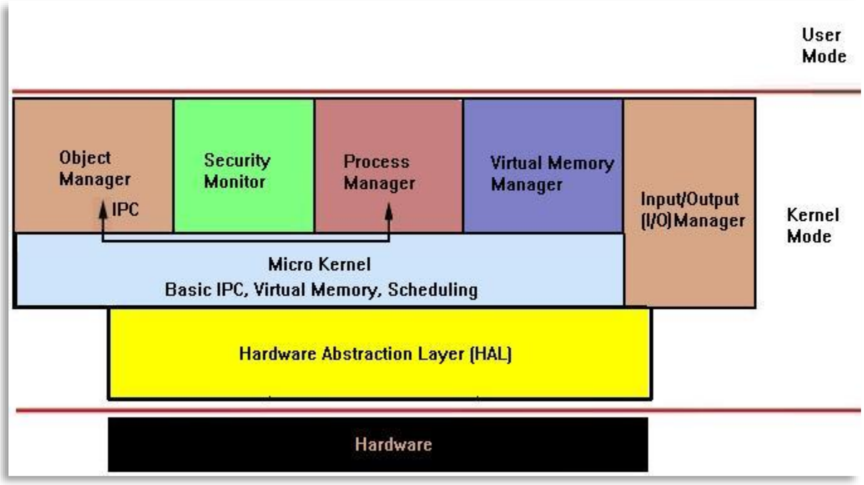
- The **Windows Executive** is a **major part of Windows running in kernel mode**.
- It sits above the core of the kernel and provides **important services** to manage programs, memory, and devices.
- It is **divided into modules** (different parts), and each part has a specific job –

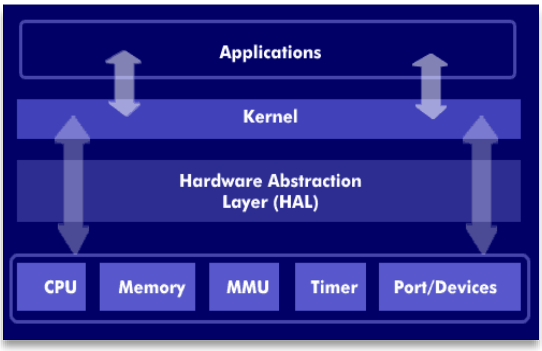
1) **Process Manager:**
   - Creates and deletes programs (processes) and decides which program uses the CPU and when.
   - Keeps track of all running programs.
2) **Memory Manager:**
   - Gives memory to programs and takes it back when done.
   - Uses virtual memory so programs can use more memory than the RAM.
   - Keeps programs' memory safe so one program can't touch another's memory.
3) **I/O Manager:**
   - Helps programs talk to devices like keyboard, mouse, printer, and disk.
   - Uses device drivers to control devices.
   - Makes I/O faster using buffering and caching.
   - Handles errors like a printer jam or disk problem.
4) **Object Manager:**
   - Treats files, devices, and programs as "objects" so they are easier to manage.
   - Helps control security and access to resources.
5) **Security Reference Monitor:**
   - Makes sure only authorized programs can access memory, files, and devices.
6) **Other helpers:**
   - **Cache Manager:** speeds up file access.
   - **Plug and Play Manager:** detects new hardware and makes it work automatically.
   - **Configuration Manager:** keeps system settings and registry information.

15. Compare the role of Hardware Abstraction Layer (HAL) with the Kernel. Why is HAL important in achieving portability across different hardware platforms?

Ans.

Aspect	Hardware Abstraction Layer (HAL)	Kernel
Definition	HAL is a software layer that provides an interface between the operating system and the underlying hardware.	Kernel is the core part of the operating system that manages hardware resources and system operations.
Main Function	Abstracts (hides) hardware-specific details and provides standard functions for hardware access.	Controls and manages system resources like CPU, memory, files, and devices.
Position in OS Architecture	Lies between the hardware and the kernel.	Lies above the HAL and interacts directly with it.
Hardware Dependency	Hardware-dependent (customized for each hardware platform).	Hardware-independent (relies on HAL for hardware-specific operations).
Purpose	To isolate the kernel and upper layers from hardware changes.	To manage all system processes, scheduling, memory, and I/O operations.
Example Tasks	Providing standardized APIs for device drivers, managing interrupts, timers, and I/O interfaces.	Process scheduling, memory allocation, file system handling, inter-process communication.
Modification Need	Needs modification when hardware changes.	Usually remains unchanged when hardware changes (thanks to HAL).





Why is HAL Important for Portability?

HAL plays a **crucial role in achieving OS portability** across different hardware platforms because:

1. Hardware Independence

- The kernel does not need to know the specific details of each hardware component.
- It communicates with devices through **standardized HAL interfaces**.

2. Easy Adaptation to New Hardware

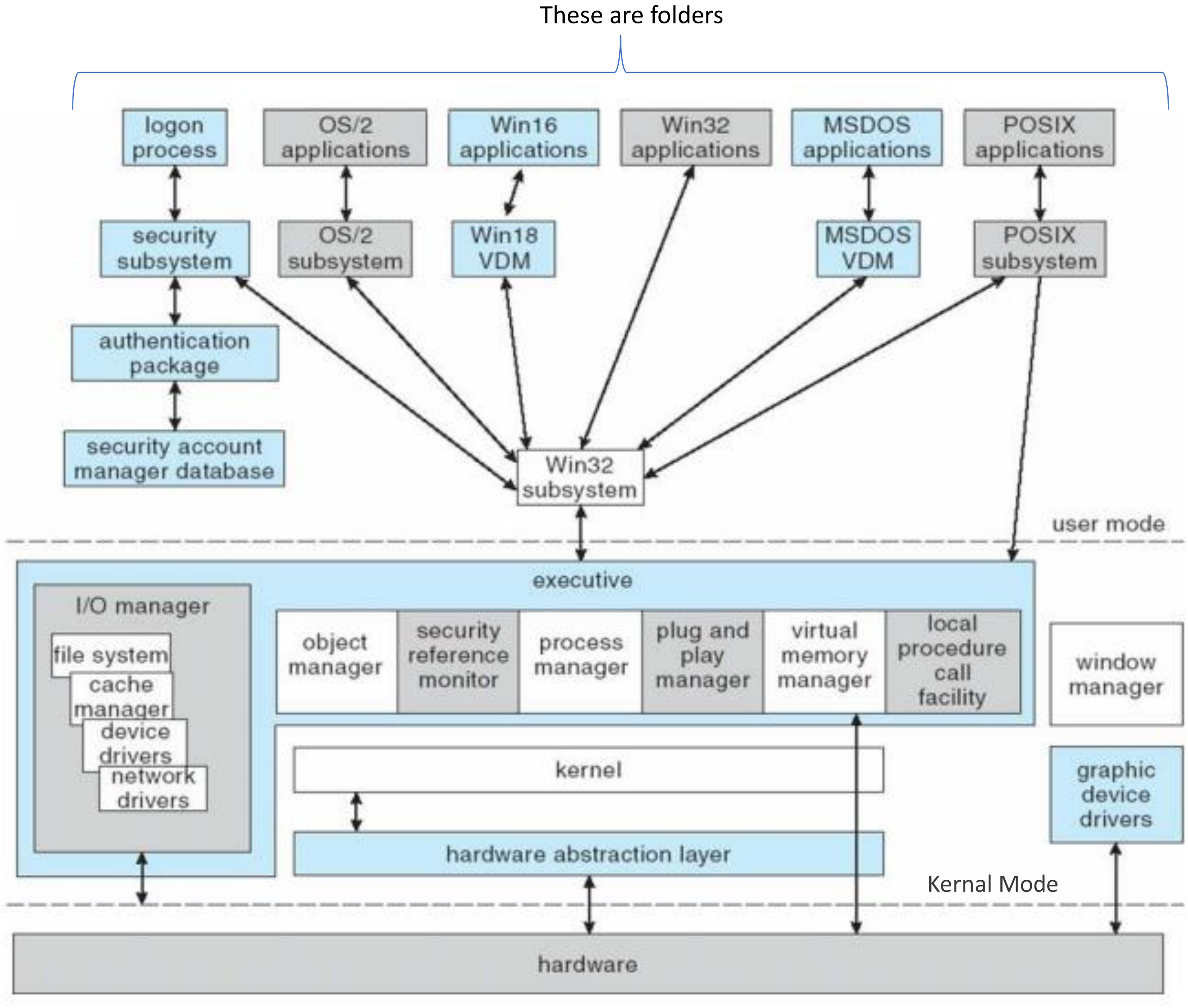
- When you move an OS to a new platform (e.g., from Intel to ARM processor), you **only need to modify or replace the HAL**, not the kernel or higher-level OS code.

3. Simplified Development

- Developers can write kernel code once and run it on various devices without rewriting for each hardware type.

16. Draw a simplified diagram of Windows 7 architecture and explain the flow of control when a user program requests access to hardware (e.g., a printer or disk).

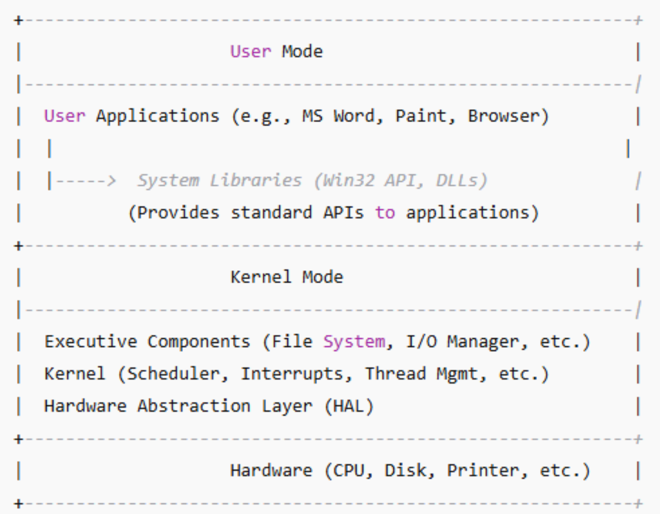
Ans.



Step-by-Step Flow of Control: Let's say you're in **MS Word** and click **"Print"** to print a document.

Step 1 – User Program Request

- The user program (MS Word) is in **user mode**.
- It calls a **Windows API function**, like PrintDocument() or a related system call.
- These functions are part of **system libraries (DLLs)** such as winspool.drv or gdi32.dll.
- **Purpose:** Provide a standard way to request OS services (like printing).



```
+------------------------------------------------------------+
|                        User Mode                            |
|------------------------------------------------------------|
| User Applications (e.g., MS Word, Paint, Browser)           |
| |                                                          |
| |----->  System Libraries (Win32 API, DLLs)                 |
|          (Provides standard APIs to applications)          |
+------------------------------------------------------------+
|                        Kernel Mode                          |
|------------------------------------------------------------|
| Executive Components (File System, I/O Manager, etc.)       |
| Kernel (Scheduler, Interrupts, Thread Mgmt, etc.)           |
| Hardware Abstraction Layer (HAL)                            |
+------------------------------------------------------------+
|                        Hardware (CPU, Disk, Printer, etc.)  |
+------------------------------------------------------------+
```

Step 2 – Transition from User Mode to Kernel Mode

- The API function internally triggers a **system call**.
- Control passes from **user mode → kernel mode** via a **system call interface**.
- **Purpose:** Move to the part of Windows that has permission to interact with hardware.

Step 3 – OS Kernel Handles the Request

- The **Windows Executive components** handle the request:
- **I/O Manager**: Routes I/O requests to the correct driver.
- **Spooler Service**: Manages print jobs and sends data to the printer driver.
- **File System Manager** (if disk I/O): Reads/writes data from/to files.
- **Purpose:** The kernel ensures proper management, scheduling, and security before touching hardware.

Step 4 – Communication with Device Drivers

- The **I/O Manager** calls the **device driver** responsible for the printer or disk.
- Drivers are small programs in **kernel mode** that know how to talk to specific hardware.
- **Purpose:** Translate general OS commands into device-specific commands.

Step 5 – HAL Interacts with Hardware

- The **device driver** sends hardware-level instructions through the **Hardware Abstraction Layer (HAL)**.
- The **HAL** converts these into commands that the specific printer or disk understands.
- **Purpose:** Provide a consistent interface for the kernel and drivers, regardless of hardware type.

Step 6 – Hardware Performs the Operation

- Finally, the **printer** prints the document or the **disk** reads/writes the requested data.
- The device sends a **signal (interrupt)** back to the kernel when it’s done.
- **Purpose:** Notify the OS that the task is complete.

Step 7 – Control Returns to User Mode

- The kernel updates the process status and passes the result back to the **application** (e.g., “Print successful!”).
- The program continues executing in **user mode**.

17. Describe how computer system performance can be measured and analysed. Define system throughput and CPU utilization. Are these two metrics related to one another? Justify your answer.

Ans.

- Computer system performance means **how fast and efficiently a computer can do its work** — like running programs, processing data, or handling multiple users.
- We can measure performance using several **quantitative (numerical)** measures.

Measure	What it means (Simple Explanation)	Example
Response Time	Time taken by the computer to respond to a request.	How long it takes for a webpage to open after your click.
Throughput	Number of tasks completed in a given time.	How many print jobs a printer completes in 1 minute.
CPU Utilization	How much of the CPU's capacity is being used.	If CPU utilization is 90%, it's working almost fully.
Memory Utilization	How much RAM is used by programs.	If 4 GB out of 8 GB is used, utilization = 50%.
Disk I/O Rate	How fast data can be read/written to disk.	Useful in data-heavy systems.
Network Latency	Time taken for data to travel between systems.	Important in online applications.

- Factors Affecting System Performance:

Factor	Explanation (Simple)
CPU Speed	Faster processors handle tasks quicker.
RAM Size	More memory = less swapping and faster performance.
Disk Type	SSDs are much faster than HDDs.
Network Speed	Affects data transfer time.
Operating System	Efficient OS manages resources better.
Running Programs	Too many apps open → system slows down.

- Example (Simple Scenario):
  - Imagine you open multiple apps (browser, video player, game):
    1. CPU usage jumps to 90%.
    2. RAM usage hits 95%.
    3. The game lags.
  - You open **Task Manager** → see "Memory" and "CPU" usage graphs.
  - You realize the **bottleneck** is RAM (not enough memory).
- **System Throughput** means the **number of tasks (processes, jobs, or requests)** a computer system can complete in a **given period of time**. Means throughput tells us **how much work the system is doing**.

  Formula:

$$\text{System Throughput} = \frac{\text{Total number of completed processes}}{\text{Total time}}$$

  Example: If a system completes 200 tasks in 10 seconds.

$$\text{System Throughput} = \frac{200}{10} = 20\ tasks\ per\ second$$

- **CPU Utilization** means the **percentage of time** the CPU is **actively working (not idle)**. Means It shows **how busy** the CPU is.
Formula:

$$\text{CPU Utilization} = \frac{\text{CPU busy time}}{\text{Total time}} \times 100\ \%$$

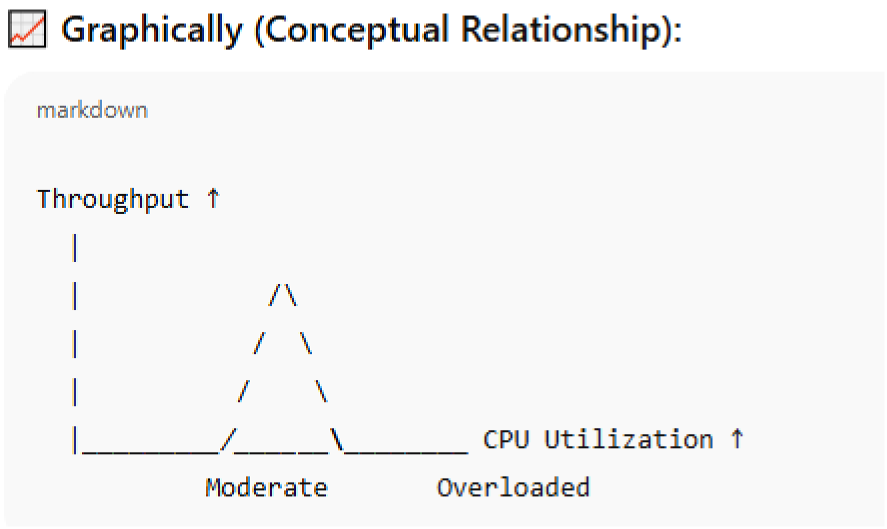
Example: If CPU was busy for 8 seconds out of 10 seconds,

$$\text{CPU Utilization} = \frac{8}{10} \times 100\ \% = 80\%$$

- **Relationship Between Throughput and CPU Utilization:**

Condition	CPU Utilization	Throughput	Reason
CPU idle most of the time	Low	Low	Not enough tasks to process
CPU working efficiently	Moderate to High	High	CPU processes many tasks successfully
CPU overloaded (too many processes)	Very High (close to 100%)	May Decrease	Too much context switching, causing slowdown

- When **CPU utilization increases** (i.e., CPU is busy doing useful work), the **throughput usually increases** — because more tasks are being processed.
- But if the CPU is **overloaded** (too many tasks waiting), it spends more time switching between processes instead of completing them — so **throughput may fall**.



📈 Graphically (Conceptual Relationship):

markdown

```
Throughput ↑
  |
  |            /\
  |           /  \
  |          /    \
  |_________/______\________ CPU Utilization ↑
           Moderate       Overloaded
```

18. Consider a system with one CPU and one I/O device. A single process P1 arrives at time 0. Its execution pattern is as follows:

- **CPU burst_1:** 2 units
- **I/O burst:** 4 units
- **CPU burst_2:** 3 units

Additional conditions:

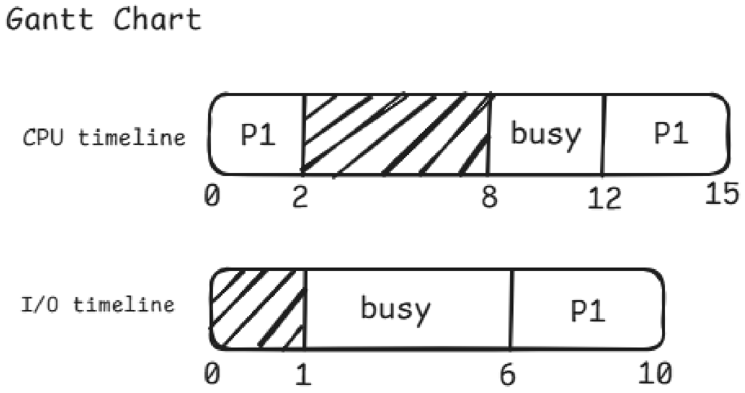
- The I/O device is busy with another job from time 1 to 6.
- The CPU is busy with another job from 8 to 12.

Scheduling policy: First Come First Serve (FCFS) is used for both CPU and I/O.

Tasks:

1. Draw the timeline (or Gantt chart) of P1's execution.
2. Calculate the Completion Time of P1.
3. Calculate the Turnaround Time (TAT) of P1.
4. Calculate the Total Waiting Time (WT) of P1 (only the time spent waiting in the Ready Queue for CPU, not I/O blocked time).

Ans.



Completion Time of P1 = 15

Turnaround Time (TAT) = Completion time − Arrival time = 15 – 0 = 15

Total Waiting Time (WT) in Ready Queue for CPU = wait before 1st CPU + wait before 2nd CPU

= 0 + (12 – 10) = 2

19. Consider a system with one CPU and one I/O device. A single process P1 arrives at time 0. Its execution pattern is as follows:

- **CPU burst_1:** 3 units
- **I/O burst:** 5 units
- **CPU burst_2:** 4 units

Background conditions:

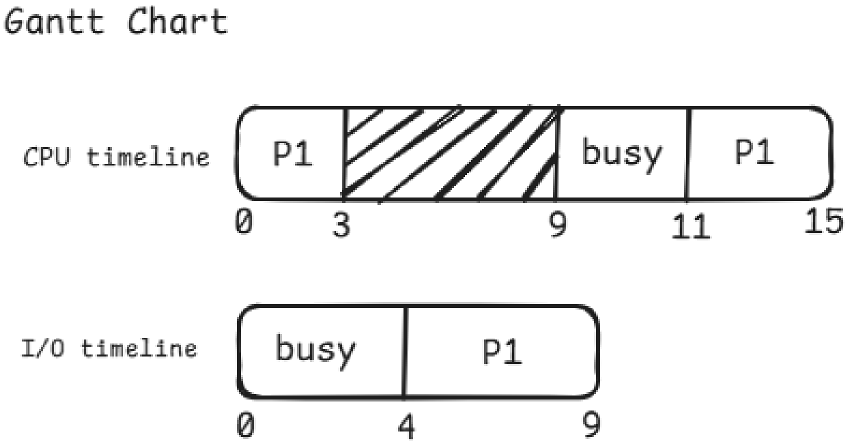
- I/O device is busy with another job from time 0 → 4.
- CPU is busy with another job from time 9 → 11.

Scheduling: FCFS for both CPU and I/O.

Tasks:

1. Draw the timeline (or Gantt chart) of P1's execution.
2. Calculate the Completion Time of P1.
3. Calculate the Turnaround Time (TAT) of P1.
4. Calculate the Total Waiting Time (WT) of P1 (only the time spent waiting in the Ready Queue for CPU, not I/O blocked time**).**

Ans.



Completion Time of P1 = 15

Turnaround Time (TAT) = Completion time − Arrival time = 15 – 0 = 15

Total Waiting Time (WT) in Ready Queue for CPU = wait before 1st CPU + wait before 2nd CPU

= 0 + (11 – 9) = 2

20. Describe key characteristics, advantages, and limitations of Multitasking (a.k.a Time Sharing) Operating systems. Consider a multitasking system in which we have three processes P1(1,2,3), P2(1,0,0), and P3(1,0,0) with their CPU and I/O requirements. For example, the CPU and I/O requirements of P1 (1,2,3) can be interpreted as P1 first uses CPU for 1 second, then I/O for 2 seconds, finally CPU for 3 seconds. If the CPU quantum is 2 seconds, then write the order of execution with a time scale for the given processes P1, P2, and P3. Consider, P1 is at the front of the ready queue followed by P2 and P3.

Ans. A **Multitasking Operating System** (also called **Time-Sharing OS**) allows **multiple programs (tasks)** to run simultaneously **at the same time** on a **single CPU**.

Key Characteristics:

Characteristic	**Description**
Time sharing	Each process gets a small time slot (quantum) of CPU time in turn.
CPU scheduling	CPU is scheduled among multiple tasks using algorithms like Round Robin.
Pre-emptive nature	If a process exceeds its time slice, the CPU is taken away and given to the next process.
Fast context switching	OS rapidly switches between processes to share CPU time fairly.
Interactive use	Designed to support multiple interactive users at terminals.
Responsiveness	System responds quickly to user commands.
Multi-user support	Several users can work on the same system simultaneously (each getting a share of CPU time).

Advantages:

Advantage	**Explanation**
Efficient CPU utilization	CPU never remains idle; it always executes some task.
Better responsiveness	Users get quick feedback because CPU switches frequently between tasks.
Supports multiple users	Many users can share a single system (common in mainframes).
Fair resource sharing	Each process/user gets a fair chance to use CPU and I/O devices.
Improves productivity	Multiple jobs progress together instead of waiting for one to finish.

Limitations / Disadvantages:

Limitation	**Explanation**
High overhead	Frequent context switching consumes CPU time and memory.
Complex OS design	Managing multiple tasks and users needs complex scheduling and protection mechanisms.
Less security	Users share the same system, so data protection can be difficult.
Performance drops	If too many processes run together, system becomes slow (thrashing).
Needs more resources	Requires large memory and fast CPU to handle many tasks efficiently.

21. Consider a system which is using the First Come First Serve (FCFS) CPU scheduling algorithm to schedule the following set of processes.

Process	**CPU-burst**	**Arrival time**
P1	10	0
P2	5	15
P3	5	20
P4	10	23

Calculate CPU utilization, Throughput, Average Turnaround Time, and Average Waiting Time.

(NOTE: In this question the context switch cost/overhead is negligible or 0 ms).

22. Suppose we have a single CPU System, and there are 10 processes in the ready queue. The largest CPU-burst for a process may be 50 ms. What is the maximum waiting time for a process to get the CPU in the Round Robin scheduling with a quantum of 50 ms.

(NOTE: In this question the context switch cost/overhead is negligible or 0 ms)

23. Consider the following set of processes, and schedule them using FCFS CPU Scheduling Algorithm. Also, Calculate CPU utilization, Throughput, Average Turnaround Time, and Average Waiting Time.

Process	Arrival Time	(CPU-burst, I/O-burst, CPU-burst)
P1	0	(3, 2, 1) Total CB= 4
P2	1	(1, 3, 1) Total CB= 2
P3	3	(1, 1, 2) Total CB= 3
P4	6	(2, 4, 3) Total CB= 5

24. Consider a variant of the RR scheduling algorithm in which the entries in the ready queue are pointers to the PCBs. What would be the effect of putting two different pointers to the same process in the ready queue? Explain with a proper diagram.

25. UNIX System V Release 3 (SVR3) employs a multilevel feedback queue (MLFQ) scheduling algorithm. Explain how this algorithm operates and discuss the reasons it is considered effective.

26. Schedule the following processes using the UNIX SVR3 algorithm.

Process	Arrival Time	Base Priority	CPU Burst (ms)	I/O Burst (ms)	Nice Value
P1	0	60	2	0	0
P2	0	60	1	0	5
P3	0	60	1	0	5

27. Define contiguous, linked and indexed file allocation methods. Also, define the file system used in LINUX/UNIX operating systems.

28. Draw and describe UNIX/Linux directory structure. Also, define absolute and relative pathname with the help of relevant examples.

29. Describe file system used in traditional UNIX Operating System.

30. In the UNIX File system (UFS) when a new file is created, the operating system must allocate and update several internal data structures.

a) Explain step by step what happens during file creation with respect to: Acquiring a free inode from the inode free list, allocating a free data block for the file's contents, Updating the parent directory with the new filename → inode mapping.

b) Discuss how the superblock, inode table, and directory blocks are modified in this process.

c) Why is journaling (or write-ahead logging in newer systems) important during this sequence of operations? Illustrate what problem may occur if the system crashes midway.

31. Modern operating systems often aim to balance functionality and security. Discuss the essential requirements of a protection mechanism in OS design, and analyze how the absence of one or more of these requirements can impact system reliability. Support your answer with examples from real-world operating systems.

32. The Access Control Matrix provides a conceptual framework for defining permissions. Explore its advantages and limitations in practical system implementation. Compare how different operating systems have adapted or deviated from the pure ACM model.

33. Access Control Lists (ACLs) and Capability Lists are two widely used mechanisms derived from the Access Control Matrix. Critically examine their comparative advantages and disadvantages.

34. Authentication has evolved from simple password-based systems to biometric and multi-factor mechanisms. Critically analyse the challenges faced by traditional UNIX authentication (with salt and hashing) and contrast it with modern multi-factor authentication approaches.

35. Explain how the concepts of deadline and response time influence the design and performance of a Real-Time Operating System (RTOS). Analyze how missing a deadline affects system behavior in Hard, and Soft, RTOS with suitable real-life examples.

36. Consider an automated airbag control system in a modern vehicle. The system continuously monitors crash sensors and must trigger airbag deployment within a few milliseconds when a collision is detected. Analyze the scenario and identify which tasks in this system can be classified as real-time tasks. Explain whether this system represents a hard or soft real-time system, justifying your answer with reasoning related to deadline and response time.

37. RT-Linux uses a dual-kernel architecture to provide real-time capabilities alongside standard Linux operations. Examine how this architecture separates real-time tasks from non-real-time processes.