OPERATING SYSTEMS MODULE 1

Page 7 – Introduction to Operating System

- Definition & Role
  An Operating System (OS) is system software that serves as a bridge between the user and the computer hardware.
  - System Software: Manages and controls hardware and provides the foundation for running application software.
  - Without an OS, hardware cannot be efficiently or easily used for productive tasks.
  - OS abstracts the complexity of hardware and presents a simplified interface to the user.
- From a broader view (*from extra PDF*):
  - A computer system can be seen as Hardware + Software.
    - Hardware: Physical components (CPU, RAM, I/O devices, storage, network interfaces).
    - Software: Application Software (user-focused tasks) + System Software (hardware management).
  - OS falls under System Software and controls how applications use hardware resources.

Page 8 – Goals of Operating System

- Primary Goal – Make the computer convenient to use:
  - Provide a friendly and intuitive environment for the user.
  - Hide hardware complexities.
  - Automate routine tasks (memory allocation, process scheduling).
- Secondary Goal – Efficient resource use:
  - Maximize utilization of CPU, memory, and I/O devices.
  - Avoid idle hardware time.
- Combined Perspective (*extra points from PDF*):
  - OS should execute user programs, help in solving user problems, and make the system responsive.
  - Achieve balance between:
    - Convenience vs. Efficiency
    - Performance vs. Fairness – ensuring one user/process doesn't monopolize resources.

Pages 10–11 – What Operating Systems Do

- As a Resource Allocator:
  - Manages CPU time, memory space, storage devices, I/O devices, and network bandwidth.
  - Decides which request gets priority when multiple processes demand the same resource.
- As a Control Program:
  - Monitors execution of programs to prevent errors and improper use of the computer.
  - Prevents unauthorized access and illegal hardware operations.
- Extra Details (*from PDF*):
  - Acts as the traffic controller for all resource usage.
  - Ensures fairness and efficiency in all resource-sharing decisions.

Page 12 – Resources Managed by an OS

Primary Resources:

1. CPU – Executes instructions; OS decides which process gets CPU time.
2. Memory – Temporary storage (RAM) for active programs.
3. File Storage – Permanent data storage on HDD/SSD.
4. I/O Devices – Keyboards, mice, displays, printers, scanners, etc.
5. Network Connections – Allow communication and sharing with other systems.

Extra from PDF:

- Each of these resources requires rules for access to avoid conflicts.
- OS uses device drivers and hardware controllers to interact with resources.

Page 13 – Extended OS Roles

- Facilitates resource sharing between processes/users.
- Optimizes program execution through efficient scheduling and memory use.
- Supports different user interfaces – CLI, GUI, touch-based.
- Ensures system performance goals are met (e.g., throughput, latency targets).
- Extra Insight (*from PDF*):
  - Acts as an enabler of multitasking – allowing multiple programs to run without interfering.
  - Ensures the system can support varied workloads (interactive, batch, real-time).

Pages 14–15 – Primary Components of an OS

1. Kernel – Core of the OS; interacts directly with hardware, manages all system resources.
    - Loaded into protected memory at boot and remains active.
    - Handles process control, memory allocation, and low-level I/O.
2. Process Scheduler – Allocates CPU time fairly to processes.
3. Memory Manager – Allocates memory to processes, prevents overlap.
4. File System – Manages creation, deletion, and organization of files.
5. Device Drivers – Translate OS instructions into device-specific operations.
6. User Interface – Allows interaction via CLI, GUI, or touch.

Extra from PDF:

- Kernel sub-components:
    - Scheduler – CPU time distribution.
    - Supervisor – Grants access to system resources.
    - Interrupt Handler – Deals with hardware-generated interrupts.
    - Memory Manager – Tracks and allocates RAM.

Pages 16–17 – Booting Process

Boot Steps:

1. Power On – Power supply sends "Voltage Good" signal to motherboard.
2. BIOS/UEFI:
    - BIOS: Legacy firmware stored in ROM/EPROM, performs POST (Power-On Self-Test).
    - UEFI: Modern firmware, stored on EFI partition, supports GUI, large drives, secure boot.
3. Boot Loader – Loaded by BIOS/UEFI, locates and loads OS kernel.
4. Kernel Initialization – Sets up memory management, loads essential device drivers, starts system daemons.
5. User Space Start – Initiates user interface and services.

Extra from PDF:

- MBR (Master Boot Record) – For BIOS systems; contains partition table and boot code.
- ESP (EFI System Partition) – For UEFI systems; contains boot loaders and .efi files.
- Early Kernel Initialization – Only loads minimal drivers required to proceed.

Pages 19–21 – OS Services

Services Helpful to the User:

- User Interface – CLI (keyboard commands), GUI (icons, menus), or touchscreen.
- Program Execution – Load programs into memory, execute, and terminate.
- I/O Operations – Manage device and file I/O requests.
- File System Manipulation – Create, delete, search, and modify files/directories; manage permissions.
- Communication – Between processes via shared memory or message passing.
- Error Detection – Detect and handle CPU, memory, I/O, or program errors.

Services for Efficient Operation:

- Resource Allocation – Fair distribution of CPU, memory, and devices.
- Logging – Record resource usage.
- Protection & Security – Prevent unauthorized access; ensure isolation between processes.

Extra from PDF:

- Debugging Facilities – Tools within OS to help developers troubleshoot issues.
- Security extends to hardware – Prevents misuse of external I/O devices.

Pages 23–25 – OS Functioning & Design Issues

OS Primary Functions:

1. Resource Management – Efficient use of CPU, memory, and devices.
2. Process Management – Start, stop, and schedule processes.
3. Memory Management – Track, allocate, and deallocate memory; handle virtual memory.
4. Security – Implement access control and encryption.
5. File Management – Organize storage, manage file operations.
6. Device Management – Coordinate with peripheral devices.
7. Networking – Enable system-to-system communication.
8. User Interface – Provide means of interaction.

Design Issues:

- System Type Influence – Desktop, mobile, distributed, or real-time systems have different requirements.

- User Goals – Easy to use, reliable, safe, and fast.
- Developer Goals – Easy to design, maintain, flexible, and efficient.
- Policy vs. Mechanism:
    - Mechanism – "How" something is done (e.g., CPU timer setup).
    - Policy – "What" is done (e.g., time quantum length).

Extra from PDF:

- Portability – Easier if OS is written in high-level languages (e.g., C/C++).
- Modularity – Separation of policy and mechanism makes adaptation easier.

Alright — let's move on to pages 26–36 from your *osmod1.pdf*, keeping the same merged, detailed, exam-ready format with added explanations from *OPERATINGSYSTEM1.pdf* where relevant.

Page 26–28 – Design Issues (continued)

Separation of Policy and Mechanism

- Mechanism – Defines how something is done.
  Example: A CPU timer interrupt is a *mechanism* for preempting processes.
- Policy – Defines what should be done.
  Example: Deciding the *length of the time quantum* for a process.
- Why Separate?
    - Policies often change depending on workload, user requirements, or hardware changes.
    - If policy is mixed with mechanism, any change requires rewriting core OS code.
    - Separating them makes the OS flexible, easier to modify, and portable.

Implementation Considerations (*from extra PDF*)

- OS is a collection of many programs, often developed by multiple teams over years.
- Languages used:
    - C and C++: Most OS kernels (Linux, Windows NT) are primarily written in these for portability and performance.
    - Assembly: For hardware-specific, low-level control.
- Portability:
    - Easier to adapt to new hardware if OS is in a higher-level language.
    - Low-level hardware interactions can be isolated into modules.

Page 29–31 – OS Structuring: Monolithic Structure

Definition

- All OS functionality is placed in one large kernel program running in a single address space.
- Examples: Original UNIX, early MS-DOS.

Structure

- Kernel includes:
  - File system
  - CPU scheduling
  - Memory management
  - Device drivers
- All code runs in kernel mode, with full access to hardware.

Merits

- High performance: Direct system calls without extra layers.
- Simple design: Everything is compiled together; easier to implement basic OS from scratch.

Demerits

- Security risks: Any bug can crash the whole system.
- Poor stability: Fault in one part can bring down everything.
- Difficult maintenance: Any change may require rebuilding the whole kernel.

Extra from PDF:

- MS-DOS is an extreme monolithic example — no separation of user and kernel space.
- UNIX is monolithic but with structured interfaces for device drivers and file systems.

Page 33–35 – OS Structuring: Layered (Modular) Approach

Definition

- OS is divided into layers, each providing services to the layer above and using services of the layer below.
- Layer 0 = hardware; Top layer = user interface.
- Each layer has well-defined responsibilities.

Merits

- Modularity: Changes in one layer don't affect others.
- Easier debugging: Can test layer-by-layer.
- Better maintainability.

Demerits

- Performance overhead: Calls must pass through multiple layers.
- Design complexity: Defining exact layer boundaries is challenging.

Extra from PDF:

- A layered OS can resemble an onion — each layer wraps services of the layer below.
- Examples: THE OS (Technische Hogeschool Eindhoven) was a pioneering layered OS.

Page 36–38 – OS Structuring: Microkernel Approach

Definition

- Removes all non-essential services from the kernel, running them in user space instead.
- The kernel contains only:
    - Inter-process communication (IPC)
    - Basic scheduling
    - Minimal memory management
- All other services (device drivers, file systems, networking) run as separate processes.

Merits

- Easier to extend: Add new services without changing the kernel.
- Better security: Most services run in user space, isolated from kernel.
- Portability: Minimal kernel code to adapt when moving to new hardware.

Demerits

- Performance penalty: IPC and context switching between kernel and user services is slower.
- Increased complexity: Message-passing mechanisms can be tricky.

Extra from PDF:

- Examples: Mach microkernel, QNX, Minix.
- Major challenge is reducing the cost of frequent kernel–user transitions.

Page 39–40 – OS Structuring: Loadable Kernel Modules (LKM) Approach

Definition

- Core kernel provides essential services.
- Additional services are loaded dynamically at boot or runtime as modules.
- A module is an object file that can be linked/unlinked to the running kernel.

Use Cases

- Add hardware support (device drivers).
- Add new file systems.
- Add system calls or security extensions.

Merits

- Flexibility: Load modules only when needed; unload to save memory.
- Modularity: Similar to layered design but without the rigid stack.
- Faster development: Can add functionality without recompiling kernel.

Extra from PDF:

- Linux uses LKMs extensively for drivers (e.g., `insmod` to insert, `rmmod` to remove).
- Reduces kernel bloat by keeping infrequently used features out of memory.

Page 41–42 – Abstractions and the Process

Abstraction in OS

- OS hides physical details of devices, memory, and time.
- Converts raw hardware into virtual resources:
  - Process abstraction for CPU.
  - Virtual memory abstraction for RAM.
  - File abstraction for storage.

Process

- Definition: A program in execution with:
  - Program code (text section)
  - Current activity (program counter, registers)

    - Associated resources (open files, memory)
- Key properties:
    - Single thread of execution
    - State (ready, running, waiting)
    - Isolation from other processes

Extra from PDF:

- Process is the fundamental unit of work for the OS scheduler.
- Threads are a lighter form of process, sharing the same resources.

Page 43 – Influence of Security

Four Security Categories

1. Availability – Prevent denial of service; keep system operational.
2. Confidentiality – Prevent unauthorized reading of data.
3. Data Integrity – Prevent unauthorized changes to data.
4. Authenticity – Verify user identity and ensure messages are genuine.

Extra from PDF:

- Security is enforced via:
    - Authentication mechanisms (passwords, biometrics).
    - Access control lists (ACLs).
    - Encryption for data at rest and in transit.

If you want, I can now move to pages 44 onward, which in your other PDF (*OPERATINGSYSTEM1.pdf*) covers detailed OS functionalities like process management, memory management, file systems, device management, security, user interfaces, and OS types — all of which are extremely high-weightage for exams.

Do you want me to continue with that next section?

Page 44–46 – Major Functions of an Operating System

An OS provides a wide set of core functions that keep the system usable, efficient, and secure. These can be grouped into management functions.

---

1. Process Management

- Definition: Process = a program in execution + its current state (registers, memory, open files).
- OS is responsible for:
    1. Process Creation & Termination – Loading code into memory, setting initial state, cleaning up after termination.
    2. Scheduling – Deciding which process gets CPU time.
        - *Short-term scheduling*: Which ready process runs next.
        - *Medium-term*: Temporarily suspending/resuming processes to balance load.
        - *Long-term*: Controlling number of processes admitted to the system.
    3. Synchronization – Coordinating processes to avoid conflicts (e.g., race conditions).
    4. Inter-process Communication (IPC) – Passing data between processes (message passing or shared memory).
    5. Deadlock Handling – Detecting, preventing, or recovering from deadlocks.

Extra from PDF:

- OS uses process control blocks (PCB) to store process details: ID, state, priority, registers, memory pointers.
- Process switching involves context switching, where CPU state is saved/restored.

---

2. Memory Management

- Definition: Allocating and tracking memory usage for processes.
- Key functions:
    1. Allocation – Assigning RAM to processes.
    2. Deallocation – Reclaiming memory after process finishes.
    3. Protection – Ensuring processes cannot access each other's memory.
    4. Swapping – Moving processes between RAM and disk to free space.
    5. Virtual Memory – Extends RAM using disk storage (paging, segmentation).

Extra from PDF:

- OS maintains a free memory list and page tables.
- Memory fragmentation is a key issue:
    - *Internal fragmentation* → unused space within allocated memory.
    - *External fragmentation* → scattered free space between allocations.

---

3. File System Management

- Definition: Controls how data is stored, retrieved, and organized.
- Key tasks:
    1. File creation/deletion
    2. Directory creation/deletion
    3. Mapping files to storage blocks
    4. File access control & permissions
    5. Buffering and caching for performance
- File attributes: Name, type, size, location, protection info, timestamps.

Extra from PDF:

- File systems can be hierarchical (tree-structured directories).
- Common file systems: FAT32, NTFS, ext4.
- File allocation methods: Contiguous, linked, indexed.

---

4. Device Management

- Definition: Controls and coordinates I/O devices.
- Functions:
    - Buffering – Temporary storage during I/O.
    - Caching – Storing frequently accessed data in faster storage.
    - Spooling – Queuing jobs for slow devices (e.g., printer).
    - Device drivers – Translate OS requests into device-specific commands.

Extra from PDF:

- OS maintains a device table with status info.
- Device independence: Same OS API works for different hardware.

---

5. Security & Protection

- Security – Protects against external threats (malware, hackers).
- Protection – Controls access to system resources to prevent misuse.
- Mechanisms:
    - Authentication – Verify user identity.
    - Authorization – Control which resources a user can access.
    - Encryption – Protect data in transit and at rest.
    - Auditing – Logging activities for review.

Extra from PDF:

- Access control can be user-based, role-based, or capability-based.
- Modern OS integrate security updates via patch management.

Page 47–48 – User Interfaces in OS

1. Command-Line Interface (CLI)

- Text-based; users type commands.
- Example: Bash (Linux), Command Prompt (Windows).
- Pros: Fast for experienced users, scriptable.
- Cons: Steep learning curve.

2. Graphical User Interface (GUI)

- Icons, windows, menus; point-and-click.
- Example: Windows desktop, macOS Finder.
- Pros: Intuitive, visually rich.
- Cons: More resource-heavy.

3. Touch-based Interfaces

- Found in mobile OS (Android, iOS).
- Gesture-based commands.

Extra from PDF:

- Some OS support multiple UI modes — e.g., Windows Server can be CLI-only or GUI-enabled.
- Shell: Program that interprets commands (e.g., Bash, PowerShell).

Page 49–50 – Types of Operating Systems

1. Batch OS

- No direct user interaction; jobs are collected and processed in batches.
- Example: Early mainframes.
- Pros: Efficient for large, repetitive jobs.
- Cons: No real-time feedback.

2. Multiprogramming OS

- Multiple jobs in memory; CPU switches between them.
- Improves CPU utilization.

3. Multitasking OS

- Multiple programs executed seemingly at the same time on a single CPU by rapidly switching between them.
- Example: Windows, Linux.

4. Real-Time OS (RTOS)

- Guaranteed response within strict deadlines.
- Hard RTOS – Missing a deadline = failure (e.g., flight control).
- Soft RTOS – Occasional delays acceptable (e.g., video streaming).

5. Distributed OS

- Manages multiple computers as a single system.
- Shares resources and workloads.

Extra from PDF:

- Time-sharing OS: Variant of multitasking, but with very short CPU time slices to give illusion of parallelism.
- Network OS: Focuses on providing network services (file sharing, printing).

Page 51 – Summary Table (Management Functions)

OS Function	Example Mechanism	Example Policy
CPU Scheduling	Timer interrupt	Round-robin, Priority
Memory Allocation	Page tables	Fixed-size vs variable-size allocation
File Management	File descriptors	Access rights, file quotas
Device Management	Device queues	FIFO, priority device scheduling
Security	ACLs, passwords	Role-based access control