Unit I – Introduction to Operating Systems

UNIT I: Introduction: Definition and types of operating systems, Batch Systems, multi programming, time– sharing parallel, distributed and real-time systems, Operating system structure, Operating system components and services, System calls, system programs, Virtual machines.

1. Definition of Operating System (OS)

- An **Operating System (OS)** is a **system software** that acts as an intermediary between the **user** and the **computer hardware**.
- It manages **hardware resources** (CPU, memory, I/O devices) and provides **services** for computer programs.

👉 **Formal definition (by Silberschatz):**

An OS is a program that controls the execution of application programs and acts as an interface between users and the computer hardware.

---

2. Types of Operating Systems

(a) Batch Operating System

- Early computers used this system.
- Jobs with similar needs are grouped (batched) and executed together without user interaction.
- Example: IBM Mainframe systems.

Advantages: Better resource utilization.
Disadvantages: No direct user interaction, long turnaround time.

(b) Multiprogramming OS

- Multiple programs are kept in memory at the same time.
- CPU executes one process, while others wait for I/O → **increases CPU utilization**.

Example: Unix, Linux.

(c) Time-Sharing OS

- Extension of multiprogramming.
- Multiple users can access a computer **simultaneously**.
- CPU time is divided into **time slices** (quantum).

Example: Modern Unix/Linux systems.

(d) Parallel OS

- Uses **multiple processors (CPU cores)** for parallel execution.
- Provides higher performance, reliability, and throughput.

Example: Windows HPC, Supercomputers.

(e) Distributed OS

- Manages a **group of independent computers** and makes them appear as a **single coherent system**.
- Resources are shared across multiple systems.

Example: Amoeba, LOCUS.

(f) Real-Time OS (RTOS)

- Provides **immediate response** to input/output requests.
- Used where strict timing constraints are required.

Types:

1. **Hard RTOS** – Deadline must be met (e.g., aircraft control system).
2. **Soft RTOS** – Deadlines are flexible (e.g., multimedia streaming).

Example: VxWorks, RTLinux, QNX.

3. Operating System Structure

Different OS structures:

1. **Monolithic Structure** – OS is a single large program (e.g., MS-DOS).
2. **Layered Approach** – OS divided into layers, each built on the one below.
3. **Microkernel** – Only essential services in kernel (IPC, scheduling, memory), others in user space (e.g., QNX, Mac OS).
4. **Modules** – Kernel with dynamically loadable modules (e.g., Linux).
5. **Hybrid Systems** – Combination of microkernel + monolithic (e.g., Windows NT).

---

4. Operating System Components and Services

(a) Major Components:

1. **Process Management** – Handles process creation, scheduling, and termination.
2. **Memory Management** – Allocation/deallocation of memory, paging, segmentation.
3. **File System Management** – Organizes and provides access to files.
4. **I/O System Management** – Manages device drivers and I/O devices.
5. **Secondary Storage Management** – Disk scheduling, storage allocation.
6. **Protection & Security** – Controls access to system resources.
7. **Networking** – Provides communication between systems.
8. **Command Interpreter/System Call Interface** – User interaction with OS.

(b) Operating System Services:

- **Program execution**
- **I/O operations**
- **File manipulation**
- **Error detection**
- **Communication (IPC)**
- **Resource allocation**
- **Security & protection**

---

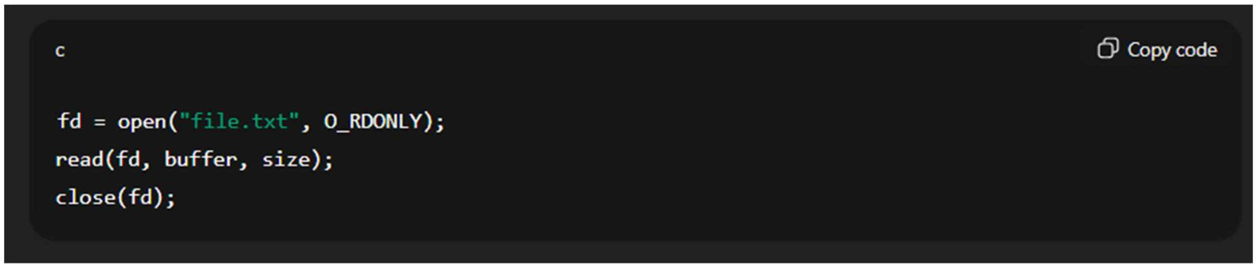
5. System Calls

- Interface between **user programs and OS**.
- Provide services like process creation, file operations, I/O handling.

Types of system calls:

1. Process control (create, terminate, wait, signal).
2. File management (open, read, write, close).
3. Device management (request, release).
4. Information maintenance (get/set time, get attributes).
5. Communication (send, receive messages).

Example (in C):



```c
fd = open("file.txt", O_RDONLY);
read(fd, buffer, size);
close(fd);
```

System Programs

- Programs provided with OS to make system easier to use.
- Examples:
    - File management utilities (ls, cp, rm in Linux)
    - Status info (ps, top)
    - Compilers, assemblers, loaders
    - Communication programs (chat, mail)

---

7. Virtual Machines

- A **virtual machine (VM)** is an abstraction of physical hardware.
- Provides an environment that appears as a separate computer.
- Implemented using software like **VMware, VirtualBox**.

Advantages:

- Isolation between systems.
- Easier testing and development.
- Better resource utilization.

Examples: Java Virtual Machine (JVM), VMware Workstation.

☑ **Summary for Exams**

- OS = interface between hardware & user.
- Types: Batch, Multiprogramming, Time-sharing, Parallel, Distributed, RTOS.
- Structures: Monolithic, Layered, Microkernel, Modular, Hybrid.
- Components: Process, Memory, File, I/O, Security, Networking.
- System Calls = interface to request OS services.
- System Programs = utilities for user support.
- Virtual Machines = abstraction of hardware for isolated execution.

Unit II – Process Management

UNIT II: Process Management: Process concept, Process scheduling, Cooperating processes, Threads, Interprocess communication, CPU scheduling criteria, Scheduling algorithms, Multiple-processor scheduling, Real-time scheduling and Algorithm evaluation.

1. Process Concept

- **Process** = a program in execution.
- Process includes:
    - **Program counter (PC)** – next instruction.
    - **Stack** – temporary data, function calls, return addresses.
    - **Data section** – global variables.
    - **Heap** – dynamic memory.

👉 A process is the **active entity**, while a program is a **passive entity**.

Process States

1. **New** – process is created.
2. **Ready** – waiting to be assigned CPU.
3. **Running** – instructions are being executed.
4. **Waiting (Blocked)** – waiting for I/O or event.
5. **Terminated** – finished execution.

👉 Represented by **Process State Diagram**.

---

2. Process Scheduling

Scheduling = deciding which process runs at a given time.

- **Long-term scheduling** – selects which jobs are admitted into system.
- **Medium-term scheduling** – suspends/resumes processes (to improve CPU utilization).
- **Short-term scheduling (CPU scheduling)** – decides which ready process gets CPU.

3. Cooperating Processes

- **Independent Process** → cannot affect/affected by others.
- **Cooperating Process** → can affect/affected by others.

Advantages of cooperating processes:

1. Information sharing (databases).
2. Computation speed-up (parallel execution).
3. Modularity (divide system into smaller processes).
4. Convenience (multi-user environment).

---

4. Threads

- **Thread = lightweight process (LWP)**.
- Shares same code, data, and resources of process but has own **program counter, register, and stack**.

Types:

- **User-level threads** (managed by user libraries).
- **Kernel-level threads** (managed by OS kernel).

Benefits: Faster context switching, responsiveness, resource sharing, multiprocessor utilization.

---

5. Interprocess Communication (IPC)

Processes communicate and synchronize using **IPC mechanisms**.

Methods:

1. **Shared Memory** – processes share a region of memory. Faster, but needs synchronization.
2. **Message Passing** – processes exchange messages (send/receive). Easier, but slower.

IPC provides:

- Data exchange, synchronization, modularity.

6. CPU Scheduling Criteria

Good scheduling = efficient CPU utilization.

Criteria include:

- **CPU Utilization** – keep CPU busy (goal: 100%).
- **Throughput** – number of processes completed per unit time.
- **Turnaround Time** – time from submission to completion.
- **Waiting Time** – time spent waiting in ready queue.
- **Response Time** – time from request submission to first response.

---

7. Scheduling Algorithms

Different algorithms decide which process runs:

(a) First Come First Serve (FCFS)

- Processes served in order of arrival.
- **Non-preemptive**.
- Simple but **convoy effect** (long job delays short jobs).

(b) Shortest Job Next (SJN) / Shortest Job First (SJF)

- Process with shortest burst time executes first.
- **Optimal (minimizes average waiting time)**.
- Can be **preemptive** (SRTF – Shortest Remaining Time First) or **non-preemptive**.

(c) Priority Scheduling

- Each process assigned a priority.
- Higher priority → executed first.
- **Problem:** Starvation (low priority may never execute).
- **Solution:** Aging (gradually increase waiting process priority).

(d) Round Robin (RR)

- Each process gets **time quantum (slice)**.
- After quantum expires → process moved to ready queue.
- **Fair** for time-sharing systems.

(e) Multilevel Queue Scheduling

- Processes divided into queues (foreground, background).
- Each queue has its own scheduling algorithm.

(f) Multilevel Feedback Queue

- Processes can move between queues.
- More flexible than multilevel queue.

(g) Multiple Processor Scheduling

- More than one CPU.
- **Asymmetric multiprocessing** – one master processor controls scheduling.
- **Symmetric multiprocessing (SMP)** – all processors are self-scheduling.

(h) Real-Time Scheduling

- Used in **real-time systems**.
- **Hard RTOS** – deadline must be met (miss = system failure).
- **Soft RTOS** – deadlines flexible.

Algorithms:

- Rate Monotonic Scheduling (RMS).
- Earliest Deadline First (EDF).

(i) Algorithm Evaluation

Methods to evaluate scheduling algorithms:

1. **Deterministic modeling** – use exact workload.
2. **Queuing models** – mathematical analysis.
3. **Simulation** – model system using simulation programs.
4. **Implementation** – implement in real OS.

☑ Quick Revision Points

- **Process = program in execution.** States: new, ready, running, waiting, terminated.
- **Threads = lightweight processes.**
- **IPC = Shared memory + Message passing.**
- **CPU Scheduling Criteria = utilization, throughput, turnaround, waiting, response.**
- **Algorithms = FCFS, SJF/SRTF, Priority, RR, Multilevel Queue, Multilevel Feedback Queue, SMP, Real-Time.**
- **Evaluation = deterministic, queuing, simulation, implementation.**

Unit III – Process Synchronization and Deadlocks

Process Synchronization and Deadlocks: The Critical-Section problem, synchronization hardware, Semaphores, Classical problems of synchronization, Critical regions, Monitors, Deadlocks-System model, Characterization, Deadlock prevention, Avoidance and Detection, Recovery from deadlock, Combined approach to deadlock handling.

1. The Critical-Section Problem

- **Critical Section:** A part of the program where the process accesses **shared resources (variables, files, memory, devices)**.
- If multiple processes enter the critical section simultaneously → **race condition** (incorrect results).

👉 **Solution must satisfy:**

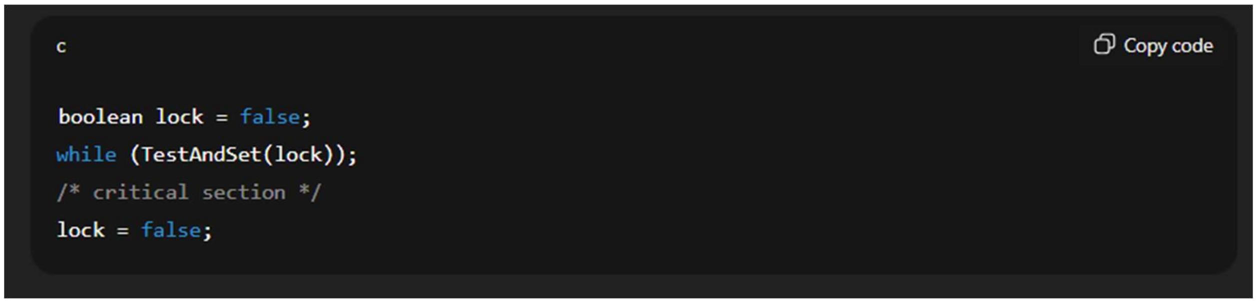
1. **Mutual Exclusion** – only one process executes critical section at a time.
2. **Progress** – if no process is in CS, some waiting process must be allowed to enter.
3. **Bounded Waiting** – after a process requests entry, it must eventually get a turn (no starvation).

---

2. Synchronization Hardware

- Hardware solutions provide **atomic operations**.
- **Test-and-Set Instruction (TS):**
    - A single atomic instruction used to lock/unlock.
    - Prevents multiple processes from entering CS.

Example (Pseudo-code):



```c
boolean lock = false;
while (TestAndSet(lock));
/* critical section */
lock = false;
```

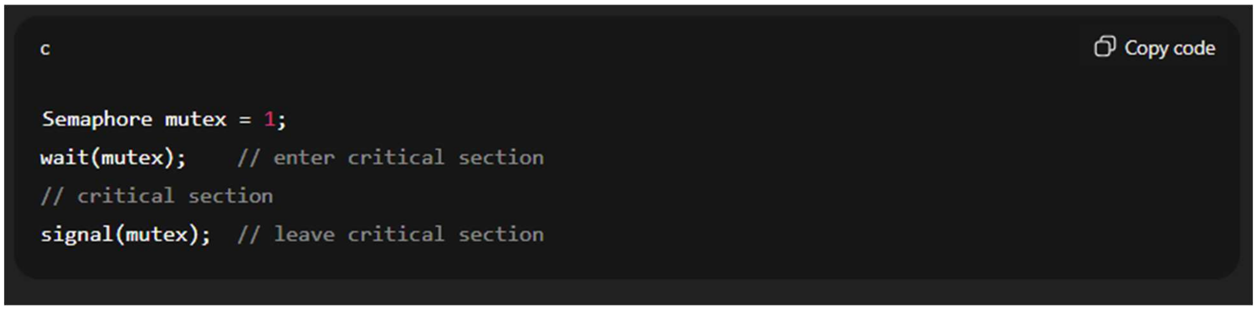
3. Semaphores

- **Semaphore:** A synchronization tool proposed by **Dijkstra**.
- Integer variable with two atomic operations:
    - wait(S) or P(S) → decreases S, if S < 0 process waits.
    - signal(S) or V(S) → increases S, wakes up waiting process.

Types of semaphores:

1. **Counting semaphore** → unrestricted range (resource counting).
2. **Binary semaphore (mutex)** → only 0 or 1 (mutual exclusion).

Example:



```c
Semaphore mutex = 1;
wait(mutex);    // enter critical section
// critical section
signal(mutex);  // leave critical section
```

4. Classical Problems of Synchronization

1. **Bounded-Buffer Problem (Producer–Consumer Problem):**
    - Producer puts items in buffer, consumer takes out.
    - Use semaphore for empty, full, and mutex.
2. **Readers–Writers Problem:**
    - Multiple readers allowed simultaneously, but only one writer at a time.
3. **Dining Philosophers Problem:**
    - Philosophers alternate between thinking and eating.
    - Need two chopsticks → possible deadlock.
    - Solved using semaphores/monitors with resource ordering.

5. Critical Regions & Monitors

- **Critical Regions:** High-level construct where compiler ensures mutual exclusion automatically.
- **Monitor:**
    - A synchronization construct that allows processes to have **mutual exclusion** inside a monitor.
    - Provides **condition variables** with wait() and signal().
    - Easier to program than semaphores (less error-prone).

---

6. Deadlocks

a. System Model

- **Deadlock** = A situation where a set of processes are blocked because each is holding a resource and waiting for another.

Example:
P1 holds R1, waiting for R2;
P2 holds R2, waiting for R1 → Deadlock.

---

b. Necessary Conditions for Deadlock (Coffman's conditions)

1. **Mutual Exclusion** – at least one resource not shareable.
2. **Hold and Wait** – process holding a resource also waits for another.
3. **No Preemption** – resource cannot be forcibly taken.
4. **Circular Wait** – circular chain of processes waiting for resources.

👉 If all four hold → **deadlock possible**.

c. Deadlock Characterization

- Represented using **Resource Allocation Graph (RAG):**
    - Nodes = Processes & Resources.
    - Edges = Allocation & Request.
- Deadlock exists if there is a **cycle** in RAG (for single instance resources).

7. Deadlock Handling Techniques

A. Deadlock Prevention

- Eliminate one of the four conditions:
    1. **Mutual Exclusion** – not possible for non-sharable resources.
    2. **Hold and Wait** – require processes to request all resources at once.
    3. **No Preemption** – preempt resource if process waits.
    4. **Circular Wait** – impose ordering of resource types.

B. Deadlock Avoidance

- Requires knowledge of future requests.
- **Banker's Algorithm (by Dijkstra):**
    - Works like a bank loan system.
    - A request is granted only if system remains in **safe state**.
    - **Safe State:** there exists a sequence in which all processes can finish without deadlock.

C. Deadlock Detection & Recovery

1. **Detection:** Use algorithms to detect cycles in RAG.
2. **Recovery:**
    - Process termination (abort one or more processes).
    - Resource preemption (take resources from some processes).

D. Combined Approach

- Use a mix of **prevention + detection + avoidance** depending on system requirements.

☑ **Quick Revision Notes**

- **Critical Section Problem** → solved using hardware, semaphores, monitors.
- **Synchronization Tools** → Semaphores (counting, binary), Monitors.
- **Classical Problems** → Producer–Consumer, Readers–Writers, Dining Philosophers.
- **Deadlock** → Needs 4 Coffman’s conditions.
- **Handling Deadlock** → Prevention (eliminate conditions), Avoidance (Banker’s Algorithm), Detection & Recovery.

Unit IV – Storage Management

Storage management: Memory Management-Logical and Physical Address Space, Swapping, Contiguous Allocation, Paging, Segmentation with paging in MULTICS and Intel 386, Virtual Memory, Demand paging and its performance, Page replacement algorithms, Allocation of frames, Thrasing, Page Size and other considerations, Demand segmentation, File systems, secondary Storage Structure, File concept, access methods, directory implementation, Efficiency and performance, recovery, Disk structure, Disk scheduling methods, Disk management, Recovery, Disk structure, disk scheduling methods, Disk management, Swap-Space management, Disk reliability

1. Memory Management

a. Logical and Physical Address Space

- **Logical Address (Virtual Address):** generated by CPU (seen by process).
- **Physical Address:** location in physical memory (RAM).
- **Memory Management Unit (MMU):** hardware device that maps logical → physical addresses.

👉 User program deals only with **logical addresses**; OS + MMU translates them into **physical addresses**.

---

b. Swapping

- A process can be temporarily **swapped out** of main memory to **backing store (disk)** and later brought back.
- Allows better CPU utilization and multiprogramming.
- **Limitation:** High **swap time** (depends on process size).

---

c. Contiguous Memory Allocation

- Memory divided into partitions, each process allocated a contiguous block.
- **Two placement strategies:**
    1. **First Fit** – allocate the first hole large enough.
    2. **Best Fit** – allocate the smallest hole that fits.
    3. **Worst Fit** – allocate the largest hole.

Problems:

- **External Fragmentation** – free memory exists but is non-contiguous.
- **Internal Fragmentation** – unused space inside allocated block.

---

d. Paging

- Physical memory divided into fixed-size **frames**.
- Logical memory divided into fixed-size **pages** (same size as frames).
- OS maintains **Page Table** for mapping pages → frames.

👉 Eliminates **external fragmentation**, but may cause **internal fragmentation** (last frame partially filled).

---

e. Segmentation

- Memory divided into variable-sized **segments** (logical units like code, stack, data).
- Each segment has a **base** and **limit** stored in **segment table**.
- Provides **logical view of program**, unlike paging which is purely physical.

---

f. Segmentation with Paging (MULTICS, Intel 386)

- Hybrid scheme: segments are divided into pages.
- Combines benefits:
    - Segmentation → user view.
    - Paging → avoids external fragmentation.

---

2. Virtual Memory

- Technique that allows execution of processes not completely in memory.
- Logical address space > physical memory.

a. Demand Paging

- Pages loaded into memory **only when needed**.
- **Page fault:** occurs when page not in memory.

Steps in Demand Paging:

1. CPU references a page.
2. If page not in memory → page fault trap.
3. OS loads required page from disk to RAM.
4. Instruction restarts.

---

b. Page Replacement Algorithms

When no free frame is available, OS replaces one.

1. **FIFO (First In First Out):** oldest page replaced.
2. **Optimal (OPT):** replace page not used for longest time in future (theoretical).
3. **LRU (Least Recently Used):** replace least recently used page.
4. **Clock Algorithm:** circular list with reference bits.

☞ **Performance measure = Page Fault Rate.**

---

c. Frame Allocation

- Decide how many frames to allocate to each process.
- Strategies:
    - **Equal allocation** – equal frames to all.
    - **Proportional allocation** – based on process size.
    - **Priority allocation** – based on process priority.

d. Thrashing

- When CPU spends more time **swapping pages** than executing processes.
- Caused by **over-allocation of processes** beyond memory capacity.
- **Solution:** Working Set Model, Page Fault Frequency control.

---

e. Page Size Considerations

- **Small Page Size** → less internal fragmentation, but larger page table.
- **Large Page Size** → smaller page table, but higher internal fragmentation.
- OS selects **optimal page size** based on workload.

---

3. File System Management

a. File Concept

- File: a collection of related data stored on secondary storage.
- **File attributes:** name, type, size, location, protection, timestamps.

b. Access Methods

1. **Sequential Access** – read/write in order.
2. **Direct Access (Random)** – jump to any record.
3. **Indexed Access** – use index for fast lookup.

---

c. Directory Implementation

- **Single-level:** one directory for all users.
- **Two-level:** separate directory for each user.
- **Tree-structured:** hierarchy with subdirectories.
- **Acyclic graph:** allows shared files.
- **General graph:** supports symbolic links.

d. File System Efficiency & Performance

- Caching, buffering, and indexing improve performance.
- Recovery techniques handle crashes (journaling, checkpoints).

---

4. Secondary Storage Structure

a. Disk Structure

- Disks consist of **platters, tracks, sectors**.
- Access time = **Seek Time + Rotational Latency + Transfer Time**.

b. Disk Scheduling Algorithms

1. **FCFS (First Come First Serve)** – simple, fair, but poor performance.
2. **SSTF (Shortest Seek Time First)** – chooses nearest request.
3. **SCAN (Elevator Algorithm)** – moves in one direction then reverses.
4. **C-SCAN (Circular SCAN)** – only one-way sweep for fairness.
5. **LOOK & C-LOOK** – optimized versions of SCAN/C-SCAN (don't go to end if not needed).

---

c. Disk Management

- **Partitioning** – dividing disk into logical sections.
- **Formatting** – creating file system.
- **Swap-Space Management** – disk space for swapping pages.

d. Disk Reliability

- Errors handled using **RAID (Redundant Array of Independent Disks):**
    - **RAID 0:** Striping (no redundancy).
    - **RAID 1:** Mirroring.
    - **RAID 5:** Block-level striping with parity.

☑ **Quick Revision (Exam-Oriented)**

- **Memory Mgmt:** Paging (no external frag), Segmentation (logical view), Hybrid (MULTICS, Intel 386).
- **Virtual Memory:** Demand Paging, Page Replacement (FIFO, OPT, LRU, Clock).
- **Thrashing:** CPU busy swapping pages.
- **File Mgmt:** Access methods → Sequential, Direct, Indexed. Directory structures → Single, Two, Tree.
- **Disk Scheduling:** FCFS, SSTF, SCAN, C-SCAN, LOOK.
- **RAID:** Increases disk reliability & performance.

Unit V – Security & Case Study

Security & Case Study: Protection and Security-Goals of protection, Domain of protection, Access matrix, Implementation of access Matrix, Revocation of Access Rights, language based protection, The Security problem, Authentication, One Time passwords, Program threats, System threats, Threat Monitoring, Encryption. Windows NT-Design principles, System components, Environmental subsystems, File system, Networking and program interface, Linux system-design principles, Kernel Modules, Process Management, Scheduling, Memory management, File Systems, Input and Output, Interprocess communication, Network structure, security

1. Protection and Security

a. Goals of Protection

- Ensure **controlled access** to resources (CPU, files, memory, I/O devices).
- Prevent **malicious or accidental misuse** of resources.
- Maintain **system integrity and user privacy**.

👉 Protection = internal control of system resources.
👉 Security = protection + defense against external threats.

---

b. Domain of Protection

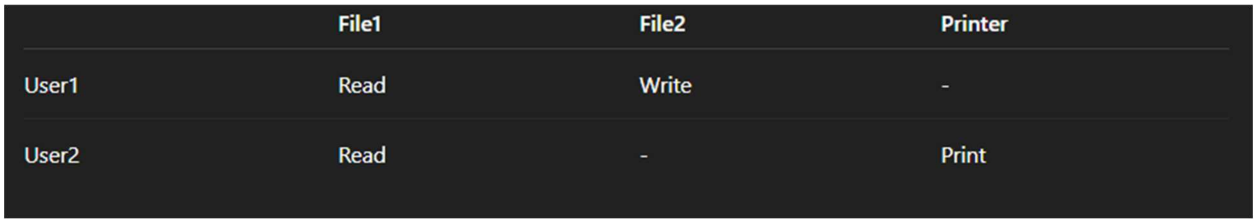
- **Domain** = set of access rights.
- Each user/process operates within a **domain**.
- Access rights specify:
  - **<object, rights>** (e.g., File F1 → read, write).

---

c. Access Matrix

- General model of protection.
- **Rows → subjects (users/processes)**.
- **Columns → objects (files, devices)**.
- **Entries → rights (read/write/execute)**.

Example:



	File1	File2	Printer
User1	Read	Write	-
User2	Read	-	Print

- **Implementation:**
    - **Access Control List (ACL):** store rights per object.
    - **Capability List:** store rights per subject.

---

d. Revocation of Access Rights

- **Immediate vs Delayed revocation**.
- **Selective revocation** (for one user) vs **General revocation** (all users).
- **Temporary revocation** (time-bound).

---

e. Language-Based Protection

- Programming language can enforce protection (e.g., Java sandbox, memory-safe languages).
- Restricts illegal memory access.

---

2. Security

a. Security Problem

- Protecting data and resources against **unauthorized access** and **system threats**.
- Balance **usability vs security**.

b. Authentication

- Verifying identity of users.
- **Methods:**
    1. **Password-based** (simple but vulnerable).
    2. **Biometric** (fingerprint, iris scan).
    3. **Token-based** (smart cards, OTPs).

---

c. One-Time Passwords (OTPs)

- Password valid only for a single session.
- Methods:
    1. **Random number generators**.
    2. **Secret key + sequence number**.
    3. **Time-based OTP (TOTP)**.

---

d. Program Threats

1. **Trojan Horse** – malicious code hidden in useful program.
2. **Trapdoor** – secret entry point bypassing normal authentication.
3. **Logic Bomb** – triggers malicious action under certain conditions.
4. **Virus** – attaches to files/programs, spreads by execution.
5. **Worm** – self-replicates across network.

---

e. System Threats

1. **Denial of Service (DoS):** overload system resources → crash.
2. **Distributed DoS (DDoS):** multiple systems attack simultaneously.
3. **Port Scanning:** probing for weaknesses.
4. **Spoofing:** fake identity to gain access.

f. Threat Monitoring

- Tools to detect suspicious activity:
    - **Audit logs** (record system activities).
    - **Intrusion Detection Systems (IDS)**.
    - **Antivirus & Firewalls**.

---

g. Encryption

- Process of **converting plaintext → ciphertext**.
- Ensures confidentiality and integrity.

Types:

1. **Symmetric Encryption** – same key for encryption & decryption (e.g., AES, DES).
2. **Asymmetric Encryption** – public key & private key (e.g., RSA).

---

3. Case Studies

a. Windows NT

- **Design Principles:** portability, extensibility, security.
- **System Components:** kernel, executive services, HAL (Hardware Abstraction Layer).
- **Environmental Subsystems:** support for multiple OS environments (Win32, POSIX).
- **File System:** NTFS (New Technology File System).
- **Networking:** integrated TCP/IP support.
- **Program Interface:** Windows API.

b. Linux

- **Design Principles:** open-source, modular, multiuser, multitasking.
- **Kernel Modules:** dynamically loadable parts (device drivers, file systems).
- **Process Management:** fork(), exec(), scheduling policies (CFS – Completely Fair Scheduler).
- **Memory Management:** paging, demand paging, swapping.
- **File Systems:** ext2/ext3/ext4, journaling support.
- **I/O System:** device drivers, buffer cache.
- **Interprocess Communication (IPC):** pipes, message queues, shared memory.
- **Networking:** full TCP/IP stack, socket interface.
- **Security:** user IDs, group IDs, file permissions, SELinux.

☑ Quick Revision Points

- **Protection vs Security:** protection = internal, security = external + protection.
- **Access Matrix:** basis of protection → ACL & Capability list.
- **Authentication:** passwords, biometrics, OTP.
- **Threats:** Program threats (virus, worm, Trojan), System threats (DoS, spoofing).
- **Encryption:** Symmetric (AES, DES), Asymmetric (RSA).
- **Windows NT:** NTFS, subsystems, portability.
- **Linux:** open-source, modular, ext file systems, CFS scheduler, SELinux.