TECHNICAL REPORT WRITING

File System and I/O Management
Security and Protection

NAME: RUPAK SARKAR

ROLL NO.: 14271024036

STREAM: MASTER OF COMPUTER APPLICATION

SUBJECT NAME: OPERATING SYSTEM

SUBJECT CODE: MCAN-202

COLLEGE NAME: MEGHNAD SAHA INSTITUTE OF TECHNOLOGY

UNIVERSITY NAME: MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

ABSTRACT

File systems and I/O management are essential for efficient data storage, retrieval, and processing in modern computing. However, security challenges such as unauthorized access, data breaches, and malware threats pose significant risks. This report explores key security mechanisms, including access controls, encryption, and intrusion detection, to protect file systems and ensure secure I/O operations. It also examines emerging trends like AI-driven threat detection and blockchain-based security. By implementing robust protection strategies, organizations can safeguard data integrity and enhance system resilience against cyber threats.

CONTENTS

INTRODUCTION

File systems and I/O (Input/Output) management are fundamental components of modern computing, responsible for organizing, storing, and retrieving data efficiently. A file system provides a structured way to manage data on storage devices, ensuring seamless access and retrieval. I/O management, on the other hand, facilitates communication between hardware components, optimizing data transfer between memory, storage, and peripheral devices.

With the increasing reliance on digital systems, security and protection in file systems and I/O management have become critical concerns. Threats such as unauthorized access, data corruption, malware attacks, and system vulnerabilities can compromise data integrity and confidentiality. To address these risks, organizations implement security measures such as access control mechanisms, encryption, authentication protocols, and intrusion detection systems.

This report explores the principles of file systems and I/O management while examining the security challenges they face. It highlights various protection strategies and emerging technologies that enhance data security, ensuring system reliability and resilience against cyber threats.

TECHNICAL REPORT

File Concept

A **file** is a logical storage unit in an operating system (OS) used to store and manage data. It acts as a bridge between users and the system's storage, allowing data to be written, read, and modified efficiently. Files can contain various types of data, including text, images, programs, and system configurations.

A computer file is defined as a medium used for saving and managing data in the computer system. The data stored in the computer system is completely in digital format, although there can be various types of files that help us to store the data.

File systems are a crucial part of any operating system, providing a structured way to store, organize, and manage data on storage devices such as hard drives, SSDs, and USB drives. Essentially, a file system acts as a bridge between the operating system and the physical storage hardware, allowing users and applications to create, read, update, and delete files in an organized and efficient manner.

1. File Attributes

Each file in an OS has specific attributes that provide metadata about the file, such as:

- **Name**: The file's identifier.
- **Type**: Indicates whether the file is a text, binary, executable, etc.
- **Location**: The storage address where the file is saved.
- **Size**: The file's total data size in bytes.
- **Protection**: Access permissions defining who can read, write, or execute the file.
- **Timestamps**: Information about file creation, modification, and last access.

2. File Operations

An OS provides several operations for managing files, including:

- **Create**: Generating a new file.
- **Open**: Accessing an existing file for reading or writing.
- **Read**: Retrieving data from a file.
- **Write**: Modifying or adding new data to a file.
- **Close**: Terminating file access after use.
- **Delete**: Removing a file permanently from storage.

3. File System Structure

A file system organizes and manages files on a storage device. Common structures include:

- **Single-Level Directory**: A simple structure where all files are stored in a single directory.
- **Two-Level Directory**: Each user has a separate directory to avoid name conflicts.

- **Hierarchical Directory (Tree Structure)**: Files are organized into directories and subdirectories for better management.
- **Graph Directory**: Allows file sharing and shortcuts using acyclic or cyclic graphs.

4. File Access Methods

Different file access methods define how data is retrieved from files:

- **Sequential Access**: Data is accessed in a linear order, often used for text files.
- **Direct Access (Random Access)**: Any part of the file can be accessed directly, suitable for databases.
- **Indexed Access**: Uses an index to locate data quickly, improving retrieval efficiency.

5. File Protection and Security

To prevent unauthorized access and data corruption, operating systems enforce various security measures:

- **Access Control Lists (ACLs)**: Define user permissions for reading, writing, or executing files.
- **File Encryption**: Protects data by converting it into an unreadable format unless decrypted.
- **User Authentication**: Ensures that only authorized users can access sensitive files.

Fundamental File System Organization and Access Methods –

A **file system** is a crucial component of an operating system that manages how data is stored, organized, retrieved, and accessed on storage devices. The organization of a file system determines its efficiency in handling data, while access methods define how users and applications interact with files.

1. File System Organization

File systems are structured in different ways to manage files effectively. The common file system organizations include:

a. Sequential File Organization

- Data is stored in a continuous sequence, making it ideal for applications requiring linear access.
- Common in text files, log files, and batch processing systems.
- Efficient for sequential access but slow for random access.

b. Indexed File Organization

- Uses an index to store pointers to file locations, allowing faster access.
- Commonly used in databases and systems that require quick searching.
- Provides both sequential and random-access capabilities.

c. Hashed File Organization

- Uses a hash function to map file records to specific locations on disk.
- Efficient for direct access and searching but can suffer from collisions.
- Used in applications requiring rapid data retrieval, such as caching and real-time systems.

d. Clustered File Organization

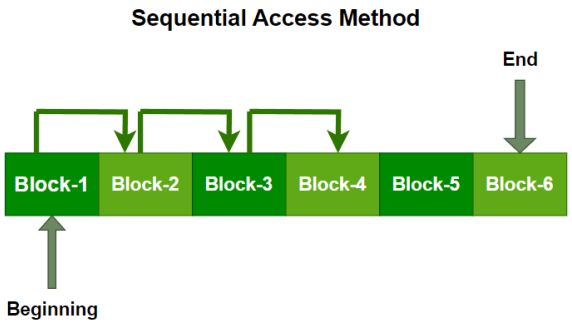
- Groups related files or data records in clusters for efficient retrieval.
- Reduces fragmentation and improves disk performance.

2. File Access Methods

Sequential Access

It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, the editor and compiler usually access the file in this fashion.

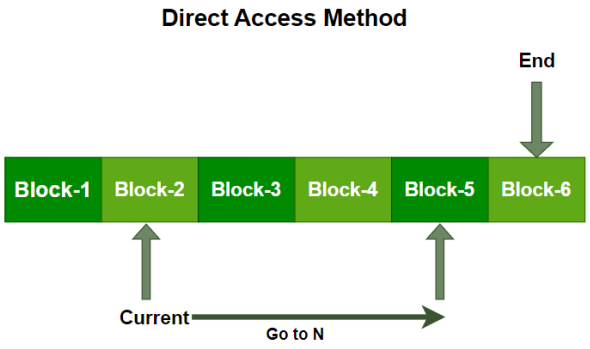
Read and write make up the bulk of the operation on a file. A read operation -read next- reads the next position of the file and automatically advances a file pointer, which keeps track of the I/O location. Similarly, for the -write next- append to the end of the file and advance to the newly written material.



Direct Access Method

Another method is direct access method also known as relative access method. A fixed-length logical record that allows the program to read and write record rapidly. in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59, and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.

A block number provided by the user to the operating system is normally a relative block number, the first relative block of the file is 0 and then 1 and so on.



File Directory Structure –

A **directory structure** is a way of organizing and managing files within an operating system. It helps users locate, store, and manage files efficiently by grouping them into directories and subdirectories. The structure of a file directory can significantly impact system performance, file accessibility, and security.

1. Types of File Directory Structures

a. Single-Level Directory

- All files are stored in a single directory.
- Simple and easy to implement but unsuitable for large systems.
- Name conflicts occur as all users share the same directory.
- Example: Early MS-DOS file systems.

b. Two-Level Directory

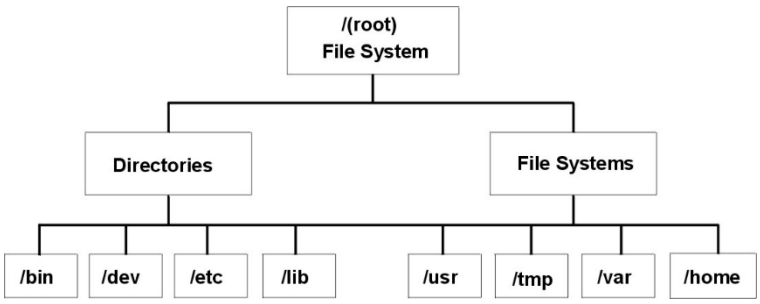
- Each user has a separate directory under the main directory (often called the "root").
- Prevents name conflicts between users.
- Limited scalability as users still have a flat structure.
- Example: UNIX systems with /home/username/ directories.

c. Tree-Structured Directory

- A hierarchical structure where directories can have subdirectories.
- Efficient for organizing large amounts of data.
- Allows easy navigation using **absolute** and **relative** paths.
- Example: Most modern operating systems (Windows, Linux, macOS).

File System Structure –

It is important to understand the difference between a file system and a directory. A file system is a section of hard disk that has been allocated to contain files. This section of hard disk is accessed by mounting the file system over a directory. After the file system is mounted, it looks just like any other directory to the end user.



File Allocation Methods –

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

1. Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

2. Linked List Allocation

In this scheme, each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk.

The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

3. Indexed Allocation

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Each file has its own index block. The i-th entry in the index block contains the disk address of the i-th file block.

Free Space Management –

Free space management is a critical aspect of operating systems as it involves managing the available storage space on the hard disk or other secondary storage devices. The operating system uses various techniques to manage free space and optimize the use of storage devices.

1. Bitmap or Bit vector

A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: 0 indicates that the block is free and 1 indicates an allocated block.

2. Linked List

In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

3. Grouping

This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks. An advantage of this approach is that the addresses of a group of free disk blocks can be found easily.

Directory Implementation –

Directory implementation in the operating system can be done using Singly Linked List and Hash table. The efficiency, reliability, and performance of a file system are greatly affected by the selection of directory-allocation and directory-management algorithms. There are numerous ways in which the directories can be implemented. But we need to choose an appropriate directory implementation algorithm that enhances the performance of the system.

Directory Implementation using Single Linked List

The implementation of directories using a singly linked list is easy to program but is time-consuming to execute. Here we implement a directory by using a linear list of filenames with pointers to the data blocks.

- To create a new file the entire list has to be checked such that the new directory does not exist previously.
- The new directory then can be added to the end of the list or at the beginning of the list.
- In order to delete a file, we first search the directory with the name of the file to be deleted. After searching we can delete that file by releasing the space allocated to it.
- To reuse the directory entry, we can mark that entry as unused or we can append it to the list of free directories.
- To delete a file linked list is the best choice as it takes less time.

Directory Implementation using Hash Table

An alternative data structure that can be used for directory implementation is a hash table. It overcomes the major drawbacks of directory implementation using a linked list. In this method, we use a hash table along with the linked list. Here the linked list stores the directory entries, but a hash data structure is used in combination with the linked list.

In the hash table for each pair in the directory key-value pair is generated. The hash function on the file name determines the key and this key points to the corresponding file stored in the directory. This method efficiently decreases the directory search time as the entire list will not be searched on every operation. Using the keys the hash table entries are checked and when the file is found it is fetched.

Efficiency and Performance in File Systems

The efficiency and performance of a file system determine how well it manages, retrieves, and stores data on storage devices. Optimizing these factors is essential for minimizing delays, reducing storage overhead, and improving system responsiveness.

Factors Affecting File System Performance

a. File Allocation Methods

- **Contiguous Allocation:** Fast access but causes fragmentation.
- **Linked Allocation:** No fragmentation but slow sequential access.
- **Indexed Allocation:** Balances performance but requires extra space for index blocks.

b. Disk Scheduling Algorithms

- **FCFS (First-Come, First-Served):** Simple but inefficient.
- **SSTF (Shortest Seek Time First):** Improves access time but can cause starvation.
- **SCAN & C-SCAN:** Reduces seek time by scanning in one or both directions.

PC Bus Structure

The **PC Bus Structure** refers to the internal communication system within a computer that connects the processor, memory, and peripheral devices. A **bus** is a set of parallel wires or traces that transmit data, addresses, and control signals between different components of the computer system.

Components of a PC Bus Structure

A typical PC bus system consists of three main types of buses:

a. Address Bus

- Carries memory addresses from the CPU to other components.
- Determines which memory location or I/O device should be accessed.
- **Unidirectional** (data flows in one direction, from CPU to memory or I/O).

b. Data Bus

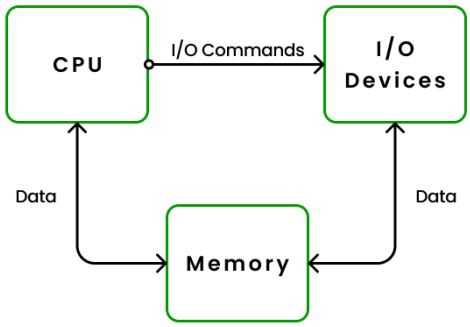
- Transfers actual data between CPU, memory, and I/O devices.
- **Bidirectional** (data flows both to and from the CPU).
- Width (e.g., **8-bit, 16-bit, 32-bit, 64-bit**) affects system performance.

c. Control Bus

- Sends control signals to manage operations (e.g., Read/Write signals).
- Coordinates access between the CPU and other components.
- Carries signals such as **interrupts, clock signals, and status flags**.

I/O Connections –

The foundation of efficient computing rests on robust interaction between users and an operating system through **Input/Output (I/O)** devices in today's world. The smooth functioning necessitates dependable exchange of data among keyboard/screen/mouse/printers/network adapters facilitated by such intermediary agents. Establishing consistent two-way communication channels between OS and all I/O devices is imperative to sustain a glitch-free experience.



I/O devices let users provide the operating system input and receive output from it. Users can issue commands, engage with programs, and traverse the system via input devices such as keyboards and mice. The user gets information, results, or visual feedback through output devices like monitors and printers. Users are able to successfully employ the features of a computer system through seamless interaction facilitated by communication between the operating system and peripheral devices.

Data Transfer Techniques –

Programmed I/O: It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually, the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Interrupt- initiated I/O: Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime, the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

Direct Memory Access: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus, we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

Bus Arbitration –

Bus Arbitration refers to the process by which the current bus master accesses and then leaves the control of the bus and passes it to another bus requesting processor unit. The controller that has access to a bus at an instance is known as a **Bus master**.

Applications of bus arbitration in computer organization:

Shared Memory Systems: In shared memory systems, multiple devices need to access the memory to read or write data. Bus arbitration allows multiple devices to access the memory without interfering with each other.

Multi-Processor Systems: In multi-processor systems, multiple processors need to communicate with each other to share data and coordinate processing. Bus arbitration allows multiple processors to share access to the bus to communicate with each other and with shared memory.

Blocking and Nonblocking IO –

Blocking and non-blocking IO are two different methods used by OS to handle I/O operations. A blocking call is where the OS decides that it needs to wait for a certain operation to complete before allowing the program to continue execution. The result of this is that user can't do anything else while waiting on a system call; if your program doesn't have any other activity, then it should be able to wait indefinitely without causing any problems.

On the other hand, non-blocking calls (also known as asynchronous operations) don't block the thread until they finish their work; instead, this type of system call returns immediately after completing its job without having any effect on what's happening in other threads.

Blocking System Call

A blocking system call is a system call that blocks the process execution until the requested operation is completed. The blocking system call can be used in synchronous I/O or asynchronous I/O.

Non-Blocking System Call

A non-blocking system call is a system call that does not block. The calling process can continue execution while the operation is in progress and returns immediately when it's complete.

The most important difference between blocking and non-blocking IO is how code behaves during the I/O operation: with a blocking IO, users must wait until data has been received before continuing execution; with a non-blocking IO, users don't have to wait for anything at all!

The main advantage of non-blocking IO is that it allows users to continue with other tasks while waiting for the I/O operation to complete. This can be especially useful when writing concurrent programs, where there are many things happening at once (like multiple threads).

Non-blocking IO is a bit harder to write than blocking IO, but it's also much more powerful and flexible. This is because non-blocking IO allows users to perform multiple I/O operations at once, in any order.

Kernel I/O Subsystem in Operating System –

The kernel provides many services related to I/O. Several services such as scheduling, caching, spooling, device reservation, and error handling – are provided by the kernel's I/O subsystem built on the hardware and device-driver infrastructure. The I/O subsystem is also responsible for protecting itself from errant processes and malicious users.

I/O Scheduling –

To schedule a set of I/O requests means to determine a good order in which to execute them. The order in which the application issues the system call is the best choice. Scheduling can improve the overall performance of the system, can share device access permission fairly to all the processes, and reduce the average waiting time, response time, and turnaround time for I/O to complete.

Buffering –

A buffer is a memory area that stores data being transferred between two devices or between a device and an application. Buffering is done for three reasons. The first is to cope with a speed mismatch between the producer and consumer of a data stream. The second use of buffering is to provide adaptation for data that have different data-transfer sizes.

Caching –

A cache is a region of fast memory that holds a copy of data. Access to the cached copy is much easier than the original file. For instance, the instruction of the currently running process is stored on the disk, cached in physical memory, and copied again in the CPU's secondary and primary cache.

The main difference between a buffer and a cache is that a buffer may hold only the existing copy of a data item, while a cache, by definition, holds a copy on faster storage of an item that resides elsewhere.

Spooling and Device Reservation –
A spool is a buffer that holds the output of a device, such as a printer that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixes together.

The OS solves this problem by preventing all output from continuing to the printer. The output of all applications is spooled in a separate disk file. When an application finishes printing then the spooling system queues the corresponding spool file for output to the printer.

Error Handling –
An OS that uses protected memory can guard against many kinds of hardware and application errors so that a complete system failure is not the usual result of each minor mechanical glitch, Devices, and I/O transfers can fail in many ways, either for transient reasons, as when a network becomes overloaded or for permanent reasons, as when a disk controller becomes defective.

Case Study: Unix File System vs. Windows File System

The **Unix File System (UFS)** and **Windows File System (NTFS, FAT32, exFAT)** are two widely used file systems, each with distinct design principles, structures, and functionalities. This case study compares their architecture, features, and performance.

1. Unix File System (UFS & ext4)

Overview:

- Used in Linux and Unix-based operating systems.
- Supports hierarchical directory structures and advanced file permissions.
- Common Unix file systems: UFS, ext3, ext4, XFS, ZFS.

Key Features:

- Inode-based structure – Uses inodes to store metadata (file size, owner, permissions).
- Journaling – Ext3, Ext4, and XFS include journaling to prevent data corruption.
- Symbolic & Hard Links – Supports different linking mechanisms.
- Permissions & Ownership – Uses read, write, execute permissions for users, groups, and others.

Advantages:

- Highly secure and stable.
- Efficient handling of large files.
- Supports multiple users and strong access control.

Disadvantages:

- More complex administration compared to Windows.
- Limited support for Windows-based applications.

2. Windows File System (NTFS, FAT32, exFAT)

Overview:

- Used in Microsoft Windows operating systems.
- Supports user-friendly file and folder management.
- Common Windows file systems: NTFS, FAT32, exFAT.

Key Features:

- NTFS (New Technology File System) – Provides advanced security and reliability.
- File Compression – Built-in file compression for space-saving.
- Access Control Lists (ACLs) – Fine-grained file permissions.
- File Encryption – Supports BitLocker encryption for data protection.
- Faster File Indexing – Uses the Master File Table (MFT) for efficient searches.

Advantages:

- User-friendly and widely supported.
- Better integration with Windows-based applications.
- More advanced features (journaling, encryption, ACLs) compared to FAT32.

Disadvantages:

- Less efficient for multi-user environments compared to Unix.
- More vulnerable to malware and fragmentation.

Security and Protection:

Overview of Security and Protection in Operating Systems

1. Introduction

Security and protection in an Operating System (OS) ensure the confidentiality, integrity, and availability of system resources. The OS provides mechanisms to prevent unauthorized access, protect user data, and safeguard system resources from threats such as malware, unauthorized users, and system failures.

Security – The security systems cover the safety of their system resources (saved data, memory, disks, etc) across malignant alteration, illegal access, and disparity or inconsistency. The security gives a mechanism (authentication and encryption) to analyze the user to permit for using the system.

Protection – The protection deals with access to the system resources. It determines what files can be accessed or permeated by a special user. The protection of the system should confirm the approval of the process and users. Due to this, these licensed users and processes will care for the central processing unit, memory, and alternative sources. The protection mechanism ought to provide a path for specifying the controls to be obligatory, besides how to implement them.

Goals of Security System –

- **Integrity:** Users with insufficient privileges should not alter the system's vital files and resources, and unauthorized users should not be permitted to access the system's objects.
- **Secrecy:** Only authorized users must be able to access the objects of the system. Not everyone should have access to the system files.
- **Availability:** No single user or process should be able to eat up all of the system resources; instead, all authorized users must have access to them. A situation like this could lead to service denial. Malware in this instance may limit system resources and prohibit authorized processes from using them.

Security Attacks –

1. Worm:
An infection program that spreads through networks. Unlike a virus, they target mainly LANs. A computer affected by a worm attacks the target system and writes a small program "hook" on it. This hook is further used to copy the worm to the target computer. This process repeats recursively, and soon enough all the systems of the LAN are affected. It uses the spawn mechanism to duplicate itself. The worm spawns copies of itself, using up a majority of system resources and also locking out all other processes.

2. Port Scanning:
It is a means by which the cracker identifies the vulnerabilities of the system to attack. It is an automated process that involves creating a TCP/IP connection to a specific port. To protect the identity of the attacker, port scanning attacks are launched from Zombie Systems, that is systems that were previously independent systems that are also serving their owners while being used for such notorious purposes.

3. Denial of Service:
Such attacks aren't aimed for the purpose of collecting information or destroying system files. Rather, they are used for disrupting the legitimate use of a system or facility.

Formal and Practical aspects of Security –

1. Formal Aspects of Security

- To formally prove a system is secure, we need:
- A security model comprising security policies and mechanisms
- A list of threats
- A list of fundamental attacks
- A proof methodology
- Manual procedures can discover security flaws
- But procedures become less reliable as systems grow
- Formal approach constructs feasible sequences of operations and deduces their consequences

2. Practical Aspects of Security

- User authentication – Passwords, biometrics, multi-factor authentication (MFA).
- Role-Based Access Control (RBAC) – Assigns permissions based on job roles.
- Antivirus & Anti-malware – Detects and removes malicious software.
- Security Auditing & Logging – Tracks user activities for forensic analysis.
- Firewalls & Intrusion Detection Systems (IDS) – Monitor and block threats.
- Encryption & Secure Communication (SSL/TLS) – Protects data in transit.
- Patching & Updates – Fixes vulnerabilities to prevent exploits.

Encryption –

Data encryption is the process of converting readable information (plaintext) into an unreadable format (ciphertext) to protect it from unauthorized access. It is a method of preserving data confidentiality by transforming it into ciphertext, which can only be decoded using a unique decryption key produced at the time of the encryption or before it. The conversion of plaintext into ciphertext is known as encryption. By using encryption keys and

mathematical algorithms, the data is scrambled so that anyone intercepting it without the proper key cannot understand the contents.

When the intended recipient receives the encrypted data, they use the matching decryption key to return it to its original, readable form. This approach ensures that sensitive information such as personal details, financial data, or confidential communications remains secure as it travels over networks or is stored on devices.

Key Objective of Encryption Data

- **Confidentiality:** Encryption ensures that only authorized parties can get access to data and recognize the information.
- **Data Integrity:** Encryption can also provide data integrity by making sure that the encrypted data remains unchanged during transmission. Any unauthorized changes to the encrypted information will render it undecipherable or will fail integrity checks.
- **Authentication:** Encryption may be used as part of authentication mechanisms to verify the identification of the communication party.
- **Non-Repudiation:** Through encryption, events can make sure that they cannot deny their involvement in growing or sending a selected piece of data.

Authentication –

Authentication is the process of verifying a user's identity before granting access to a system. Password security is a key aspect of authentication, ensuring that only authorized users can access system resources. Strong authentication mechanisms prevent unauthorized access, data breaches, and identity theft.

1. Single-Factor authentication: This was the first method of security that was developed. On this authentication system, the user has to enter the username and the password to confirm whether that user is logging in or not. Now if the username or password is wrong, then the user will not be allowed to log in or access the system.

Advantage of the Single-Factor Authentication –

- It is a very simple to use and straightforward system.
- it is not at all costly.
- The user does not need any huge technical skills.

2. Two-factor Authentication: In this authentication system, the user has to give a username, password, and other information. There are various types of authentication systems that are used by the user for securing the system. Some of them are: – wireless tokens and virtual tokens. OTP and more.

Advantages of the Two-Factor Authentication –

- The Two-Factor Authentication System provides better security than the Single-factor Authentication system.
- The productivity and flexibility increase in the two-factor authentication system.
- Two-Factor Authentication prevents the loss of trust.

Passwords: Password verification is the most popular and commonly used authentication technique. A password is a secret text that is supposed to be known only to a user. In a password-based system, each user is assigned a valid username and password by the system administrator.

Access Descriptors and the Access Control Matrix –

Access Descriptor

An access descriptor holds essential information about the structure and attributes of the DBMS table or view you want to access, such as DBMS connection information, the name of the database, the table name and column names, and data types.

Access Matrix

An **Access Matrix** is a digital model utilized to control and manage permissions. This model defines the rights each user has for different resources. In simple terms, it's a table that shows what actions an individual or a group of users can perform on specific objects within a system. It represents the access control mechanism that specifies which actions (e.g., read, write, execute) are allowed or denied for each subject on each object.

Different Types of Rights

There are different types of rights the files can have. The most common ones are:

- *Read*- This is a right given to a process in a domain that allows it to read the file.
- *Write*- Process in the domain can be written into the file.
- *Execute*- The process in the domain can execute the file.
- *Print*- Process in the domain only has access to a printer.

Protection Structures

In an operating system (OS), "protection structures" refer to mechanisms that control access to system resources like files, memory, and devices, ensuring only authorized users or processes can access them, thereby maintaining data confidentiality, integrity, and availability; key aspects of protection structures include access control lists, user permissions, memory segmentation, and privileged instructions, all managed by the OS kernel to prevent unauthorized access.

Goal:

To prevent unauthorized access to system resources, protecting data from corruption and ensuring proper system operation.

Key elements:

- *Access control lists (ACLs):* Define which users or groups have specific read, write, and execute permissions on files or system resources.
- *User/Group IDs:* Identifiers associated with each user, allowing the system to differentiate access levels based on identity.
- *Protection domains:* Logical boundaries that separate different parts of the system, restricting communication between them.
- *Kernel mode:* A privileged execution state where the OS can access sensitive hardware and memory areas, not accessible to user applications.
- *Memory segmentation:* Dividing memory into distinct segments for each process, preventing one process from accessing another's memory space.

CONCLUSION

File Systems and I/O Management, along with Security and Protection, are critical components of modern operating systems, ensuring efficient data storage, seamless hardware communication, and robust system security.

File systems provide structured storage, retrieval, and management of data using various allocation and directory structures. Efficient I/O management techniques, such as buffering, caching, and scheduling, enhance system performance while data transfer methods like DMA and interrupt-driven I/O optimize resource utilization.

Security and protection mechanisms safeguard the system from unauthorized access, cyber threats, and data breaches. Access control models, encryption, authentication, and intrusion detection systems ensure the confidentiality, integrity, and availability of system resources. By implementing robust security policies and efficient file and I/O management strategies, operating systems can provide **reliable, secure, and high-performance computing environments** for users and applications.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who contributed to the completion of this technical report on File Systems and I/O Management, Security and Protection.

First and foremost, I extend my heartfelt thanks to my teacher, Mrs. Aparna Datta for her valuable guidance, support, and insightful feedback, which helped shape the direction of this report. Their expertise provided a strong foundation for understanding the complexities of operating systems.

Finally, I am grateful to my friends, classmates, and family members for their constant encouragement and motivation throughout this process. Their support has been invaluable in ensuring the successful completion of this report.

Thank you all for your contributions and assistance.