UNIT – 5 : PROTECTION AND CASE STUDIES

PART – 1 : PROTECTION

Protection:

A mechanism that controls the access of programs, processes, or users to the resources defined by a computer system is referred to as protection. You may utilize protection as a tool for multi-programming operating systems, allowing multiple users to safely share a common logical namespace, including a directory or files.

Protection tackles the system's internal threats. It provides a mechanism for controlling access to processes, programs, and user resources. In simple words, It specifies which files a specific user can access or view and modify to maintain the proper functioning of the system. It allows the safe sharing of **common physical address space** or **common logical address space** which means that multiple users can access the memory due to the physical address space.

Let's take an **example** for a better understanding, suppose In a small organization there are four employees p1, p2, p3, p4, and two data resources r1 and r2. The various departments frequently exchange information but not sensitive information between all employees. The employees p1 and p2 can only access r1 data resources and employees p3 and p4 can only access r2 resources. If the employee p1 tries to access the data resource r2, then the employee p1 is restricted from accessing that resource. Hence, p1 will not be able to access the r2 resource.

Need of protection:

- To ensure data safety, process and program safety against illegal user access, or even program access, we need protection.
- It is to ensure that programs, resources and data are accessed only according to the systems' policies.
- It is also to ensure that there are no access rights' breach, no unauthorized access to the existing data, no virus or worms.
- There can be security threats such as unauthorized reading, writing, modification or preventing the system to work properly for the authorized users themselves.

Goals of Protection:

- Therefore, protection is a method of safeguarding data and processes against malicious and intentional intrusion. For that purpose, we have protection policies that are either designed by the system itself or specified by the management itself or are imposed by the programmers individually to protect their programs with extra safety.
- It also gives a multiprogramming OS the sense of safety that is required by its users to share common space like files or directories.
- The policies bind how the processes are to access the resources present in the computer system, resources like CPU, memory, software and even the OS. Both the OS designer and the application programmer are responsible for this. However, these policies always change from time to time.

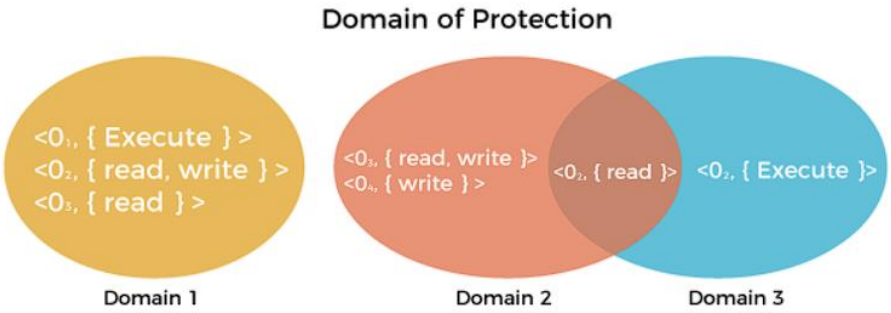
Principles of Protection:

- The ***principle of least privilege*** dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program needs special privileges to perform a task, it is better to make it a SGID program with group ownership of "network" or "backup" or some other pseudo group, rather than SUID with root ownership. This limits the amount of damage that can occur if something goes wrong.
- Typically, each user is given their own account, and has only enough privilege to modify their own files.
- The root account should not be used for normal day to day activities - The System Administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges

Domain of Protection:

- The protection policies limit the access of each process with respect to their resource handling. A process is bound to use only those resources which it requires to complete its task, in the time limit that it requires and also the mode in which it is required. That is the protected domain of a process.
- A computer system has processes and objects, which are treated as abstract data types, and these objects have operations specific to them. A domain element is described as **<object, {set of operations on object}>.**

- Each domain consists of a set of objects and the operations that can be performed on them. A domain can consist of either only a process or a procedure or a user. Then, if a domain corresponds to a procedure, then changing domain would mean changing procedure ID. Objects may share a common operation or two. Then the domains overlap.



Association Between Process and Domain:

Processes switch from one domain to other when they have the access right to do so. It can be of two types as follows.

1. **Fixed or static** – In fixed association, all the access rights can be given to the processes at the very beginning but that give rise to a lot of access rights for domain switching. So, a way of changing the contents of the domain are found dynamically.
2. **Changing or dynamic** – In dynamic association where a process can switch dynamically, creating a new domain in the process, if need be.

Access Matrix:

Access Matrix is a security model of the protective state of a computer system. For each object, the permissions for every process executing in the domain are specified using an **access matrix**.

Access matrix in os is shown as a **two-dimensional matrix**, where the columns of the matrix represent **objects**, while the rows represent **domains**. Each matrix cell represents a specific set of access rights that are granted to processes of the domain this indicates that each entry access (i, j) specifies the set of actions that a process executing in domain Di may invoke on object Oj. The access matrix implements policy decisions and these policy decisions involve which rights should be included in the (i, j)th entry like reading, writing, and executing. (This policy is usually decided by the operating system).

Components of an Access Matrix:

The two-dimensional Access matrix works on three components:

- **Subjects:** Rows of the matrix represent subjects that perform some operation on objects. From the perspective of operating systems, subjects are usually users or processes.

- **Objects:** objects are the resources that the subject accesses to perform some operations, and matrix columns represent objects. Objects are typically files, devices, or any other system resources.

- **Rights:** Rights are the permissions that specify the operations or actions a subject can perform on an object. The matrix cell depicting access rights to perform operations on objects makes the subject-object pair; subjects can read, write, perform operations, execute, etc.

Different types of rights:

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

1. Read- This is a right given to a process in a domain, which allows it to read the file.
2. Write- Process in domain can write into the file.
3. Execute- Process in domain can execute the file.
4. Print- Process in domain only has access to printer.

Sometimes, domains can have more than one right, i.e. combination of rights mentioned above.

Let us now understand how an access matrix works from the example given below.

	F1	**F2**	**F3**	**Printer**
D1	read		read	
D2				print
D3		read	execute	
D4	read write		read write	

Observations of above matrix:

- There are **four domains** and **four objects**– three files(F1, F2, F3) and one printer.
- A process executing in D1 can read files F1 and F3.

- A process executing in domain D4 has same rights as D1 but it can also write on files.
- Printer can be accessed by only one process executing in domain D2.
- A process executing in domain D3 has the right to read file F2 and execute file F3.

Mechanism of Access Matrix:

The mechanism of access matrix consists of many policies and semantic properties. Specifically, we must ensure that a process executing in domain Di can access only those objects that are specified in row i. Policies of access matrix concerning protection involve which rights should be included in the **(i, j)th entry**. We must also decide the domain in which each process executes. This policy is usually decided by the operating system. The users decide the contents of the access-matrix entries. Association between the domain and processes can be either static or dynamic. Access matrix provides a mechanism for defining the control for this association between domain and processes.

Switch operation: When we switch a process from one domain to another, we execute a switch operation on an object(the domain). We can control domain switching by including domains among the objects of the access matrix. Processes should be able to switch from one domain (Di) to another domain (Dj) if and only if a switch right is given to access(i, j).

This is explained using an example below:

	F1	**F2**	**F3**	**Printer**	**D1**	**D2**	**D3**	**D4**
D1	read		read			switch		
D2				print			switch	switch
D3		read	execute					
D4	read write		read write		switch			

According to the above matrix,

- A process executing in domain D2 can switch to domain D3 and D4.

- A process executing in domain D4 can switch to domain D1

- and process executing in domain D1 can switch to domain D2.

Implementation of Access Matrix:

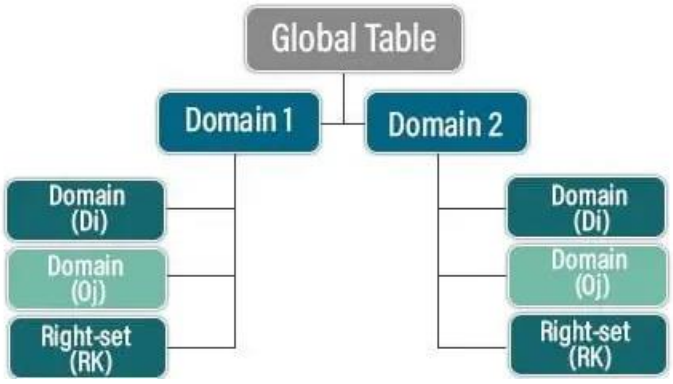
As already discussed, the access matrix in the operating system likely occupies a significant amount of memory and is very sparse. Therefore, it is storage inefficient to directly implement an access matrix for access control. The access matrix can be subdivided into rows or columns to reduce the inefficiency. To increase efficiency, the columns, and rows can be collapsed by deleting null values. **Four widely used access matrix implementations can be formed using these decomposition methods**:

- Global Table
- Capability Lists for Domains
- Access Lists for Objects
- Lock and key Method

1. Global Table:

This access control method is very simple; it maintains an ordered entry of domain, object, and rights in a file, i.e., <domain, object, right-set>, where the domain Di performs an operation M on the object Oj with the access right. The global table searches for the tripled entries <Domain(Di), Object(Oj), right-set(RK)>, and if the system locates these entries, the operation proceeds; otherwise, it throws an exception error, where M € Rk, which works great for smaller entries. The main drawback of this method is that it cannot handle large datasets with vast numbers of subject-object pairs, leading to scalability issues.

Tree Form of Global Table:



Explanation:

Here,

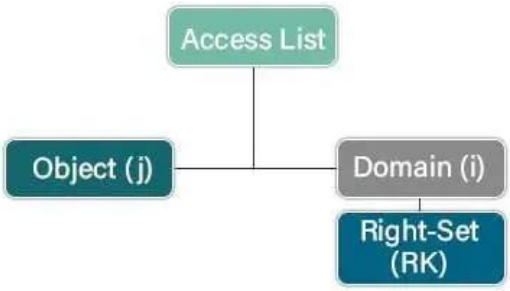
- **Domain:** represents subjects or users requesting access to objects
- **Objects:** represents the file or resources that the subject will access

- **Right-set:** means access right for the domain to perform operations on objects.
- Global table stores access rights associated with all subject-object pairs for Domain 1 and Domain 2
- Each entry represents the < Domain(Di), Object(Oj), right-set(RK)>. If operation M is present in the right set, it grants access and the operation proceeds.
- The exception raises otherwise.

2. Access Lists:

Access List maintains the list associated with each object that specifies the domain and the corresponding access rights of the object. Each entry in the access list has the domain identifier with its granted access right, and there is an ordered pair of <domain, right-set> in the access list for corresponding objects. Operation M, executed by domain Di on object Oj, searches for the entry <Domain(Di), right-set(RK)>, with M€RK in the object Oj access list. The operation proceeds or verifies the default set if the triple set is present. Access authorization occurs if "M" is found in the default set; otherwise, an exception triggers.

Tree Form of Access List:



Explanation:

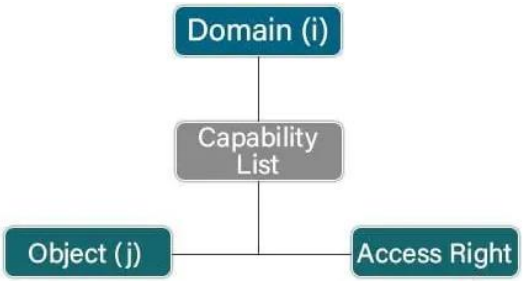
Here,

- **Objects:** represents the file or resources that the subject will access
- **Domain:** defines subjects or users requesting access to objects
- **Right-set:** means access right for the domain to perform operations on objects.
- It simply searches for the access list of object (Oj) to find the <Domain (Di), right-set(RK)>, with operation M€RK.
- The system verifies or searches for the default set upon discovering the domain identifier with the corresponding access rights set.

3. Capability Lists:

A Capability list outlines accessible objects and the operations that every domain can execute. It is performing Operation M on Object Oj. A process runs on operation M, which states the object Oj capability. When a subject possesses the specified object's capability, it grants access.

Tree Form of Capability List:



Explanation:

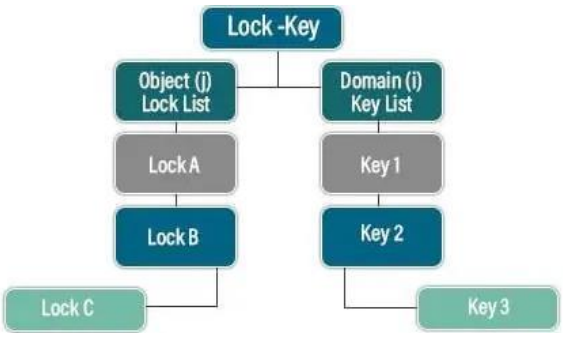
Here,

- **Objects:** represents the file or resources that the subject will access
- **Domain:** represents subjects or users requesting access to objects
- **Right-set:** represents access right for the domain to perform operations on objects.
- **Capability List:** A domain carries out access only if it possesses the capability for the object.

4. Lock-Key Mechanism:

The lock-key mechanism works by locks associating with objects and keys associating with subject-by-bit patterns. A domain can only access objects if it has key-fitting object locks. The process cannot make any key modifications. A domain (Di) process tries to access an object (Oj). If the key of the domain pairs with the lock of object Oj, then only a process can access the object.

Tree Form of Lock-Key Mechanism:



Explanation:

Here,

- **Objects Lock List:** Each has a list of object locks that corresponds to access permission
- **Domain Lock List:** Each holds a key that fits the object lock
- **Access:** Access is granted to the domain only if its key satisfies the lock requirement

Example:

Consider the below example for a better understanding of implementing an **access matrix in the operating system**.

Now, let's take an example when there are four files having the following access rights (files f1, f2, f3, and f4) and three domains (D1, D2, D3)

ACCESS MATRIX				
Object / Domain	File 1	File 2	File 3	File 4
D1	*read write*	*read*	*read*	*read*
D2	*read*	*write*	-	*exceute*
D3	*write*	-	*write*	*exceute*

The access matrix, as seen in the figure above, here represents the set of access rights as:

- Any file can be **read** by a process running in the D1 domain and can only be **written** into f1.
- A process in D2 domain has access to **read** f1, **write** to f2, and **execute** f4.
- D3 processes in the domain can **write** to f1 and 3 and can **execute** f4.

The **contents** of the access-matrix entries are normally decided by the users. The access matrix's Oj column is added when a user creates a new object, Oj, with the proper initialization entries as provided by the creator. Processes should be able to switch between different domains.

An access matrix can be used to specify and implement both **static and dynamic access rights**. When a process switches between different domains, it's similar to when we do a **(domain switch)** operation on an object (the domain). The table below demonstrates the possibility of domain switches from the D1 domain to D2, D2 to D3, and D2 to D1.

ACCESS MATRIX WITH DOMAIN AS OBJECTS						
Object / Domain	File 1	File 2	File 3	D1	D2	D3
D1	*read write*	*read*	*write*	-	*switch*	-
D2	*write*	*exceute*	*write*	*switch*	-	*switch*
D3	*read*	-	*exceute*	-	-	-

An **asterisk** (*) appended to the access right indicates the capability of copying an access right from one domain to another of the access matrix. The access right may only be copied within the column (i.e., for the object) for which it is defined by the copyright.

Access Control:

Access Control is a method of limiting access to a system or resources. Access control refers to the process of determining who has access to what resources within a network and under what conditions. It is a fundamental concept in security that reduces risk to the business or organization. Access control systems perform identification, authentication, and authorization of users and entities by evaluating required login credentials that may include passwords, pins, bio-metric scans, or other authentication factors. Multi-factor authentication requires two or more authentication factors, which is often an important part of the layered defense to protect access control systems.

Components of Access Control:

- **Authentication:** Authentication is the process of verifying the identity of a user. User authentication is the process of verifying the identity of a user when that user logs in to a computer system.
- **Authorization:** Authorization determines the extent of access to the network and what type of services and resources are accessible by the authenticated user. Authorization is the method of enforcing policies.
- **Access:** After the successful authentication and authorization, their identity becomes verified, This allows them to access the resource to which they are attempting to log in.
- **Manage:** Organizations can manage their access control system by adding and removing authentication and authorization for users and systems. Managing these systems can be difficult in modern IT setups that combine cloud services and physical systems.
- **Audit:** The access control audit method enables organizations to follow the principle. This allows them to collect data about user activities and analyze it to identify possible access violations.

How Access Control Works?

Access control involves determining a user based on their credentials and then providing the appropriate level of access once confirmed. Credentials are used to identify and authenticate a user include passwords, pins, security tokens, and even biometric scans. Multifactor authentication (MFA) increases security by requiring users to be validated using more than one method. Once a user's identity has been verified, access control policies grant specified permissions, allowing the user to proceed further. Organizations utilize several access control methods depending on their needs.

Types of Access Control:

1. Discretionary Access Control (DAC):

In Discretionary access control, the file owner and resources set the access right permission. The data owner can grant permission to other users to access or perform any action on resources. For example, any operating system owner can set access permissions for other users, which allows flexibility and user preference.

Example: let's take an example where the owner of a personal computer sets access rights permission to other users to use his system documents.

Here,

Subject: user A, user B, user C

Object: Document 1, Document 2, Document 3

Access rights: Read, Write, Print

Access Matrix:

Object\Domain	Document 1	Document 2	Document 3
User A	Read	Print	Write
User B	Print	–	Read
User C	Write	Print	–

Explanation:

In the above observation, the owner of an object (documents) sets access right permission for a subject (User) to perform some operations on objects at their discretion.

- User A has read access to Document 1, print access to Document 2, and write access to Document 3

- User B has print access to document 1 and read access to document 3

- User C has Write access to document 1 and print access to document 2

2. Mandatory Access Control (MAC):

In Mandatory Access Control, the security policies or the system administrator decides the access permission based on the object's sensitivity. Access control assigns access according to the security levels of the subject and object; no user or owner can change access. The military or government uses these access rights for data protection and security.

Example: let's take an example where the owner of a personal computer sets access rights permission to other users to use his system documents

Here,

Subject: user A, user B, user C

Object: Document 1, Document 2, Document 3

Access rights: Read, Write, Print

Access Matrix:

Object\Domain	Document 1	Document 2	Document 3
User A	Read	Print	Write
User B	Print	–	Read
User C	Write	Print	–

Explanation:

- User A has read access to Document 1, print access to Document 2, and write access to Document 3

- User B has print access to document 1 and read access to document 3

- User C has write access to document 1 and print access to document 2

3. Role-Based Access Control (RBAC):

Roles based on their responsibilities grant access permission to users. Role-based access control is common in enterprise systems, some organizations, and healthcare systems with diverse roles.

Example: Let's take an example of health care with different users accessing system resources according to their roles

Here,

Subject: Doctor, Nurse, Receptionist

Object: Patient Information, Medicine records, Appointment

Access rights: Read, Edit

Access Matrix:

Object\Domain	Patient Information	Medicine Record	Appointment
Doctor	Read write	Read Write	Read Write
Nurse	Read Write	Read	Read
Receptionist	Read Write	–	Read Write

Explanation:

From the above matrix, we can observe that different roles are granted additional access rights to Hospital resources as

- Doctor has read and Write access to patient information, medicine records, and appointments.

- Nurse has Read and write access to patient information and only read access to medicine records and appointments.

- Receptionists have read and write access to patient information and appointments.

4. Attribute-Based Access Control (ABAC):

Access permission is granted based on different attributes and policies of subject, object, and environment. Here, attributes can be a role, resources, time, system type, and location. Modern environments commonly use this matrix where access is given or denied based on different attributes like the time and location of the user.

Example: Let's take an example where company HR, Team lead, and Employee have different access controls according to time attributes to utilize company resources.

Here,

Subject: HR, Team Lead, and Employee

Object: Office Computer, Files

Access Matrix:

Subject	Attribute: Time	Object: Office Computer	Object: Files
HR	24/7	Access	Access
Team lead	9 am-5 pm	Access	Access
Employee	9 am-5 pm	Access	No access

Explanation:

In the above observation, objects with different roles have access to subjects according to time attribute:

- Company HR has access to Company Computers and files all the time

- Team lead has access to a computer and files in the office working hour

- Employees have access to Company Computers during working hours

5. History-Based Access Control (HBAC): Access is granted or declined by evaluating the history of activities of the inquiring party that includes behavior, the time between requests and content of requests.

6. Identity-Based Access Control (IBAC): By using this model network administrators can more effectively manage activity and access based on individual requirements.

7. Organization-Based Access control (OrBAC): This model allows the policy designer to define a security policy independently of the implementation.

8. Rule-Based Access Control (RAC): RAC method is largely context based. Example of this would be only allowing students to use the labs during a certain time of day.

Different access control models are used depending on the compliance requirements and the security levels of information technology that is to be protected. Basically, access control is of 2 types:

- **Physical Access Control:** Physical access control restricts entry to campuses, buildings, rooms and physical IT assets.
- **Logical Access Control:** Logical access control limits connections to computer networks, system files and data.

Challenges of Access Control:

- **Distributed IT Systems:** Current IT systems frequently combine internet and on-premise networks. These systems may be distributed geographically and comprise various devices, assets, and virtual machines. Access is allowed to all of these devices, and keeping track of them can be challenging.
- **Policy Management:** Policy makers within the organization create policies, and the IT department converts the planned policies into code for implementation.

Coordination between these two groups is essential for keeping the access control system up to date and functioning properly.

- **Monitoring and Reporting:** Organizations must constantly check access control systems to guarantee compliance with corporate policies and regulatory laws. Any violations or changes must be recognized and reported immediately.
- **Access Control Models:** Access control mechanisms provide varying levels of precision. Choosing the right access control strategy for your organization allows you to balance acceptable security with employee efficiency.

Revocation of Access Rights:

- The need to revoke access rights dynamically raises several questions:
  - **Immediate versus delayed** - If delayed, can we determine when the revocation will take place?
  - **Selective versus general** - Does revocation of an access right to an object affect *all* users who have that right, or only some users?
  - **Partial versus total** - Can a subset of rights for an object be revoked, or are all rights revoked at once?
  - **Temporary versus permanent** - If rights are revoked, is there a mechanism for processes to re-acquire some or all of the revoked rights?
- With an access list scheme revocation is easy, immediate, and can be selective, general, partial, total, temporary, or permanent, as desired.
- With capabilities lists the problem is more complicated, because access rights are distributed throughout the system. A few schemes that have been developed include:
  - **Reacquisition** - Capabilities are periodically revoked from each domain, which must then re-acquire them.
  - **Back-pointers** - A list of pointers is maintained from each object to each capability which is held for that object.
  - **Indirection** - Capabilities point to an entry in a global table rather than to the object. Access rights can be revoked by changing or invalidating the table entry, which may affect multiple processes, which must then re-acquire access rights to continue.
  - **Keys -** A unique bit pattern is associated with each capability when created, which can be neither inspected nor modified by the process.
    - A master key is associated with each object.

- When a capability is created, its key is set to the object's master key.
- As long as the capability's key matches the object's key, then the capabilities remain valid.
- The object master key can be changed with the set-key command, thereby invalidating all current capabilities.
- More flexibility can be added to this scheme by implementing a ***list*** of keys for each object, possibly in a global table.

File Security:

File security is all about safeguarding your business-critical information from prying eyes by implementing stringent access control measures and flawless permission hygiene. Apart from enabling and monitoring security access controls, decluttering data storage also plays an important role in securing files. Regularly optimize file storage by purging old, stale, and other junk files to focus on business-critical files. Tackle data security threats and storage inefficiencies with periodic reviews and enhancements to your file security strategy.

Why is file security important?

- **To protect sensitive data:** Personally identifiable information (PII), electronic personal health information (ePHI), confidential contracts, and other business-critical data must be stored safely. Careless transmission or use of such files could lead to data privacy violations, resulting in heavy fines for the organization.
- **To secure file sharing:** Files transferred through unsecured channels can be misused by insiders or hackers for malicious activities. Comprehensive data leak prevention software can help prevent unauthorized movement of business-critical data out of the organization.
- **To avoid data breaches:** In 2019, personal details of 10.6 million MGM resort guests were breached. The impact of such a breach can be fatal to any organization. It is not just the fines and legal consequences, but also the loss of trust that can destroy a business.

What Are the Types of File Security?

File access security is a crucial part of keeping your data safe and secure. There are several different types of file access security, each with its own unique advantages and disadvantages.

Access Control Lists (ACLs): One type of file access security is access control lists (ACLs). ACLs allow you to specify exactly who has access to a particular file and what actions they are allowed to perform on it. For example, you could use ACLs to allow only certain users to read a file, while preventing others from making any changes to it.

Encryption: Another type of file access security is encryption. Encryption involves converting data into a coded format that can only be accessed by those with the correct decryption key—protecting your sensitive data from unauthorized access.

Creating regular backups of your important files, you can protect yourself against data loss due to hardware failure or accidental deletion. It's important to use a combination of these file access security measures to ensure that your data is as safe and utilized appropriately.

File Security Best Practices:

- **Eliminate permission hygiene issues**: The principle of least privileges (POLP) ensures that only the bare minimum privileges required to complete a task is granted. It is advisable to define access control lists (ACL) for files and folders based on user roles and requirements. Resolve permission hygiene propagation issues like undue privileges, and open access to files with a security permission analyzer.
- **Secure file sharing channels**: All file transfers should be authorized and secure. Audit all the possible ways files can be transferred, and block private devices like personal USB drives. Use USB data theft protection software to stop unofficial data transfers.
- **Implement file server auditing**: Be wary of multiple failed accesses, bulk file renames, or modifications. Mass, unofficial file modifications such as delete events may indicate a ransomware attack. Be prepared by automating your incident response against file threats with robust file server auditing software.
- **Enforce authentication and authorization protocols**: Enforce multi-factor authentication (MFA) for all users in your organizations. MFA makes it difficult for hackers to penetrate the network. Authorize only valid and official data access requests. Grant open access to all employees and partners only when necessary.
- **Conduct file storage analysis**: Analyze and manage your file repositories periodically. Know where your critical files are stored in the organization. Continuous review of stale files and unused files helps eliminate permission misuse incidents. Revoke permissions on files owned by former employees. Compute the cost of storing stale files with the help of our infographic.

How is file security different from data security?

- Files are the most basic securable units of a repository. Often data is stored and shared as files and folders. Therefore, file security is a subset of data security that focuses on the secure use of files.
- Data security protects data in use, in transit, and at rest. Infrastructural and software controls are used to implement stringent data security strategies. File security, on the other hand, protects sensitive files like personal information of customers and other business files.

User Authentication:

The user authentication process is used just to identify who the owner is or who the identified person is. On a personal computer, generally, user authentication can be performed using a password.

When a computer user wants to log into a computer system, the operating system (OS) installed on that computer system generally wants to determine or check who the user is. This process is called "**user authentication**."

It is sometimes critical to authenticate the user because the computer system may contain sensitive information about the owner.

Most methods of authenticating computer users when they attempt or attempt to log into the system are based on one of the three principles listed below:

- Something the user is aware of
- Something that the user possesses
- Something is wrong, the user.

That computer users who want to cause trouble on a specific computer system must first log in, which means getting past whatever authentication method or procedure is in place. Those computer users are called hackers.

Basically, "hacker" is a term of honour that is reserved for or given to a great computer programmer, as a normal computer user or programmer can't get access into anyone's system without permission.

Common Methods for User Authentication:

1. Password-Based Authentication: The most widely used form of authentication where users enter a password associated with their account.

Implementation:

- Enforce strong password policies (length, complexity, expiration).
- Use password hashing and salting to securely store passwords.
- Implement account lockout after multiple failed login attempts to prevent brute force attacks.
- Encourage users to avoid common passwords and enable two-factor authentication (2FA).

2. Two-Factor Authentication (2FA): Requires users to provide two forms of identification: something they know (password) and something they have (token, smartphone).

Implementation:

- Uses methods such as one-time codes sent via SMS or generated by an authenticator app.
- Adds an extra layer of security, even if passwords are compromised.
- Examples: Google Authenticator, Authy, hardware tokens.

3. Biometric Authentication: Uses physical or behavioral characteristics of an individual for authentication.

Implementation:

- Fingerprints, facial recognition, iris scans, voice recognition.
- Provides strong authentication but may require specialized hardware.
- Often used in smartphones and high-security environments.

4. Multi-Factor Authentication (MFA): Combines two or more authentication factors (e.g., password, 2FA) for increased security.

Implementation:

- Provides enhanced protection against unauthorized access.
- Example: Requiring a password (something the user knows) and a fingerprint scan (something the user has).

5. Certificate-Based Authentication: Uses digital certificates to authenticate users.

Implementation:

- Each user has a unique digital certificate signed by a trusted Certificate Authority (CA).
- Requires PKI (Public Key Infrastructure) for issuing and managing certificates.
- Commonly used in secure web browsing (SSL/TLS) and VPN connections.

6. Single Sign-On (SSO): Allows users to authenticate once and access multiple services without re-entering credentials.

Implementation:

- Users log in once and gain access to all authorized systems.
- Reduces password fatigue and improves user experience.
- Requires a centralized identity provider (IdP) like Active Directory, LDAP, or OAuth.

7. OAuth and OpenID Connect: Protocols for authentication and authorization between applications.

Implementation:

- OAuth allows users to grant third-party applications access to their resources without sharing passwords.
- OpenID Connect is an authentication layer on top of OAuth, providing user authentication and identity services.

8. Risk-Based Authentication: Analyzes various factors to assess the risk of a login attempt.

Implementation:

- Considers location, device, IP address, time of access, and user behavior.
- Adapts authentication requirements based on risk level.
- Helps detect and prevent unauthorized access attempts.

9. Adaptive Authentication: Similar to risk-based authentication but dynamically adjusts authentication requirements based on user behavior.

Implementation:

- Evaluates user behavior during a session (e.g., typing speed, mouse movements).
- May prompt for additional authentication if behavior is unusual.
- Enhances security without inconveniencing users with unnecessary authentication steps.

10. Passwordless Authentication: Removes the need for passwords altogether, relying on other authentication methods.

Implementation:

- Uses methods like biometrics, security keys, or mobile push notifications.
- Eliminates the risk of password-related attacks.
- Increasingly popular for improved security and user experience.

Best Practices for User Authentication:

1. Strong Password Policies:

- Require complex passwords with a mix of letters, numbers, and symbols.
- Enforce regular password changes.
- Educate users on creating secure passwords and discourage password reuse.

2. Multi-Factor Authentication (MFA):

- Implement 2FA or MFA wherever possible, especially for critical systems.
- Use a combination of factors for stronger authentication.

3. Centralized Identity Management:

- Use centralized authentication systems like Active Directory or LDAP.
- Simplifies user management and reduces the risk of credential sprawl.

4. Regular Auditing and Monitoring:

- Monitor authentication logs for unusual activity.
- Conduct periodic security audits and reviews.

5. Encryption of Authentication Data:

- Ensure that passwords and authentication data are encrypted in transit and at rest.
- Use secure protocols like HTTPS.

6. User Training:

- Educate users on common security threats (phishing, social engineering).
- Provide training on how to recognize and report suspicious activity.

7. Update and Patching:

- Keep authentication systems and software up to date with security patches.
- Patch known vulnerabilities promptly to prevent exploitation.

8. Secure Recovery Options:

- Implement secure password recovery or account recovery processes.
- Verify the identity of users before allowing password resets or account recovery.

PART – 2 : CASE STUDY

Linux Operating System:

The Linux Operating System is a type of operating system that is similar to Unix, and it is built upon the Linux Kernel. The Linux Kernel is like the brain of the operating system because it manages how the computer interacts with its hardware and resources. It makes sure everything works smoothly and efficiently. But the Linux Kernel alone is not enough to make a complete operating system. To create a full and functional system, the Linux Kernel is combined with a collection of software packages and utilities, which are together called Linux distributions. These distributions make the Linux Operating System ready for users to run their applications and perform tasks on their computers securely and effectively. Linux distributions come in different flavors, each tailored to suit the specific needs and preferences of users.

Introduction to Linux:

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by ***Linus Torvalds***. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term ***"Linux"*** in the title, but the Free Software Foundation uses the ***"GNU/Linux"*** title to focus on the necessity of GNU software, causing a few controversies.

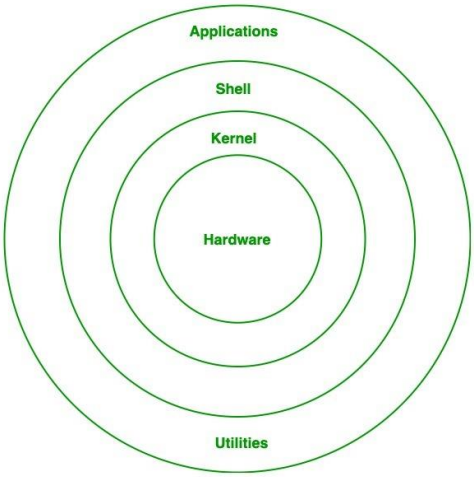
Famous Linux distributions are Ubuntu, Fedora Linux, and Debian, the latter of which is composed of several different modifications and distributions, including Xubuntu and Lubuntu. Commercial distributions are SUSE Linux Enterprise and Red Hat Enterprise Linux. Desktop distributions of Linux are windowing systems like Wayland or X11 and desktop environments like KDE Plasma and GNOME.

- Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems.
- Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022.
- However, Linux is used by just around 2.6% of desktop computers as of November 2022.
- Also, Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system.

- It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers.

Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License). For example, the Linux kernel is licensed upon the GPLv2.

Architecture of Linux:



1. **Kernel:** Kernel is the core of the Linux based operating system. It virtualizes the common hardware resources of the computer to provide each process with its virtual resources. This makes the process seem as if it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes. Different types of the kernel are:
    - Monolithic Kernel
    - Hybrid kernels
    - Exo kernels
    - Micro kernels
2. **System Library:**Linux uses system libraries, also known as shared libraries, to implement various functionalities of the operating system. These libraries contain pre-written code that applications can use to perform specific tasks. By using these libraries, developers can save time and effort, as they don't need to write the same code repeatedly. System libraries act as an interface between applications and the kernel, providing a standardized and efficient way for applications to interact with the underlying system.

3. **Shell:**The shell is the user interface of the Linux Operating System. It allows users to interact with the system by entering commands, which the shell interprets and executes. The shell serves as a bridge between the user and the kernel, forwarding the user's requests to the kernel for processing. It provides a convenient way for users to perform various tasks, such as running programs, managing files, and configuring the system.
4. **Hardware Layer:** The hardware layer encompasses all the physical components of the computer, such as RAM (Random Access Memory), HDD (Hard Disk Drive), CPU (Central Processing Unit), and input/output devices. This layer is responsible for interacting with the Linux Operating System and providing the necessary resources for the system and applications to function properly. The Linux kernel and system libraries enable communication and control over these hardware components, ensuring that they work harmoniously together.
5. **System Utility:** System utilities are essential tools and programs provided by the Linux Operating System to manage and configure various aspects of the system. These utilities perform tasks such as installing software, configuring network settings, monitoring system performance, managing users and permissions, and much more. System utilities simplify system administration tasks, making it easier for users to maintain their Linux systems efficiently.

Basic Features of Linux Operating System:

- **Portable** − Portability means software can works on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** − Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** − Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** − Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Kernel Mode vs User Mode:

Kernel component code executes in a special privileged mode called **kernel mode** with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in **User Mode** which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

History of Linux OS:

1. **Precursor Developments (1960s - 1980s):**
   - In the late 1960s and early 1970s, Unix was developed at AT&T Bell Labs, laying the groundwork for future Unix-like operating systems.
   - In the 1980s, Richard Stallman launched the GNU Project, aiming to create a free Unix-like operating system.
   - Stallman developed essential software components like the GNU C Compiler (GCC) and the GNU General Public License (GPL), which enabled free software distribution and modification.
2. **Origins (1991 - 1994):**
   - In 1991, Linus Torvalds began developing the Linux kernel on a personal computer based on the MINIX operating system, which was a Unix-like system designed for educational purposes.
   - Torvalds initially intended Linux to be a free alternative to MINIX, but it quickly grew into a full-fledged operating system.
   - On October 5, 1991, Torvalds announced the first official version of the Linux kernel, version 0.02.
3. **Early Development and Growth (1994 - 1998):**
   - In 1994, the first major Linux distributions, such as Slackware, Debian, and Red Hat Linux, were released, making it easier for users to install and use Linux.
   - The Linux kernel and associated utilities were licensed under the GNU General Public License (GPL), ensuring its free and open-source nature.
   - Linux gained popularity in the academic and research communities, as well as among hobbyists and enthusiasts.

4. **Commercial Adoption and Enterprise Support (1998 - 2005):**
    - Linux began to gain traction in the commercial sector, with companies like Red Hat, SUSE, and Canonical providing enterprise-level support and services.
    - Major hardware and software vendors started supporting Linux, recognizing its potential in the server and data center markets.
    - The Linux kernel development process became more structured, with the introduction of the stable and development kernel branches.

5. **Desktop and Mobile Expansion (2005 - 2015):**
    - Linux distributions like Ubuntu and Fedora focused on improving the desktop user experience, making Linux more accessible to mainstream users.
    - Linux found its way into mobile devices and embedded systems, powering smartphones (e.g., Android), tablets, and Internet of Things (IoT) devices.
    - Cloud computing platforms, such as Amazon Web Services (AWS) and Google Cloud Platform (GCP), adopted Linux as the primary operating system for their services.

6. **Continued Growth and Adoption (2015 - present):**
    - Linux has become the dominant operating system for servers, cloud computing, and supercomputers, powering many of the world's most powerful systems.
    - The Linux kernel has continued to evolve, incorporating new features, improving performance, and supporting emerging technologies like containers (e.g., Docker) and Kubernetes.
    - Linux has also gained popularity in various industries, including finance, healthcare, media, and scientific research, due to its reliability, security, and customizability.

Throughout its history, Linux has been driven by a vibrant community of developers, contributors, and users worldwide. Its open-source nature has allowed for continuous innovation, collaboration, and adaptation to changing technological landscapes, solidifying its position as a leading operating system in various domains.

Design Principles:

Here are some of the key design principles that have guided the development of Linux:

1. **Modularity:** Linux follows a modular design approach, where the kernel and other components are divided into separate modules or layers. This modularity allows for easier development, maintenance, and customization, as individual modules can be updated or replaced without affecting the entire system.
2. **Hierarchical File System:** Linux inherits the hierarchical file system structure from Unix, where all files and directories are organized in a tree-like structure, starting from the root directory (/). This design principle promotes organization and ease of navigation within the file system.
3. **Everything is a File:** In Linux, almost everything is treated as a file, including devices, pipes, and sockets. This design principle simplifies the way resources are managed and accessed within the system, providing a consistent interface for interacting with different components.
4. **Multiuser and Multitasking:** Linux is designed to support multiple users and multitasking, allowing multiple users to work on the system simultaneously and run multiple programs concurrently. This principle is essential for efficient resource sharing and collaboration.
5. **Open Source:** Linux is an open-source operating system, which means that its source code is freely available for anyone to study, modify, and distribute. This design principle promotes transparency, collaboration, and continuous improvement by leveraging the collective efforts of the global developer community.
6. **Portability:** Linux is designed to be portable across different hardware architectures and platforms. This design principle ensures that Linux can run on a wide range of devices, from embedded systems to supercomputers, without significant modifications.
7. **Security:** Security is a fundamental design principle in Linux, with features like access control lists (ACLs), user and group permissions, and the principle of least privilege. These security mechanisms help protect the system and data from unauthorized access and potential vulnerabilities.
8. **Command-Line Interface (CLI):** While Linux distributions often provide graphical user interfaces (GUIs), the command-line interface (CLI) remains an essential design principle. The CLI provides a powerful and efficient way to interact with the system, automate tasks, and perform advanced system administration tasks.
9. **Network Transparency:** Linux is designed with network transparency in mind, allowing seamless communication and integration with other systems and networks. This design principle facilitates distributed computing, resource sharing, and remote access capabilities.

Kernel Modules:

Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. A module can be configured as built-in or loadable. To dynamically load or remove a module, it has to be configured as a loadable module in the kernel configuration.

There are several advantages that come with using kernel modules:

- The kernel does not have to rebuild your kernel as often. This saves time and prevents the possibility of introducing an error in rebuilding and reinstalling the base kernel. Once you have a working base kernel, it is good to leave it untouched as long as possible.
- It is easier to diagnose system problems. A bug in a device driver which is bound into the kernel can stop the system from booting at all. It can also be really hard to tell which part of the base kernel caused the trouble. If the same device driver is a module, though, the base kernel is up and running before the device driver even gets loaded. If the system dies after the base kernel is up and running, it's an easy matter to track the problem down to the trouble-making device driver and just not load that device driver until the problem is fixed.
- Using modules can save memory, because they are loaded only when the system is actually using them. All parts of the base kernel stay loaded, in real storage, not just virtual storage.
- Modules are much faster to maintain and debug. What would require a full reboot to do with a filesystem driver built into the kernel can be done with a few quick commands using modules. It is possible to try out different parameters or even change the code repeatedly in rapid succession, without waiting for a boot.
- Modules are not slower, by the way, than base kernel modules. Calling either one is simply a branch to the memory location where it resides.

Module Location:

The code necessary to create a new kernel with new module included, or old modules removed is usually: **/lib/modules/$(uname -r)/kernel** although on some distributions the code is found in **/usr/lib/modules/$(uname -r)/kernel**. The /usr/lib and /lib directory is where Linux stores object libraries and shared libraries that are necessary to run certain commands, including kernel code.

These directories are only necessary if you wish to rebuild your kernel with new modules included, or old modules removed. It would be best to seek out assistance from a source knowledgeable with the specific distribution you are working with.

Kernel Modules Subdirectories:

Directory	Description
arch	The arch subdirectory contains all of the architecture specific kernel code. It has further subdirectories, one per supported architecture, for example i386 and alpha.
include **Although man distributions place the include files in the /usr/src/$(uname -r)**	The include subdirectory contains most of the include files needed to build the kernel code. It too has further subdirectories including one for every architecture supported. The include/asm subdirectory is a soft link to the real include directory needed for this architecture, for example include/asm-i386. To change architectures, you need to edit the kernel makefile and rerun the Linux kernel configuration program.
init	This directory contains the initialization code for the kernel and it is a very good place to start looking at how the kernel works.
mm	This directory contains all of the memory management code. The architecture specific memory management code lives down in arch/*/mm/, for example arch/i386/mm/fault.c.
drivers	All of the system's device drivers live in this directory. They are further sub-divided into classes of device driver, for example block.
ipc	This directory contains the kernels inter-process communications code.
modules	This is simply a directory used to hold built modules.
fs	All of the file system code. This is further sub-divided into directories, one per supported file system, for example vfat and ext2.
kernel	The main kernel code. Again, the architecture specific kernel code is in arch/*/kernel.
net	The kernel's networking code.
lib	This directory contains the kernel's library code. The architecture specific library code can be found in arch/*/lib/.
scripts	This directory contains the scripts (for example awk and tk scripts) that are used when the kernel is configured.

Case Study: Process Management in an Operating System

Background:

ABC Corporation, a large technology firm, has recently upgraded its operating system to a more advanced version. With this upgrade, they are keen on optimizing their process management to improve efficiency and resource utilization.

Challenge:

The previous OS lacked advanced process management capabilities, leading to inefficiencies such as long wait times for users, uneven resource distribution, and occasional system crashes due to poorly managed processes. ABC Corporation needs to implement a robust process management system to address these issues.

Implementation Steps:

1. Process Creation: The new OS provides advanced process creation capabilities. When a user initiates a program or task, the OS creates a corresponding process. For example, when a user launches a web browser, a process is created for that browser session.

2. Process Scheduling: To optimize CPU utilization and responsiveness, the OS employs a scheduling algorithm. The "Round Robin" scheduling algorithm is chosen for its fairness and simplicity. Each process is assigned a fixed time slice, typically 10 milliseconds, before being preempted and moved to the back of the queue.

3. Process States: The OS manages processes in various states: Running, Ready, Blocked (or Waiting), and Terminated.

- **Running**: The process is currently executing instructions on the CPU.
- **Ready**: The process is ready to execute but is waiting for its turn in the CPU scheduling queue.
- **Blocked (Waiting)**: The process is waiting for an event (such as user input or a resource) to occur.
- **Terminated**: The process has finished execution.

4. Process Suspension and Resumption: The OS supports suspending and resuming processes. When a process is suspended, its state and context are saved to secondary storage, allowing other processes to utilize CPU resources. When the process is resumed, it is loaded back into memory with its previous state intact.

5. Process Synchronization: To prevent conflicts and ensure data integrity, the OS provides mechanisms for process synchronization. ABC Corporation implements semaphore-based synchronization. For instance, when multiple processes need access to a shared resource, they use semaphores to coordinate access and avoid data corruption.

6. Process Communication: Inter-process communication (IPC) is crucial for collaboration among processes. ABC Corporation uses message passing for IPC. Processes can send and receive messages through defined communication channels, facilitating data exchange and cooperation.

Results:

Improved Performance: The Round Robin scheduling algorithm has significantly improved CPU utilization. Processes are allocated fair CPU time, reducing wait times and enhancing system responsiveness.

Enhanced Stability: With proper process states and management, system crashes due to process errors have decreased. The OS effectively manages process termination and resource allocation.

Efficient Resource Utilization: Process suspension and resumption mechanisms optimize resource usage. Background processes can be suspended to prioritize critical tasks, improving overall system efficiency.

Secure Interactions: Semaphore-based synchronization ensures secure access to shared resources, preventing data corruption and conflicts between processes.

Streamlined Collaboration: Message passing for IPC enables seamless collaboration between processes. Applications can communicate efficiently, leading to smoother workflow and enhanced productivity.

Conclusion:

Through the implementation of advanced process management features, ABC Corporation has successfully optimized its operating system. The new system offers improved performance, stability, resource utilization, and secure inter-process communication. These enhancements have not only addressed previous inefficiencies but have also laid a foundation for future scalability and innovation within the organization.

This case study showcases how a modern operating system can leverage sophisticated process management techniques to enhance efficiency and user experience.

Case Study: Scheduling in an Operating System

Background:

XYZ Corporation operates a large data center hosting various web services and applications. They recently upgraded their operating system to improve resource utilization and responsiveness. With this upgrade, they focused on implementing an advanced scheduling algorithm to manage the numerous processes running on their servers.

Challenge:

The previous OS scheduling algorithm was causing uneven resource distribution, leading to some services experiencing sluggish performance while others remained underutilized. XYZ Corporation aimed to implement a more efficient scheduling algorithm to address these issues and improve overall system performance.

Implementation Steps:

1. Evaluation of Scheduling Algorithms: The IT team at XYZ Corporation conducted a thorough evaluation of various scheduling algorithms, considering factors such as CPU utilization, turnaround time, waiting time, and response time. After careful analysis, they decided to implement the **Weighted Round Robin (WRR)** scheduling algorithm.

2. Weighted Round Robin (WRR) Algorithm: The WRR algorithm assigns a weight to each process based on its priority and resource requirements. Higher-priority processes or those requiring more resources are assigned a higher weight. The algorithm then allocates CPU time slices to processes in a round-robin fashion, but with each process receiving time proportional to its weight.

3. Process Priority: XYZ Corporation categorized their services and applications into priority levels:

- High-Priority: Critical services like payment processing and customer support.
- Medium-Priority: Web applications and APIs.
- Low-Priority: Background tasks such as data backups and system maintenance.

4. Setting Weights: The IT team assigned weights to processes based on their priority levels and resource requirements. For example:

- High-Priority: Weight of 4
- Medium-Priority: Weight of 2
- Low-Priority: Weight of 1

5. Dynamic Adjustments: To adapt to changing workload demands, the WRR algorithm supports dynamic adjustments. If a process's priority changes or if new processes are added, the weights are adjusted accordingly.

6. Monitoring and Feedback: XYZ Corporation implemented monitoring tools to track CPU utilization, response times, and overall system performance. This data is fed back into the scheduling algorithm to continuously fine-tune weights and priorities.

Results:

Improved Resource Utilization: The WRR algorithm ensures that high-priority processes receive more CPU time, leading to improved responsiveness for critical services.

Balanced Workloads: Medium and low-priority processes are not starved of resources. The algorithm's weighted approach ensures a fair distribution of CPU time, preventing underutilization.

Enhanced System Stability: With more balanced workloads, the system experiences fewer instances of overload and better stability.

Dynamic Adaptability: The ability to dynamically adjust weights allows XYZ Corporation to quickly respond to changes in workload patterns. New services or changes in priority are seamlessly accommodated.

Performance Monitoring: Real-time monitoring tools provide valuable insights into system performance. The IT team can identify bottlenecks, optimize resource allocation, and make data-driven decisions.

Conclusion:

By implementing the Weighted Round Robin (WRR) scheduling algorithm, XYZ Corporation has significantly improved its operating system's performance and resource utilization. Critical services receive the necessary CPU time, while other processes are allocated resources fairly. The system's stability has increased, and the IT team can easily adapt to changing workload demands. Real-time monitoring ensures that the system operates efficiently, with data guiding continuous improvements.

This case study demonstrates how a carefully chosen scheduling algorithm can have a profound impact on system performance, responsiveness, and stability. XYZ Corporation's implementation of the WRR algorithm serves as a model for other organizations aiming to optimize their operating system scheduling.

Case Study: Memory Management in an Operating System

Background:

ABC Bank, a large financial institution, operates a variety of critical applications ranging from online banking to customer relationship management systems. With an increasing demand for digital services, they encountered memory-related issues in their legacy operating system, leading to frequent crashes and performance degradation. ABC Bank decided to upgrade their OS to improve memory management and system stability.

Challenge:

The previous OS lacked efficient memory management, resulting in memory leaks, fragmentation, and inadequate utilization of RAM. ABC Bank aimed to implement advanced memory management techniques to address these challenges and ensure optimal performance of their banking applications.

Implementation Steps:

1. Virtual Memory Implementation: The upgraded OS introduced a sophisticated virtual memory system, allowing ABC Bank's applications to access more memory than physically available. This feature helps prevent memory exhaustion and enables efficient use of secondary storage (such as hard drives) as an extension of RAM.

2. Page Replacement Algorithm: To manage virtual memory efficiently, the OS implemented the **Least Recently Used (LRU)** page replacement algorithm. LRU ensures that the least recently used page is replaced when the system needs to free up memory space. This minimizes the number of page faults and optimizes memory utilization.

3. Memory Segmentation: ABC Bank's applications have varying memory requirements. The OS utilizes memory segmentation to divide memory into segments based on the type of data or application. This segmentation helps in organizing and protecting memory, preventing one application from accessing another's memory space.

4. Memory Protection: To enhance security and prevent unauthorized access, the OS implements memory protection mechanisms. Each segment is assigned access rights (read-only, read-write, execute-only, etc.), ensuring that applications operate within their designated memory boundaries.

5. Dynamic Memory Allocation: ABC Bank's applications often require dynamic memory allocation for tasks such as processing customer transactions or generating reports. The OS supports dynamic memory allocation through techniques like **First Fit** and **Best Fit** algorithms. This allows applications to request and release memory as needed, reducing memory wastage.

6. Memory Leak Detection: To detect and mitigate memory leaks, the OS includes built-in tools for memory leak detection. These tools monitor application memory usage and identify instances where allocated memory is not released after use. ABC Bank's developers can then rectify these issues to prevent memory bloat and system instability.

Results:

Improved System Stability: The implementation of virtual memory and the LRU page replacement algorithm has significantly improved system stability. Memory leaks and fragmentation are minimized, leading to fewer crashes and system slowdowns.

Enhanced Performance: With efficient memory management, ABC Bank's applications experience improved performance. The LRU algorithm ensures that frequently accessed pages remain in memory, reducing the number of costly disk accesses.

Optimal Resource Utilization: Memory segmentation allows for optimal resource utilization. Applications are allocated memory segments based on their requirements, preventing one application from monopolizing system resources.

Secure Memory Access: Memory protection mechanisms ensure secure memory access. Applications operate within their designated memory boundaries, preventing unauthorized access or corruption.

Scalability: The OS's dynamic memory allocation capabilities support the bank's growth. As new applications are developed, they can efficiently request and release memory, ensuring scalability without compromising system performance.

Conclusion:

By upgrading to an OS with advanced memory management capabilities, ABC Bank has overcome the memory-related challenges that plagued their legacy system. The implementation of virtual memory, the LRU page replacement algorithm, memory segmentation, and dynamic memory allocation has led to improved system stability, enhanced performance, and optimal resource utilization. With secure memory access and built-in memory leak detection, the bank's critical applications run smoothly, providing a seamless banking experience for customers.

This case study highlights the importance of efficient memory management in ensuring the stability, performance, and security of an operating system, particularly in a critical environment such as a financial institution. ABC Bank's success in implementing advanced memory management techniques serves as a model for organizations looking to optimize their systems for reliability and efficiency.

Case Study: File Systems in an Operating System

Background:

XYZ Manufacturing is a global company with multiple offices and manufacturing plants worldwide. They deal with large volumes of data ranging from product designs to inventory management. Facing challenges with data organization, retrieval speed, and data loss prevention, XYZ Manufacturing decided to revamp their file system as part of their IT infrastructure upgrade.

Challenge:

The existing file system lacked features for efficient data storage, retrieval, and backup. XYZ Manufacturing aimed to implement an advanced file system to overcome these challenges, improve data management, and ensure data integrity across their global operations.

Implementation Steps:

1. Choice of File System: After evaluating various file systems, XYZ Manufacturing opted for the **ZFS (Zettabyte File System)** due to its advanced features and data protection mechanisms.

2. Data Deduplication: ZFS offers native data deduplication, which is crucial for XYZ Manufacturing's storage efficiency. This feature identifies duplicate blocks of data across the file system and only stores unique data, saving significant storage space.

3. Copy-on-Write (CoW) Mechanism: ZFS employs a Copy-on-Write mechanism, where new data is written to a new location rather than overwriting existing data. This ensures data integrity and eliminates the risk of corruption during power failures or crashes.

4. Snapshots: XYZ Manufacturing utilizes ZFS snapshots for point-in-time copies of their data. Snapshots allow for quick and efficient backups without consuming additional storage space. They can also be used for data recovery in case of accidental deletion or corruption.

5. RAID-Z for Redundancy: To ensure data redundancy and fault tolerance, XYZ Manufacturing implements **RAID-Z**, which is similar to RAID but designed specifically for ZFS. RAID-Z provides various levels of data redundancy, allowing XYZ Manufacturing to choose the level that best suits their needs.

6. Scalability and Expansion: With ZFS, XYZ Manufacturing has a scalable file system that can easily accommodate their growing data needs. Adding additional storage pools or expanding existing ones is seamless, ensuring they can scale their infrastructure without downtime.

7. Data Integrity Checks: ZFS includes built-in mechanisms for data integrity checks, such as checksums. This feature verifies the integrity of data on disk and detects any errors, ensuring that data remains consistent and reliable.

Results:

Improved Data Storage Efficiency: ZFS's data deduplication feature has significantly improved storage efficiency for XYZ Manufacturing. Duplicate data is eliminated, saving storage space and reducing costs.

Enhanced Data Protection: The CoW mechanism and RAID-Z provide robust data protection. Data integrity is maintained, and the risk of data loss due to corruption or hardware failures is minimized.

Efficient Backup and Recovery: ZFS snapshots allow XYZ Manufacturing to perform quick backups and easily recover data from specific points in time. This enhances data protection and reduces downtime in case of data loss incidents.

Seamless Scalability: With ZFS, XYZ Manufacturing can scale their storage infrastructure seamlessly. Adding more storage or expanding existing pools does not disrupt operations, ensuring continuity of their global operations.

Reliable Data Integrity: ZFS's built-in data integrity checks ensure that data remains consistent and reliable. Any errors or corruption are detected and corrected, maintaining the integrity of critical data.

Conclusion:

By implementing the ZFS file system, XYZ Manufacturing has transformed their data management practices. The advanced features such as data deduplication, CoW mechanism, snapshots, RAID-Z, and scalability have brought significant improvements in storage efficiency, data protection, backup, and recovery. The file system's reliability and data integrity checks provide peace of mind, ensuring that critical data is safe and accessible across their global operations.

This case study demonstrates how the choice of a robust file system like ZFS can have a profound impact on an organization's data management practices. XYZ Manufacturing's success in implementing ZFS serves as an example for other enterprises looking to enhance their data storage, protection, and scalability capabilities.

Case Study: Input and Output (I/O) in an Operating System

Background:

XYZ Logistics is a global transportation company with a fleet of trucks, ships, and airplanes. They rely heavily on real-time data for tracking shipments, managing inventory, and coordinating logistics operations. Facing challenges with slow I/O performance and data bottlenecks, XYZ Logistics decided to upgrade their I/O system as part of their digital transformation initiative.

Challenge:

The existing I/O system was unable to handle the volume of real-time data generated by XYZ Logistics' operations. This resulted in delays, data loss, and inefficiencies in tracking shipments and managing inventory. XYZ Logistics aimed to implement an advanced I/O system to improve data throughput, reliability, and responsiveness.

Implementation Steps:

1. High-Speed I/O Interfaces: XYZ Logistics upgraded their hardware to include high-speed I/O interfaces such as **NVMe (Non-Volatile Memory Express)** for storage and **10 Gigabit Ethernet** for networking. These interfaces provide significantly higher data transfer rates compared to traditional interfaces.

2. I/O Scheduler Optimization: The operating system's I/O scheduler was optimized for the company's workload. They chose the **Deadline I/O Scheduler** for its ability to prioritize I/O requests based on deadlines, ensuring that time-sensitive operations such as real-time tracking receive priority.

3. Buffering and Caching: XYZ Logistics implemented intelligent buffering and caching mechanisms to improve I/O performance. This included utilizing the operating system's buffer cache to store frequently accessed data in memory, reducing the need for disk reads.

4. Asynchronous I/O: To enhance concurrency and responsiveness, XYZ Logistics adopted asynchronous I/O operations. This allows I/O requests to be submitted without waiting for the operation to complete, enabling the system to continue processing other tasks while waiting for I/O operations to finish.

5. RAID for Data Redundancy: For data reliability, XYZ Logistics implemented **RAID (Redundant Array of Independent Disks)** configurations. They chose RAID 5 for their storage arrays, providing both data redundancy and improved I/O performance by striping data across multiple disks.

6. Disk Partitioning and Mounting: XYZ Logistics partitioned their disks intelligently based on data access patterns. Frequently accessed data was placed on faster storage devices, while archival data was stored on slower but higher-capacity devices. They also optimized disk mounting options for improved read/write speeds.

7. I/O Monitoring and Optimization: The IT team implemented robust I/O monitoring tools to track performance metrics such as throughput, latency, and IOPS (Input/Output Operations Per Second). This data was used to continuously optimize the I/O system, identifying bottlenecks and fine-tuning configurations.

Results:

Enhanced Data Throughput: The adoption of high-speed I/O interfaces such as NVMe and 10 Gigabit Ethernet significantly improved data transfer rates. XYZ Logistics experienced faster data processing and reduced latency in tracking shipments and managing inventory.

Improved System Responsiveness: Asynchronous I/O operations allowed the system to handle multiple I/O requests concurrently, enhancing system responsiveness. Real-time tracking updates and inventory management operations became more efficient.

Data Redundancy and Reliability: RAID 5 configurations provided data redundancy, ensuring that critical data is protected against disk failures. This enhanced data reliability and minimized the risk of data loss.

Intelligent Data Placement: By intelligently partitioning disks and utilizing caching mechanisms, XYZ Logistics optimized data access. Frequently accessed data was readily available in memory, reducing the need for disk reads and improving overall performance.

Continuous Optimization: The implementation of I/O monitoring tools allowed the IT team to proactively identify and address performance bottlenecks. Continuous optimization based on real-time data ensured that the I/O system remained efficient and responsive.

Conclusion:

By upgrading their I/O system with high-speed interfaces, optimized schedulers, intelligent caching, RAID configurations, and continuous monitoring, XYZ Logistics has overcome the challenges of slow I/O performance. The enhanced data throughput, system responsiveness, and data reliability have significantly improved their logistics operations. Real-time tracking of shipments and efficient inventory management are now possible, leading to increased efficiency and customer satisfaction.

This case study highlights the critical role of an efficient I/O system in a data-intensive environment such as logistics. XYZ Logistics' success in implementing advanced I/O techniques serves as a model for other organizations seeking to improve data throughput, reliability, and responsiveness in their operations.

Case Study: Inter-Process Communication (IPC) in an Operating System

Background:

ABC Tech is a software development company working on a complex project that requires collaboration among multiple processes. Their project involves building a real-time chat application where users can communicate with each other. To facilitate seamless communication between various components of the chat application, ABC Tech decided to implement robust inter-process communication mechanisms.

Challenge:

The existing communication methods, such as using files for data exchange or shared memory, were proving to be inefficient and prone to data inconsistencies. ABC Tech needed a reliable and efficient IPC mechanism to ensure real-time messaging, data consistency, and synchronization among the different processes of the chat application.

Implementation Steps:

1. Message Queues: ABC Tech implemented **Message Queues** as the primary IPC mechanism for the chat application. Each user's chat session is managed by a separate process, and message queues facilitate communication between these processes.

2. Message Format: Messages in the queues are formatted to include sender information, recipient information, timestamp, and the actual message content. This structured format ensures that messages are processed correctly by the receiving processes.

3. Process Synchronization: To prevent race conditions and ensure data integrity, ABC Tech implemented **semaphores** for process synchronization. Semaphores are used to control access to shared resources such as the message queues. For example, a semaphore ensures that only one process can write to a message queue at a time.

4. Error Handling: The IPC implementation includes robust error handling mechanisms. If a process fails to send a message or encounters an error, it notifies the appropriate process and retries the operation. This ensures that messages are not lost due to communication failures.

5. Broadcast Messages: ABC Tech introduced the concept of **broadcast messages** for system-wide notifications. When a user sends a broadcast message, it is distributed to all active chat sessions, allowing users to send announcements or alerts to everyone connected to the chat application.

6. Event Handling: The IPC system includes event-driven mechanisms for handling user actions such as sending a message, receiving a message, joining a chat room, or leaving a chat room. These events trigger the appropriate actions in the corresponding processes, ensuring seamless user experiences.

7. Testing and Optimization: Before deploying the IPC system, ABC Tech conducted extensive testing to ensure reliability and efficiency. They monitored IPC performance metrics such as message delivery times, system resource usage, and throughput. Based on the test results, they optimized the IPC implementation for better performance.

Results:

Real-Time Communication: The implementation of message queues enabled real-time communication between users in the chat application. Messages are delivered promptly, providing a seamless chatting experience.

Data Consistency: Structured message formats and process synchronization mechanisms ensured data consistency across all chat sessions. Users receive messages in the correct order, and there are no data conflicts.

Efficient Resource Usage: The IPC system optimized resource usage by controlling access to shared resources using semaphores. This prevented resource contention and improved system efficiency.

System-Wide Notifications: The introduction of broadcast messages allowed users to send announcements or alerts to all connected users simultaneously. This feature enhanced the communication capabilities of the chat application.

Robust Error Handling: The IPC system's error handling mechanisms ensured that messages were not lost due to communication failures. Processes were able to recover from errors and retry operations when necessary.

Seamless User Experience: Event-driven mechanisms for handling user actions resulted in a seamless user experience. Users could send and receive messages, join chat rooms, and receive notifications without delays or interruptions.

Conclusion:

By implementing a robust IPC system using message queues, semaphores for synchronization, broadcast messages, and event-driven mechanisms, ABC Tech successfully addressed the challenges of inter-process communication in their real-time chat application. The system now facilitates seamless communication among users, ensures data consistency, and provides efficient resource usage.

This case study demonstrates the importance of effective inter-process communication mechanisms in complex applications. ABC Tech's success in implementing IPC mechanisms serves as a model for other software development companies seeking to enhance communication and collaboration among processes in their applications.

Case Study: Real-Time Operating System (RTOS) in Automotive Systems

Background:

XYZ Automotive is a leading car manufacturer known for its advanced vehicle systems and safety features. To enhance their vehicles' performance, safety, and user experience, XYZ Automotive decided to incorporate a Real-Time Operating System (RTOS) into their automotive systems. The RTOS would be used to control various functions such as engine management, safety systems, infotainment, and driver assistance features.

Challenge:

XYZ Automotive faced several challenges in their automotive systems, such as managing complex electronic control units (ECUs), ensuring real-time responsiveness for critical systems like brakes and airbags, and integrating multiple software components seamlessly. They needed an RTOS that could handle these challenges while providing reliability, determinism, and efficient resource management.

Implementation Steps:

1. Selection of RTOS: After evaluating several RTOS options, XYZ Automotive chose **QNX Neutrino RTOS** for its proven track record in safety-critical applications, real-time capabilities, and scalability.

2. Safety-Critical Systems: QNX Neutrino was used to control safety-critical systems such as Anti-lock Braking System (ABS), Electronic Stability Control (ESC), and Airbag Control Units (ACUs). These systems require real-time responsiveness to ensure the safety of passengers in various driving conditions.

3. Multicore Support: Many modern vehicles have multicore processors to handle multiple tasks simultaneously. QNX Neutrino's support for multicore architectures allowed XYZ Automotive to leverage the full potential of their hardware while maintaining determinism and reliability.

4. Communication Protocols: The RTOS facilitated communication between ECUs using various protocols such as Controller Area Network (CAN), Local Interconnect Network (LIN), and FlexRay. These protocols ensured seamless data exchange among different vehicle systems.

5. Infotainment System: XYZ Automotive integrated the infotainment system with the RTOS to provide a seamless user experience. The RTOS managed multimedia playback, navigation, and connectivity features, ensuring smooth operation without impacting critical systems' performance.

6. Over-the-Air (OTA) Updates: To keep the vehicle software up to date, XYZ Automotive utilized QNX Neutrino's support for Over-the-Air (OTA) updates. This allowed them to remotely update software components, fix vulnerabilities, and introduce new features without requiring physical access to the vehicle.

7. Integration Testing: Before deployment, XYZ Automotive conducted extensive integration testing to ensure that all components, including the RTOS, ECUs, sensors, and actuators, worked seamlessly together. This testing included real-world scenarios to validate the system's performance and safety.

Results:

Improved Safety Features: The RTOS's real-time capabilities ensured quick response times for safety-critical systems like ABS, ESC, and airbags. This enhanced vehicle safety and reduced the risk of accidents.

Enhanced User Experience: Integration of the infotainment system with the RTOS provided a smooth and responsive user interface. Passengers could enjoy multimedia, navigation, and connectivity features without disruptions.

Efficient Resource Management: QNX Neutrino's efficient resource management optimized the use of multicore processors, ensuring that critical tasks had the necessary resources while minimizing latency.

Seamless Communication: The RTOS facilitated seamless communication between ECUs using various protocols. This enabled different vehicle systems to exchange data effectively, enhancing overall vehicle performance.

Remote Updates and Maintenance: OTA updates supported by the RTOS allowed XYZ Automotive to remotely update vehicle software. This streamlined maintenance processes and ensured that vehicles could receive updates without requiring physical visits to service centers.

Compliance with Safety Standards: By using QNX Neutrino, XYZ Automotive ensured compliance with automotive safety standards such as ISO 26262. The RTOS provided a reliable and deterministic platform for safety-critical applications.

Conclusion:

By incorporating QNX Neutrino RTOS into their automotive systems, XYZ Automotive successfully addressed the challenges of managing complex ECUs, ensuring real-time responsiveness for safety systems, and integrating multiple software components seamlessly. The RTOS improved vehicle safety, enhanced user experience, optimized resource management, enabled efficient communication between ECUs, supported OTA updates, and ensured compliance with safety standards.

This case study demonstrates the benefits of using a Real-Time Operating System like QNX Neutrino in automotive applications. XYZ Automotive's success in leveraging RTOS capabilities serves as an example for other car manufacturers seeking to enhance their vehicles' performance, safety, and user experience through advanced software solutions.

Case Study: Development of Mobile Operating System

Introduction:

An operating system is system software that manages hardware and software i.e. it is an interface between user and computer hardware. Each mobile has consists of an operating system. A mobile operating system is also called as mobile OS which operating system that allows devices such as phones, tablets and many other handheld devices to run applications and programs. Phone starts up when a device is turned on. It has included the features of a personal computer operating system as well.

The Timeline of Mobile Operating System:

- 1973-1993 – Mobile phones used embedded systems.
- 1994 – The first smartphone with touchscreen, email, PDA features.
- 1996 – Palm OS was introduced.
- 1998 – Symbian OS was developed by Symbian Ltd.
- 1999 – Nokia S40 platform was introduced.
- 2000 – Symbian OS on a smartphone with the launch of Ericsson R380.
- 2001 – Kyocera 6035, smartphone with Palm OS.
- 2002 – Introduction of Microsoft's Windows CE (Pocket PC) , BlackBerry released its first smartphone.
- 2005 – Nokia introduced Maemo OS.
- 2007 – Apple iPhone with iOS was introduced, Google formed Open Handset Alliance (OHA).
- 2008 – OHA released Android.
- 2009 – Palm introduced WebOS, Samsung announced Bada OS.
- 2010 – Windows Phone was released.
- 2011 – A mobile Linux distribution MeeGo was introduced.
- 2012 – Apple released iOS 6.
- 2013 – BlackBerry released OS, Apple released iOS 7, Google released Android KitKat 4.4.
- 2014 – Microsoft released Windows Phone 8.1, Apple released iOS 8, BlackBerry released 10.3, Google released Android 5.0 Lollipop.
- 2015 – Google released Android 5.1 Lollipop, Apple released iOS 9, Google released Android 6.0 Marshmallow, Microsoft released Windows 10 Mobile.
- 2016 – Microsoft released Windows 10 Mobile Anniversary update, Apple announced iOS 10, Google released Android 7.0 Nougat, Apple released iOS 10, Tizen released Tizen 3.0, BlackBerry released BlackBerry 10.3.3.
- 2017 – Samsung launched Android–based Samsung Galaxy S8, Microsoft released Windows 10 Creators update, Samsung introduced Tizen 4.0, Google released Android 8.0 Oreo, BlackBerry announced Android-based BlackBerry Secure, Apple introduced iPhone 8, iPhone X, iOS 11.

Some of the most common and current mobile operating system is as follows:

1. ANDROID:

It was developed by Google based on Linux kernel. It is the most popular OS in mobile phone. It is free, open source software and primarily designed for touchscreen mobiles and other electronic devices. Android was founded by Andy Rubin, Rich Miner, Nick Sears and Chris White in October 2003, in Palo Alto, California. In July 2005, Google was acquired Android and Andy Rubin, Rich Miner and Chris White joined Google.

Andy Rubin led a team to developed a mobile device platform powered by Linux kernel. In 2007, the green Android logo was designed by Irina Blok who was a graphic designer. Android mobile operating system was released various versions named after dessert or sweet items.

TABLE: ANDROID VERSIONS

Android Version	**Codename**	**Release Date**	**Key Features**
1.0	-	September 2008	Initial release with basic functionalities
1.5	Cupcake	April 2009	On-screen keyboard, video recording
1.6	Donut	September 2009	Quick Search Box, camera enhancements
2.0 - 2.1	Eclair	October 2009	Multiple accounts, Exchange support
2.2	Froyo	May 2010	Performance optimizations, Adobe Flash support
2.3	Gingerbread	December 2010	UI refinements, NFC support
3.0 - 3.2	Honeycomb	February 2011	Tablet-specific, holographic UI
4.0	Ice Cream Sandwich	October 2011	Unified UI, Face Unlock
4.1 - 4.3	Jelly Bean	July 2012	Project Butter, Google Now
4.4	KitKat	October 2013	Optimizations for low-spec devices
5.0 - 5.1	Lollipop	November 2014	Material Design, improved notifications
6.0	Marshmallow	October 2015	Doze mode, app permissions
7.0 - 7.1	Nougat	August 2016	Split-screen, improved notifications
8.0 - 8.1	Oreo	August 2017	Picture-in-picture, Project Treble
9	Pie	August 2018	Adaptive Battery, Gesture navigation
10	Quince Tart	September 2019	Dark mode, enhanced privacy controls
11	Red Velvet Cake	September 2020	Chat bubbles, media controls
12	Snow Cone	October 2021	Material You design, privacy dashboard
13	Tiramisu	September 2022	Private Compute Core, Sound Notifications
14	Upside Down Cake	September 2023	Smart Charging, Personalized Suggestions

2. IPHONE / IOS (APPLE Phone):

It was originally developed by Apple co . exclusively for its hardware. It is the most popular mobile operating system after Android OS. This mobile operating system was unveiled in 2007 for the iPhone smart phone. iOS supports many version of Apple's own manufactured devices such as iPhone, iPad and iPod Touch. The current version is iOS11, which was released on September 19, 2017. It is now available for all iOS smart phone devices.

TABLE : IPHONE VERSIONS

iOS Version	Release Date	Key Features
iOS 1.0	June 2007	Core functionalities, Safari browser, Google Maps, iTunes integration
iOS 2.0	July 2008	App Store, Push Notifications, MobileMe
iOS 3.0	June 2009	Copy and paste, MMS, Spotlight Search, Voice Control
iOS 4.0	June 2010	Multitasking, FaceTime, iBooks, Folders
iOS 5.0	October 2011	Notification Center, iMessage, iCloud, Siri, Reminders
iOS 6.0	September 2012	Apple Maps, Passbook, Facebook integration, Siri improvements
iOS 7.0	September 2013	Redesigned UI (flat design), Control Center, AirDrop, Siri updates
iOS 8.0	September 2014	Continuity, Health app, Apple Pay, QuickType keyboard
iOS 9.0	September 2015	Proactive Assistant, Multitasking on iPad, Apple Music
iOS 10.0	September 2016	Redesigned Lock Screen, Home app, Siri improvements
iOS 11.0	September 2017	Files app, ARKit, Do Not Disturb while driving, Siri improvements
iOS 12.0	September 2018	Screen Time, Memoji, Group FaceTime, Siri Shortcuts
iOS 13.0	September 2019	Dark Mode, Sign in with Apple, Photos and Camera updates
iOS 14.0	September 2020	App Library, Widgets, Picture-in-Picture, Privacy enhancements
iOS 15.0	September 2021	Focus mode, FaceTime improvements, Live Text, Notification Summary
iOS 16.0	September 2022	Further privacy enhancements, new widget designs, and more
iOS 17.0	September 2023	Enhanced AI capabilities, redesigned Control Center, advanced privacy controls, improved notifications management

3. BLACKBERRY Pone:

BlackBerry Ltd. is a Canadian multinational company. Blackberry is originally created by Mike Lazaridis and Douglas Fregin as Research In Motion in 1984. The BlackBerry h a s refers to the unique physical keyboard design on early devices. It is based on Android Open Source Project. It is specialized in mobile productivity and secure communications. BlackBerry is branded for tablets, smartphones and many more products. Blackberry Secure operating system is based on Android Open Source Project, which was announced on August 2017. Blackberry Secure version 1.x (based on Android "Marshmallow" 6.x and "Nougat" 7.x) is the latest version

TABLE: BLACKBERRY VERSIONS

Version	Release Date	Features / Information
1	January 1999	Released for the BlackBerry 850 pager, debut of BlackBerry OS
4	March 2002	Released on the BlackBerry 5810 smartphone, first support for Java
5.0	August 4, 2009	Introduced on the BlackBerry Bold 9000, expanded to other devices
6.0	April 2010	Announced in April 2010, released with new WebKit-based browser
7.0	August 2011	Featured performance improvements, "Liquid Graphics," improved browser
7.1	January 2012	Update to 7.0 with new features: Wi-Fi hotspot, Wi-Fi calling, FM radio support, Blackberry Tag

Conclusion:

There are varieties of Mobile phones available in market with various latest mobile Operating Systems. Year by year the versions are released and new mobile operating systems are developed by the companies for the users.