File systems

Storage Management

Programs has to be loaded to main memory for execution.

But, main memory is usually too small to accommodate all the data and programs permanently.

So, the computer system must provide secondary storage to back up main memory.

Most computers use disk as the most common secondary storage device for both programs and data.

The FILE SYSTEM provides the mechanism for:

- Storage of data and programs on the disk.
- Access to data and programs on the disk.

Files are mapped by OS to physical devices.

→ A collection of related information defined by its creator.

Files are normally organized into directories for ease of use.

Different storage devices vary in many aspects.

E.g.

- Some devices transfer a character or a block of characters at a time.
- Some can be accessed only sequentially.
- Some can be accessed randomly.
- Some transfer data synchronously.
- Some transfer data asynchronously.
- Some are dedicated while some are shared.
- Some can be read-only.
- Some can be read-write.

Because of all this device variation, the operating system needs to provide a wide range of functionality to applications, to allow them to control all aspects of the devices.

Objectives:

- Understand the functions of file systems.
- Understand the interfaces to file systems.
- Understand file system designs, access methods and directory structures.

Concept of File

Different storage media can be used for storing information in a computer.

E.g. Magnetic disks, magnetic tapes, and optical disks, etc.

The operating system abstracts from the physical properties of its storage devices to define a logical storage unit –

The File

Files are mapped by the operating system onto physical devices.

Usually Non-Volatile

Contents are not lost after power is OFF

Definition:

A file is a named collection of related information that is recorded on secondary storage.

From a User's perspective:

- A file is the smallest allotment of logical secondary storage.
- Data cannot be written to secondary storage unless they are within a file.

Files represent both data and programs.

- can be numeric, alphabetic, alphanumeric, or binary.
- may be free form, such as text files, or may be formatted rigidly.

In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user.

The information in a file is defined by its creator.

Many different types of information may be stored in a file:

E.g.

- Source Programs
- Object Programs
- Executable Programs
- Numeric Data
- Text
- Graphic Images
- Sound Recordings
- Video Recordings

File Attributes

Naming a file:

- A file is named, for the convenience of its human users, and is referred to by its name.
- A name is usually a string of characters.

E.g. myfile.c

- When a file is named, it becomes independent of the process, the user, and even the system that created it.

A file's attributes vary from one OS to another but typically consist of these:

1. **Name** - The symbolic file name is the only information kept in human readable form.
2. **Identifier** - This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
3. **Type** - This information is needed for systems that support different types of files.
4. **Location** - This information is a pointer to a device and to the location of the file on that device.
5. **Size** - The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.

A file's attributes vary from one OS to another but typically consist of these:

6. **Protection / Permission** - Access-control information determines who can do reading, writing, executing, and so on.
7. **Creation Date** – The date on which the file was created.
8. **Last Modified Date** – The date on which the file was last modified.
9. **Owner** – The information about who is the owner of the file.

A file's attributes vary from one OS to another but typically consist of these:

1. Name
2. Identifier
3. Type
4. Location
5. Size
6. Protection / Permission
7. Creation Date
8. Last Modified Date
9. Owner

The attributes of the file are stored in the File Control Block (FCB)

Creating a file

Steps for creating a file:

1. Check if space is available in the file system to create the new file and find space for it.
2. Make an entry for the new file in the directory.

Writing a file

Steps for writing a file:

1. Make a system call specifying both the name of the file and the information to be written to the file.
2. With the given name of the file, the system searches for the file in the directory and locates it.
3. A write pointer is maintained which points to the location in the file from where the next write must take place.
4. After the writing is done, the write pointer is updated.

Reading a file

Steps for reading a file:

1. Make a system call specifying both the name of the file and where (in memory) the next block of the file should be read.
2. With the given name of the file, the system searches for the file in the directory and locates it.
3. A read pointer is maintained which points to the location in the file from where the next read must take place.
4. After the reading is done, the read pointer is updated.
5. Mostly read and write operations use the same pointer.

Reading a file

Steps for reading a file:

1. Make a system call specifying both the name of the file and where (in memory) the next block of the file should be read.
2. With the given name of the file, the system searches for the file in the directory and locates it.
3. A read pointer is maintained which points to the location in the file from where the next read must take place.
4. After the reading is done, the read pointer is updated.
5. Mostly read and write operations use the same pointer.

Repositioning within a file (File Seek)

Steps for repositioning within a file:

1. The directory is searched for the appropriate file.
2. The current-file-position pointer is repositioned to a given value.

Repositioning within a file need not involve any actual I/O.

Deleting a file

Steps for deleting a file:

1. The directory is searched for the named file.
2. Once the directory is found, all the file space occupied by the file is released so that it can be used by other files.

Truncating a file

The user may want to erase the contents of a file but keep its attributes.

Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged.

Steps for truncating a file:

1. Keep all the attributes of the file unchanged.
2. Except- The file length. Reset it to zero.
3. Release the file space.

Common file operations

- Creating a file.
- Writing a file.
- Reading a file.
- Repositioning within a file.
- Deleting a file.
- Truncating a file.

Other file operations

- Appending new information to the end of an existing file.
- Renaming an existing file.
- Editing existing data on a file.
- Saving a file.
- Creating a copy of a file.
- Hiding a file.
- Sending a file.
- Printing a file.

Etc.

The open ( ) system call

Most of the common file operations that we discussed involve searching the directory for the entry associated with the named file.

To avoid this constant searching, many systems require that an open() system call be made before a file is first used actively.

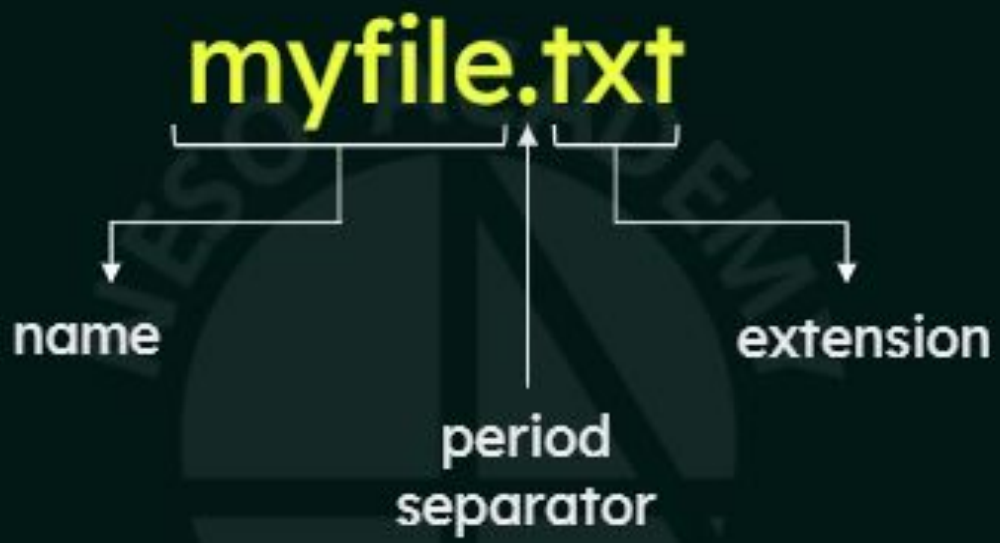
- The operating system keeps a small table, called the open-file table, containing information about all open files.
- When a file operation is requested, the file is specified via an index into this table, so no searching is required.
- When the file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table.

File Types

When a file system or an Operating System is designed, the file types that the OS should recognize and support must be considered.

If an operating system recognizes the type of a file, it can then operate on the file in reasonable ways.

The common technique for implementing file types is to include the type as part of the file name.


myfile.txt
name
period
separator
extension

Common File Types

File Type	Usual Extension	Function
Executable	exe, com, bin or none	Ready-to-run machine-language program
Object	obj, o	Compiled machine language, not linked
Source code	c, cpp, pas, java, asm, py	Source code in various languages
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word processor	wp, doc, rtf	Various word processor formats
Library	lib, a	Libraries of routines for programmers
Print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
Archive	arc, zip, tar	Related files grouped into one file sometimes compressed for archiving or storage
Multimedia	mp3, mp4, avi	Binary file containing audio or A/V information

Other ways of identifying file types

In Mac OS X:

Each file has a 'type' specified with it to identify its type.

E.g.

For text files the type is – TEXT

For applications, the type is – APPL

In UNIX systems:

It uses a crude magic number stored at the beginning of some files to indicate roughly the type of the file.

Access Methods

The information in files can be accessed in several ways. Some systems provide just one access method for files while other systems may support many access methods.

The two most common access methods are:

1. Sequential Access
2. Direct Access

Sequential Access

- This is the simplest access method.
- Information in the file is processed in order, one record after the other.
- Most common editors, compilers, etc. access files in this method.

READ OPERATION – read next - reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.

WRITE OPERATION – write next - appends to the end of the file and advances to the end of the newly written material (the new end of file).

Direct Access (Relative Access)

- A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order.
- In Direct Access – the file is viewed as a numbered sequence of blocks or records.
- Hence, there are no restrictions on the order of reading or writing for a direct-access file.
- Direct-access files are of great use for immediate access to large amounts of information.
- E.g. Databases

READ OPERATION	– read n -	reads from block number 'n' from the file.
WRITE OPERATION	– write n -	writes to block number 'n' of the file.

The block number provided by the user to the operating system is normally a relative block number which is an index to the relative beginning of the file.

Directory Structure

The file systems of computers can be extensive. Systems need to store huge number of files on disks.

So, to manage this huge amount of data, they must be organized properly.

Directories are used for organizing this data.

Storage Structure

- A disk or any large storage device can be used entirely for a file system.
- But it is better to have multiple file systems on a disk or use parts of a disk for a file system and other parts for other things like swap space, raw disk space, etc.

These parts are called –

PARTITIONS

SLICES

MINIDISK

A file system can be created on each of these parts of the disk.

These parts can also be further combined to form larger structures called

VOLUMES

A file system can be created on these as well.

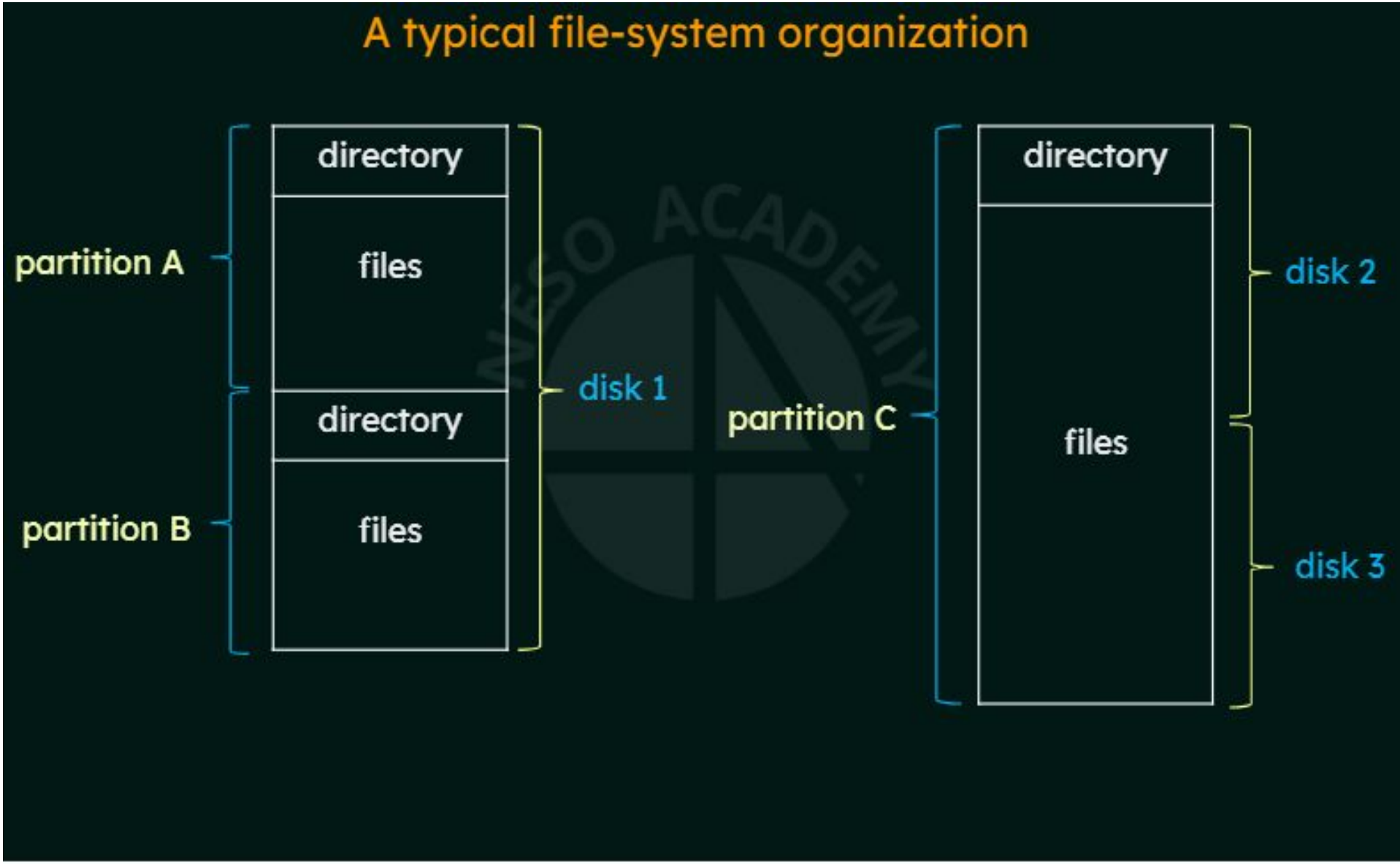
Storage Structure

- Each Volume can be thought of as a virtual disk.
- Volumes can store multiple Operating Systems allowing the system to boot and run more than one Operating Systems.
- Each volume that contains a file system must also contain information about the files in the system.

This information is kept in entries in a device directory.

A directory records information such as
- name
- location
- size
- type

for all files on that volume.


A typical file-system organization
directory
files
directory
files
partition A
partition B
disk 1
partition C
directory
files
disk 2
disk 3

Directory Overview

Operations that can be performed on a directory:

- Search for a file.
- Create a file.
- Delete a file.
- List a directory.
- Rename a file.
- Traverse the file system.

Single-Level Directory

Single-level directory is the simplest directory structure.

Here, all files are contained in the same directory.

Advantages:

- Implementation is easy as there is just a single directory.
- File operations like creating, reading, deleting, repositioning, renaming, etc. are easily done.
- Searching for files will be fast if the number of files are less and are of small sizes.

Limitations:

- Problems start arising when the number of files increases or when the system has more than one user.
- All the files must have unique names since all the files are in the same directory.
  E.g. If two users name their files 'test', then the unique-name rule is violated.
- As the number of files increases:
  - Even a single user will find it difficult to remember all the names of the files.
  - Searching for a file will become difficult.
  - Keeping track of all the files will be a very difficult task.

Two-Level Directory

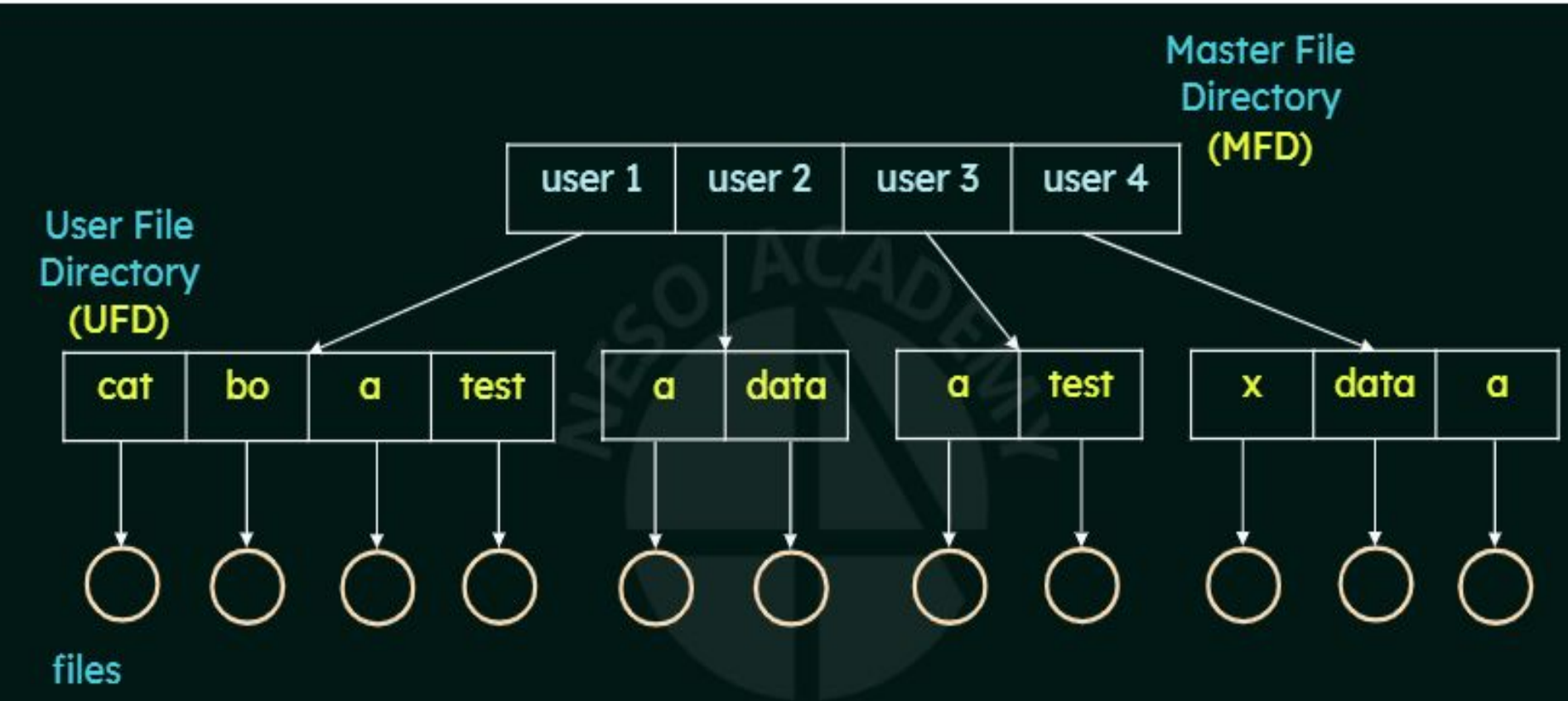
Here, each user has his/her own directory called the

User File Directory (UFD).

Information about each UFD is stored in a system directory called the

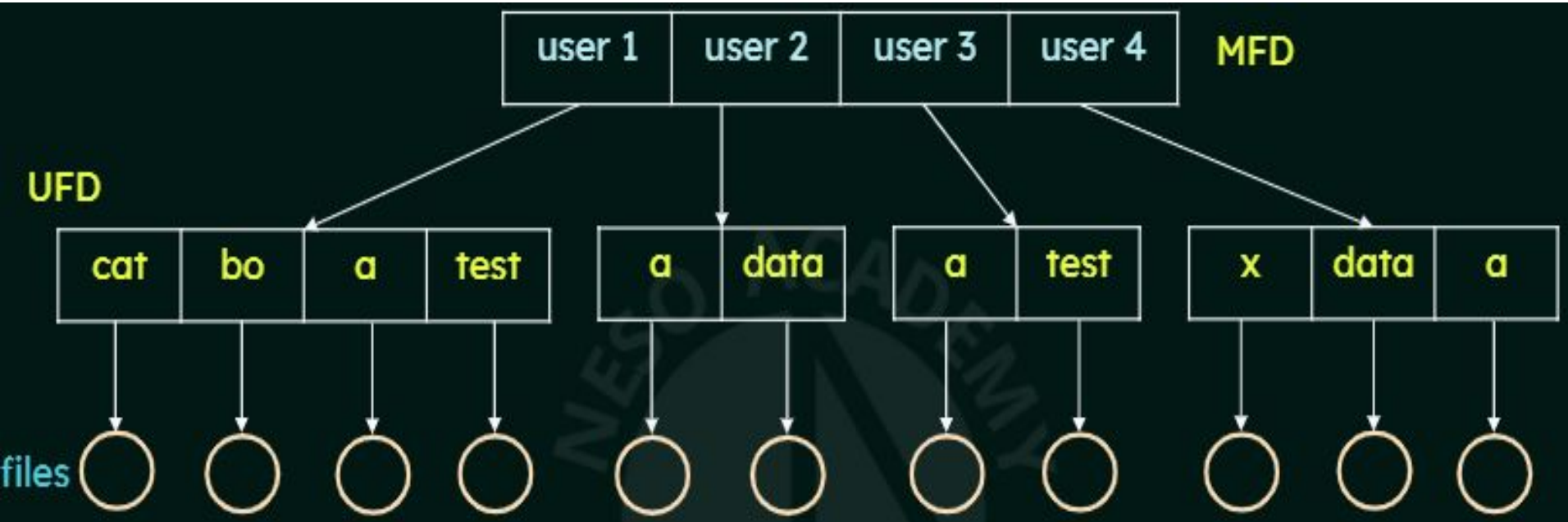
Master File Directory (MFD).

The MFD is indexed by user name or account number, and each entry points to the UFD for that user.



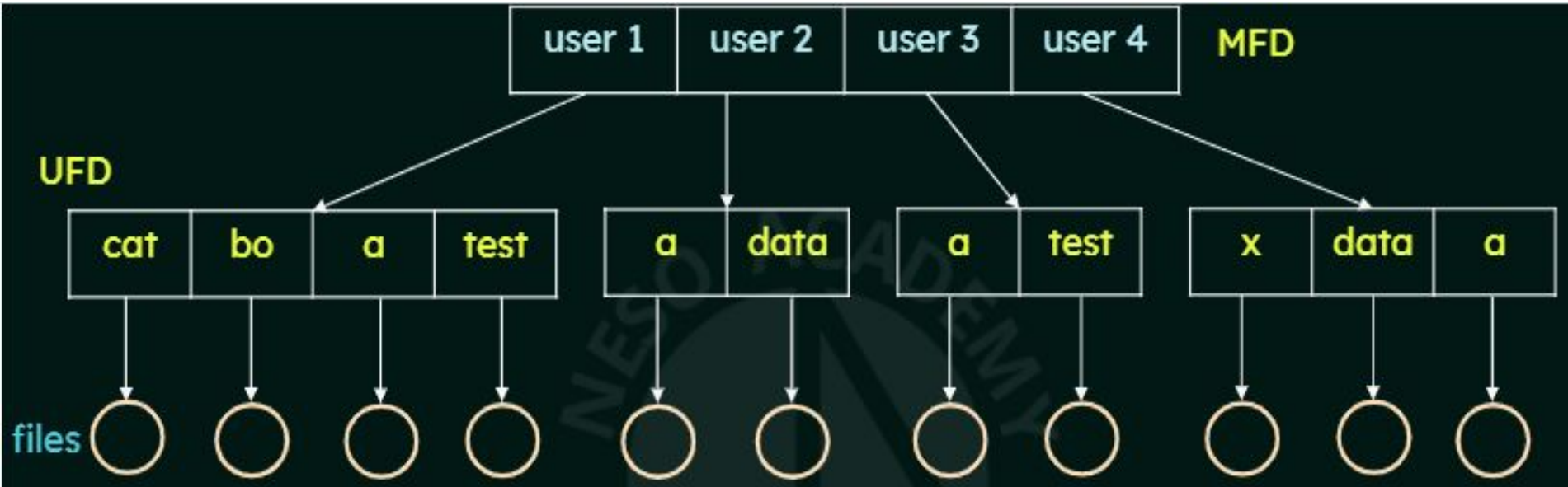
When a user refers to a particular file, only his own UFD is searched.

Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.



Problems:

- Users are isolated from each other.
- When two or more users wants to cooperate on a task and access the same file, it becomes difficult as the system doesn't permit this.

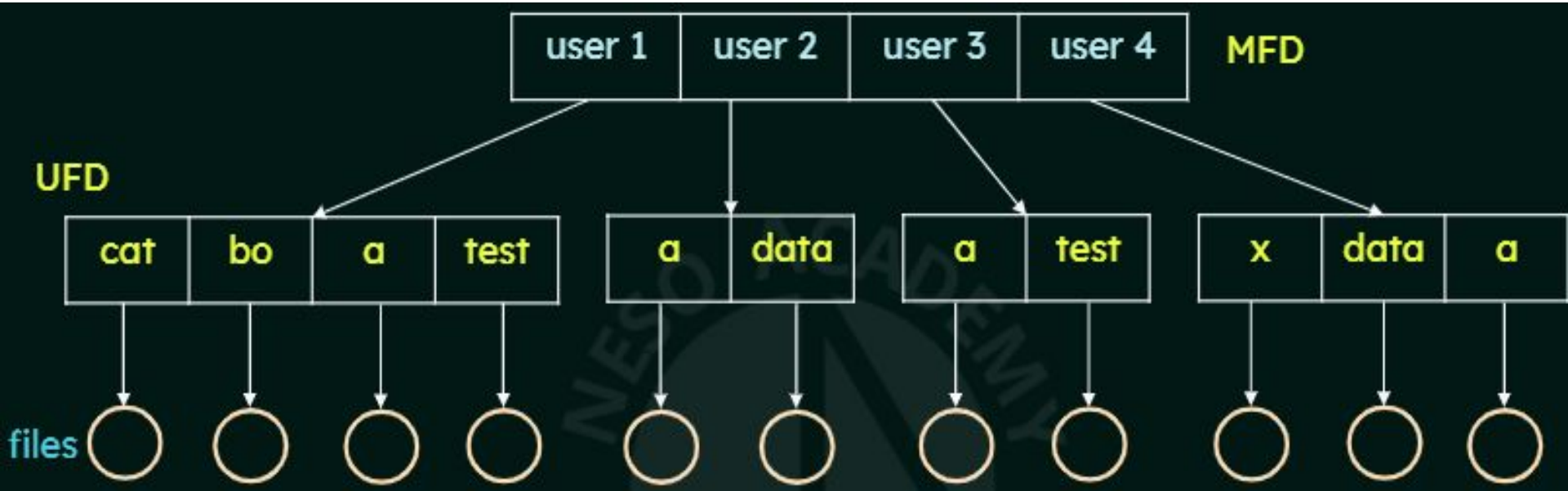


If users are allowed to access other users' files, then the files must be named with their complete path which specifies who is the owner of the file.

E.g. /user1/test

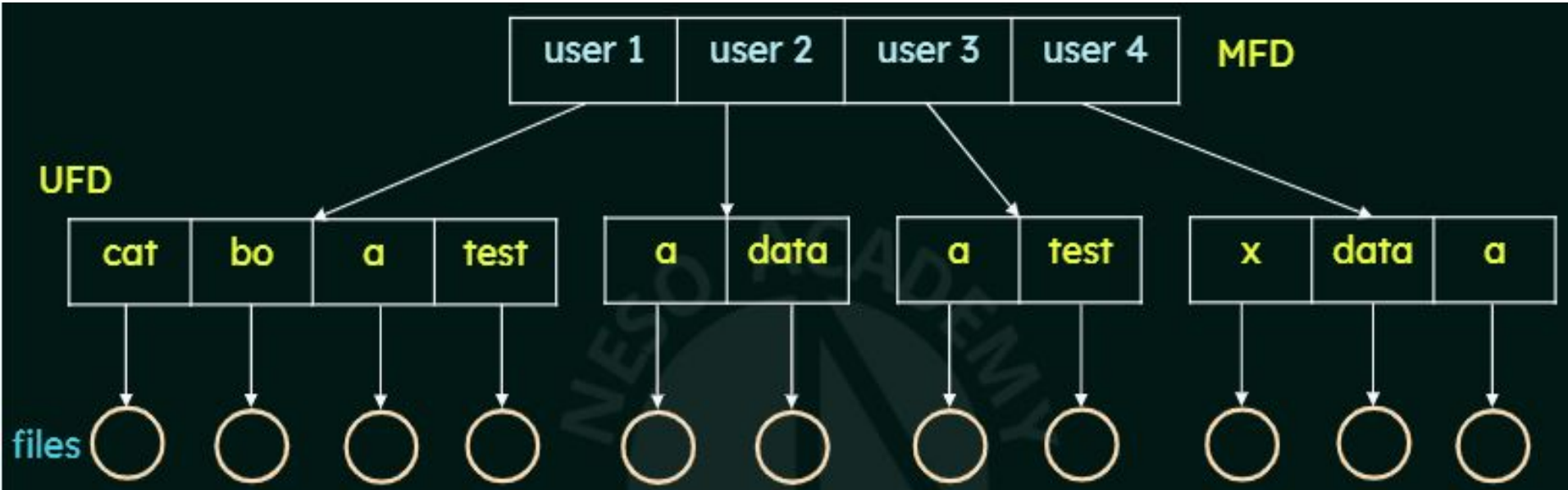
/user3/test

/user4/data



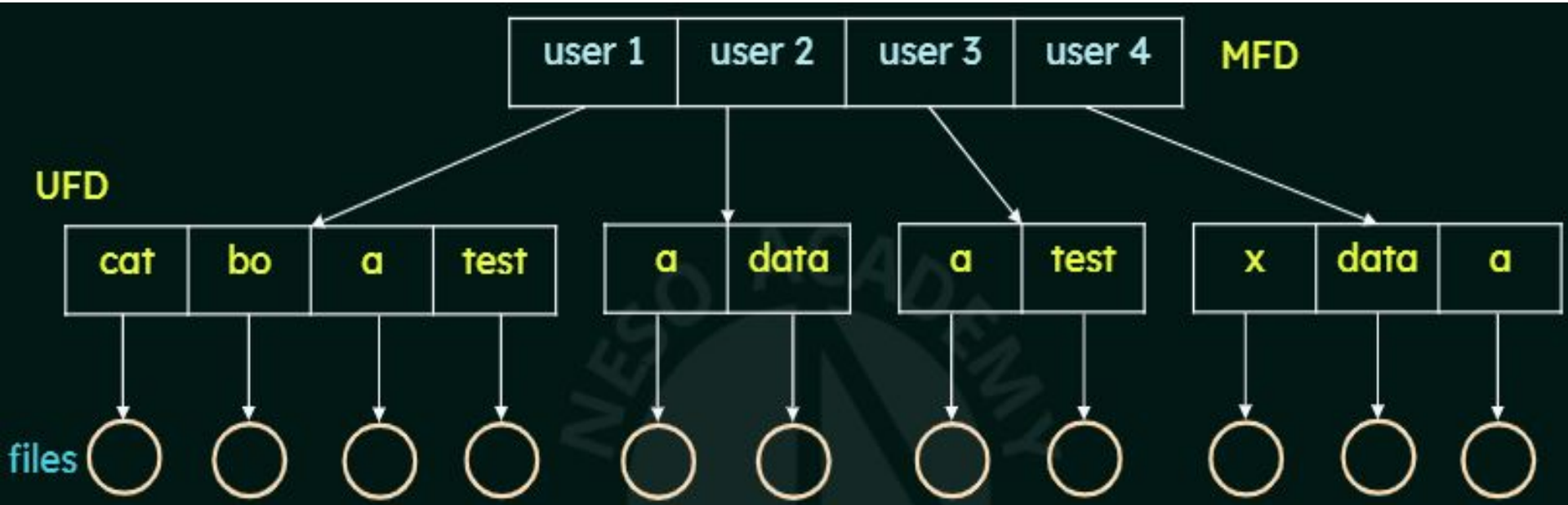
Every system has its own syntax for naming files and directories.

E.g. In MS-DOS



Another Problem:

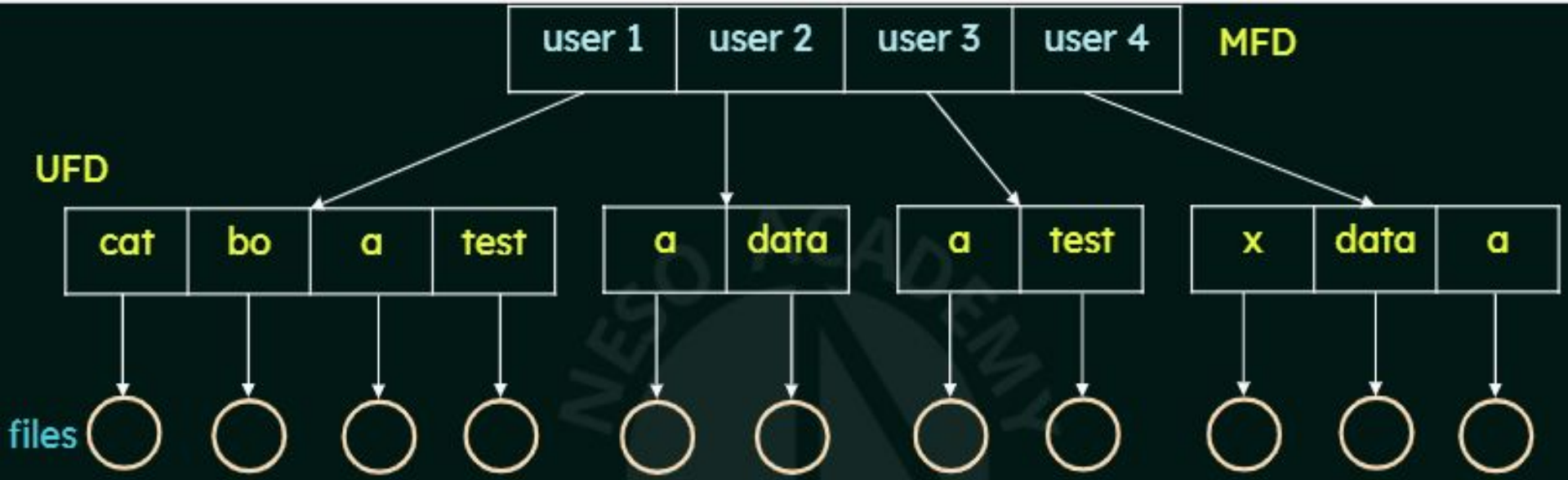
- There are system files that has to be used by several users.
- When these files are to be accessed by some user, it will be searched for in that user's UFD. But it won't be available there.



Solution to the Problem:

- Copy the system files to each UFD.

  - This would waste an enormous amount of space.



Solution to the Problem:

- A special user directory is defined to contain the system files (E.g. user 0)
  - Whenever a file name is given to be loaded, the OS first searches the local UFD. If the file is found, it is used. If it is not found, the system automatically searches the special user directory that contains the system files.

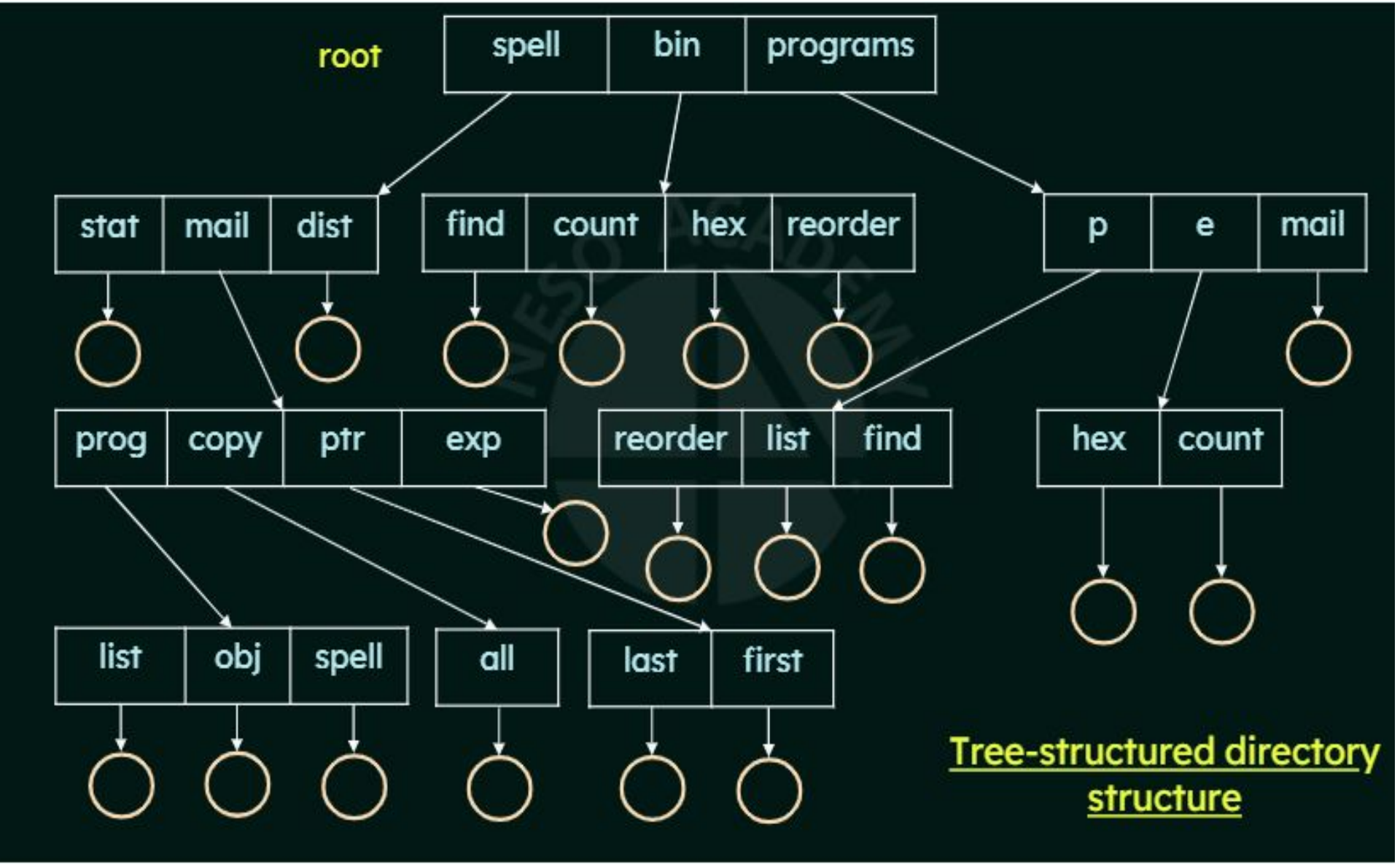
Tree-Structured Directories

The natural generalization of a two-level directory structure is to extend the directory structure to a tree of arbitrary height.

This generalization allows users to create their own subdirectories and to organize their files accordingly.

A tree is the most common directory structure.

The tree has a root directory, and every file in the system has a unique path name.



Tree-structured directory structure

Current directory

- Each process has a current directory.
- The current directory should contain most of the files that are of current interest to the process.
- When reference is made to a file, the current directory is searched.
- If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file.

Path names

Two Types:

1. **Absolute:**

   Begins at the root and follows a path down to the specified file, giving the directory names on the path.

2. **Relative:**

   Defines a path from the current directory.

E.g.

If the current directory is **root/spell/mail**

Absolute path name: **root/spell/mail/ptr/first**

Relative path name: **ptr/first**

Deleting a directory

The policy to be followed while deleting a directory has to be decided.

If the directory is empty its entry in the directory that contains it can simply be deleted.

If the directory is not empty:

- Some systems will not allow deleting a directory unless it is empty. So to delete directory in such systems, all its files and subdirectories must be deleted first.

- Some systems provide commands to delete directories irrespective of whether they are empty or not.

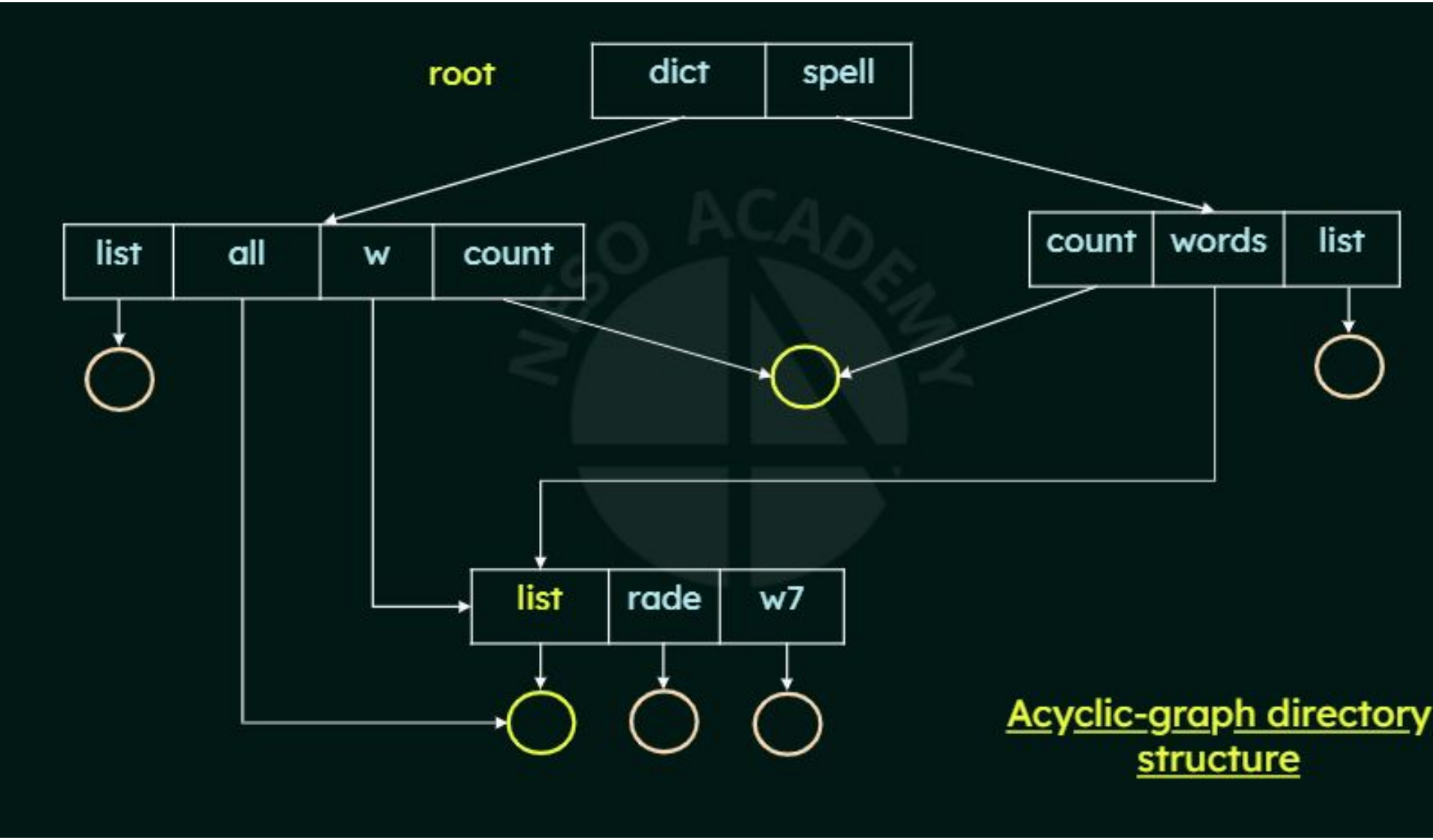
  E.g. The rm –r command in UNIX

Acyclic-Graph Directories

Need for Acyclic-Graph directories – SHARING.

A tree structure prohibits the sharing of files or directories.

An acyclic graph — that is, a graph with no cycles — allows directories to share subdirectories and files.


root
dict
spell
list
all
w
count
count
words
list
list
rade
w7
Acyclic-graph directory structure

- When people are working as a team, all the files they want to share can be put into one directory.
- The UFD of each team member will contain this directory of shared files as a subdirectory.

Implementation of shared files and subdirectories:

1. Links: A link is effectively a pointer to another file or subdirectory.

2. Duplicating: Duplicate all information about shared files in both sharing directories. Thus, both entries are identical and equal. A major concern here is in maintaining consistency when the files are modified.

Problems to be considered in acyclic-graph directory structures

- **Same file with many paths and names:**

  A file may now have multiple absolute path names. Also, different file names may refer to the same file. This may yield wrong count of files while trying to take a count of the number of files.

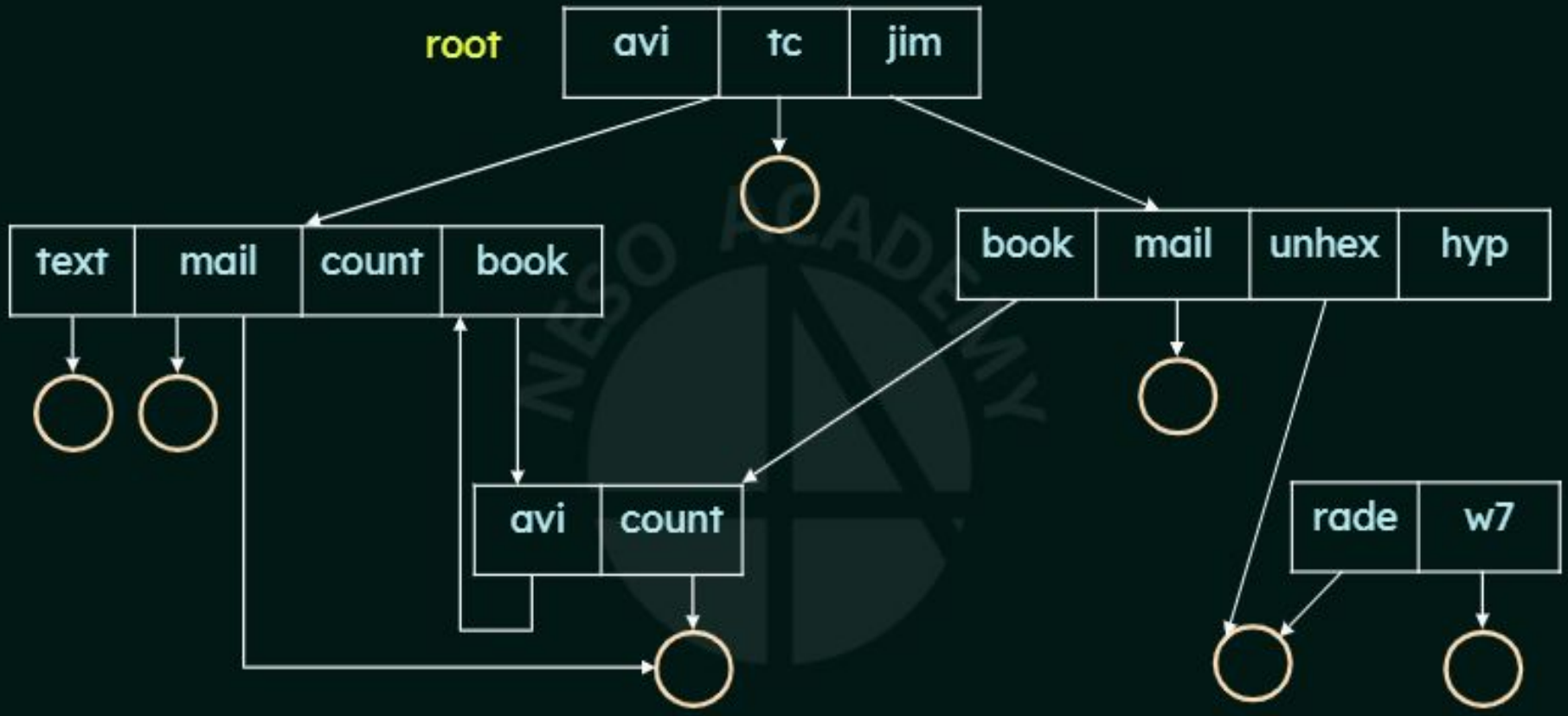
- **Deleting of shared files:**
  - If the file is deleted whenever someone deletes it, it will lead to dangling pointers to the non-existent file.
  - If shared files are implemented using symbolic links, then deletion of a file by one user will delete only the link and won't affect the original file. But if the original file is deleted, it will leave the links dangling. Later, these links can be removed.
  - Another approach - preserve the file until all references to it are deleted.

General Graph Directory

Problem with Acyclic-graph directories – It has to be ensured that there are no cycles.

Advantage of this – It is relatively simple to traverse the graph.

- If we start with a two-level directory and allow users to create subdirectories, a tree-structured directory results.
- If we continue adding files and subdirectories in this way, the tree structure is preserved and continues.
- But when we add links to an existing tree-structured directory, the tree structure is destroyed, resulting in a simple graph structure (which may or may not contain cycles).



General graph directory

Problems to be considered in general graph directory structures

Searching for files:

- If cycles are allowed to exist in the directory, we want to avoid searching any component twice, for reasons of correctness as well as performance.
- A poorly designed algorithm might result in an infinite loop continually searching through the cycle and never terminating.
- One solution is to limit arbitrarily the number of directories that will be accessed during a search.

Problems to be considered in general graph directory structures

Deleting files:

- In an acyclic-graph directory structure, if the reference count is 0 for any file, then that means there are no references to that file or directory and it can be safely deleted.

- But when cycles exist, the reference count may not be 0 even when it is no longer possible to refer to a directory or file because of the possibility of self-referencing (cycle).

- To avoid this we can use the scheme of Garbage Collection.

Garbage Collection

It is a scheme to determine when the last reference has been deleted and the disk space can be reallocated.

Steps:

1. Traverse the entire file system and mark everything that can be accessed.
2. Traverse again a second time and free up all the unmarked cases from the first step.

Garbage collection for a disk-based file system, however, is extremely time consuming and is thus seldom attempted.

File-System Mounting

Just as a file must be opened before it is used, a file system must be mounted before it can be available to processes on the system.

A directory structure can be built out of multiple volumes as we have discussed earlier. These must be mounted to make them available within the file-system name space.

Mounting is a process by which the OS makes the files and directories on a storage device available for users to access within the file-system.

Mounting procedure

- The Operating System is given the name of the device and the mount point.

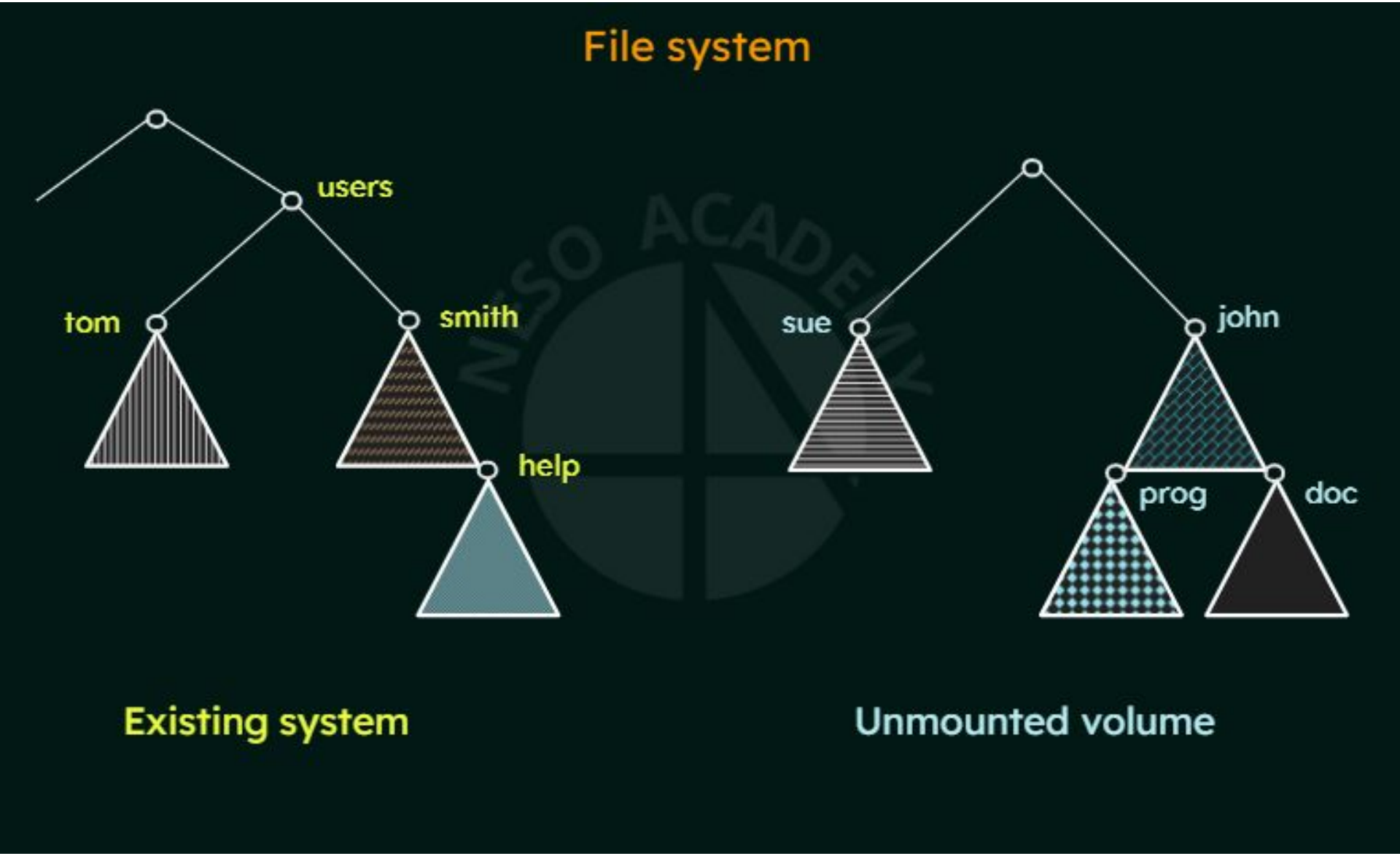
  the location within the file structure where the file system is to be attached.
- E.g. In UNIX system:

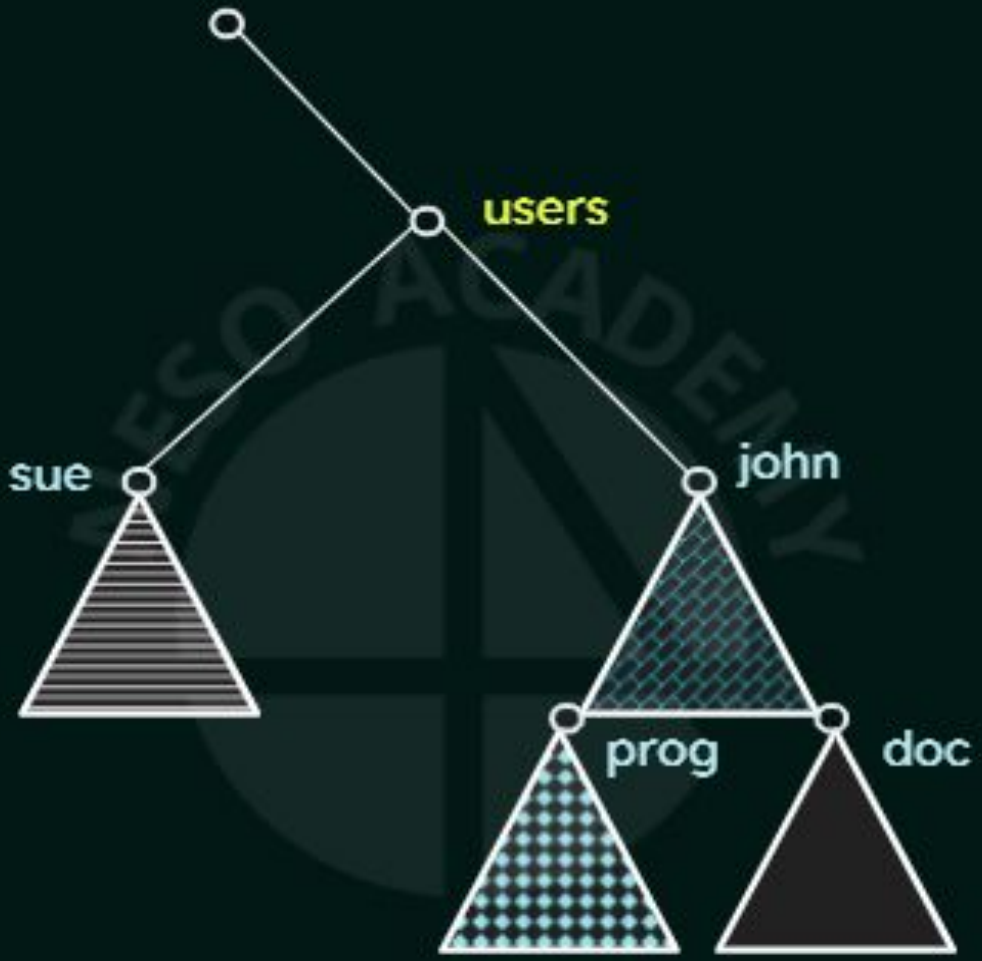
  A file system containing a user's home directories might be mounted as /home

  To access the directory structure withing that file system - /home/john

  If it was mounted at /users

  Then the path would be /users/john
- The operating system verifies that the device contains a valid file system.
- The operating system notes in its directory structure that a file system is mounted at the specified mount point.


File system
users
tom
smith
help
sue
john
prog
doc
Existing system
Unmounted volume



Mount Point

File Sharing

File sharing is very desirable for users who want to collaborate and to reduce the effort required to achieve a computing goal.

Therefore, user-oriented operating systems must accommodate the need to share files in spite of the inherent difficulties.

Multiple Users

The issues of

File sharing, file naming, and file protection

become pre-eminent when an operating system accommodates multiple users.

Given a directory structure that allows files to be shared by users, the system must mediate the file sharing.

The system can either:

- Allow a user to access the files of other users by default

or

- Require that a user specifically grant access to the files

How to implement sharing when there are multiple users

- The system must maintain more file and directory attributes than that are needed on a single-user system.

- Most systems follow the concept of file or directory owner (or user) and group.

OWNER - The user who can change attributes and grant access and who has the most control over the file.

GROUP – A subset of users who can share access to the file.

Example

In a UNIX System:

Owner of a file: Can issue all operations on a file.

Members of the file group: Can execute one subset of the above operations which is defined by the owner.

The Owner ID and Group ID of a given file or directory is stored with the other file attributes.

- When a user requests an operation on a file, the user ID can be compared with the owner attribute to determine if the requesting user is the owner of the file.
- Likewise, the group IDs can be compared.
- The result indicates which permissions are applicable.
- The system then applies those permissions to the requested operation and allows or denies it.

Remote File Systems

With the introduction and possibility of networks communication among remote computers became possible.

Networking allows the sharing of resources spread across a campus or even around the world.

One obvious resource to share is data in the form of files.

Methods of remote file-sharing

- Manually transferring files between system via programs like ftp (File Transfer Protocol).

- Using distributed file system (DFS):
  Here, data is stored on a server. The data is accessed and processed as if it was stored on the local client machine. The DFS makes it convenient to share information and files among users on a network in a controlled and authorized way.

- The World Wide Web:
  A browser is needed to gain access to the remote files, and separate operations (essentially a wrapper for ftp) are used to transfer files.

The Client-Server Model

Remote file systems allow a computer to mount one or more file systems from one or more remote machines.

Here,

The machine containing the files – THE SERVER

The machine seeking access to the files – THE CLIENT

The server:

- Declares that a resource is available to clients.
- Specifies which is the resource (which file).
- Specifies to which clients they are available.

A server can serve multiple clients, and a client can use multiple servers.

Client identification

A client can be specified by a network name or other identifiers like IP address.

But client identification is difficult as these addresses or identifiers can be spoofed or imitated.

As a result, an unauthorized client could be allowed access to the server.

- To have a more secure approach, encryption keys can be used.
- But the implementation is challenging due to issues like both the serve and client side needs to use the same encryption algorithms.
- Due to these issues and due to difficulties in solving them, unsecure authentication methods are most commonly used.

Example

In a UNIX system with NFS (Network File System):

- The User ID's of the client and server must match for file sharing to take place.
- E.g. If a user has ID – 1000 on the client and 2000 on the server, the authentication will be difficult.
- The server must either trust the client or present the correct/matching user ID.
- Once authentication is done:

  - The remote file system is mounted.

  - File operation requests are sent on behalf of the user across the network to the server via the DFS protocol.

  - The server checks if the user has the access rights to perform the specified operation.

  - The request is either allowed or denied.

Failure Modes

Local file systems can fail for a variety of reasons.

- Failure of the disk containing the file system
- Corruption of the directory structure or other disk-management information
- Disk-controller failure
- Cable failure
- Host-adapter failure

Most of these failures will cause a host to crash and an error condition to be displayed, and human intervention will be required to repair the damage.

Remote file systems have even more failure modes due to complexity of network systems and the required interactions between remote machines.

The network can be interrupted between two hosts as a result of

Although some networks have built-in capability to recover from these failures, including multiple paths between hosts, many do not.

Any single failure can thus interrupt the flow of DFS commands.

Example

A client is in the midst of using a remote file system.

- Suddenly a partitioning of the network, a crash of the server, or a scheduled shutdown of the server has occurred.
- The remote file system will no longer be reachable now.

What to do now ?

The system can:

- Either terminate all operations to the lost server.

OR

- Delay operations until the server is again reachable.

Recovery from failure

Maintain a state information at both client and server side:

- If both client & server maintain knowledge of their current activities and open files, then they can seamlessly recover from a failure.

Implementing a stateless DFS:

- Used in NFS.
- Used in situation where the server crashes but must recognize that it has remotely mounted exported file systems and opened files.
- It assumes that a client request for a file read or write would not have occurred unless the file system had been remotely mounted and the file had been previously open.
- The NFS protocol carries all the information needed to locate the appropriate file and perform the requested operation.

Consistency Semantics

Consistency semantics represent an important criterion for evaluating any file system that supports file sharing.

What is its use ?

These semantics specify how multiple users of a system are to access a shared file simultaneously.

They specify when modifications of data by one user will be observable by other users.

Example

UNIX Semantics

The UNIX file system uses the following consistency semantics:

- Writes to an open file by a user are visible immediately to other users that have this file open.
- One mode of sharing allows users to share the pointer of current location into the file. Thus, the advancing of the pointer by one user affects all sharing users. Here, a file has a single image that is accessed as an exclusive resource.
- Contention for this single image causes delays in user processes.

Example

Andrew File System (AFS) Semantics

The Andrew file system uses the following consistency semantics:

- Writes to an open file by a user are not visible immediately to other users that have the same file open.
- Once a file is closed, the changes made to it are visible only in sessions starting later. Already open instances of the file do not reflect these changes.
- Here, a file may be associated temporarily with several (possibly different) images at the same time. Consequently, multiple users are allowed to perform both read and write accesses concurrently on their images of the file, without delay.

Example

Immutable-Shared-Files Semantics

The Immutable-Shared-Files concept uses the following consistency semantics:

- Once a file is declared as shared by its creator, it cannot be modified.

- An immutable file has two key properties:
  1) Its name may not be reused.
  2) Its contents may not be altered.

Protection

Types of Access

Information stored in a computer system must be kept safe from:

1) Physical damage (Reliability)

File systems can be damaged by:

- Hardware problems
- Power surges/ failures
- Head crashes
- Dirt
- Temperature extremes
- Vandalism, etc.

Reliability is generally provided by duplicate copies of files.

2) Improper access (Protection)

- File systems must be allowed to be accessed only by authorized users.
- In a single user system, the protection can be done easily.
- But in multiuser system, other mechanisms are needed for proper protection of files.

Types of Access

- **Complete protection** can be provided by prohibiting access to everyone except the owner.

- **Free access** can be provided to everyone thus having no protection.

  The above two methods are too extreme for general use.

- **Controlled access** can be provided by limiting the types of file access that can be made by different users.

File operations that may be controlled

- **Read** - Read from the file.
- **Write** - Write or rewrite the file.
- **Execute** - Load the file into memory and execute it.
- **Append** - Write new information at the end of the file.
- **Delete** - Delete the file and tree its space for possible reuse.
- **List** - List the name and attributes of the file.

Other operations, such as **renaming**, **copying**, and **editing** the file, may also be controlled.

Protection

Access Control

The most common approach to the protection problem is to make access dependent on the identity of the user.

How can this be implemented ?

Using an Access Control List (ACL):

With each file and directory an access-control list (ACL) is associated specifying user names and the types of access allowed for each user.

Access Control List

- When a user requests access to a particular file, the operating system checks the access list associated with that file.
- If that user is listed for the requested access, the access is allowed.
- If not, a protection violation occurs, and the user job is denied access to the file.

Problems with this approach:

- Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.

- The directory entry, previously of fixed size, now needs to be of variable size, resulting in more complicated space management.

Solution to the problems

Use a condensed version of the access list.

Systems mostly recognize three classifications of users in connection with each file:

- **Owner** - The user who created the file is the owner.
- **Group** - A set of users who are sharing the file and need similar access is a group, or work group.
- **Universe** - All other users in the system constitute the universe.

Example

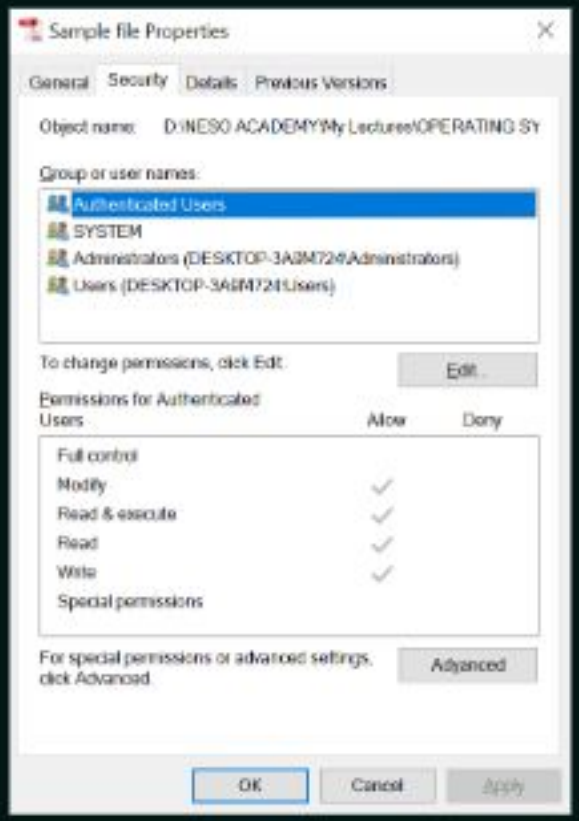
In a UNIX System

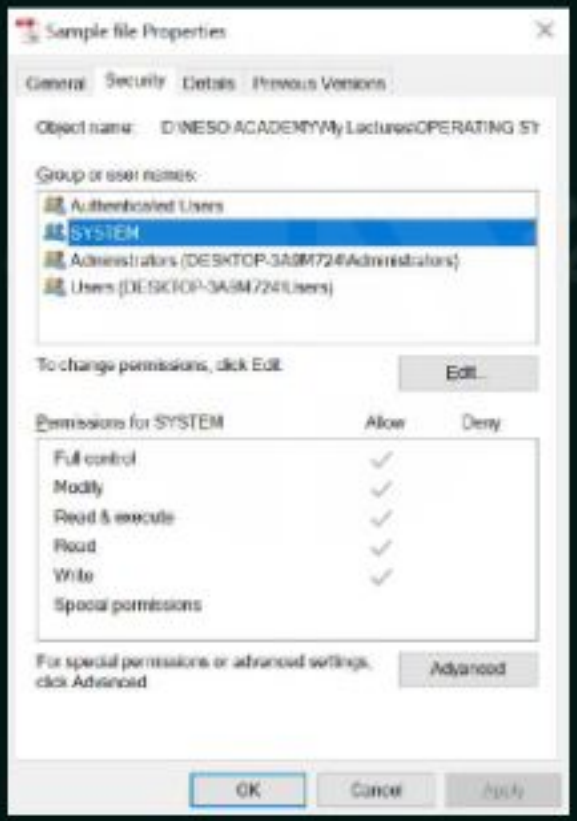
-rw-rw-r-	1 robert	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 robert	staff	512	Jul 8 09:30	private/
drwxrwxr-x	2 robert	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 robert	student	512	Aug 3 10:34	student-proj/
-rw-r--r--	1 robert	staff	9423	Feb 23 04:45	program.c
-rwxr-xr-x	1 robert	staff	20471	Feb 24 12:00	program
drwx--x--x	4 robert	faculty	512	Jul 30 11:14	lib/
drwx------	3 robert	staff	1024	Aug 26 05:31	mail/
drwxrwxrwx	3 robert	staff	512	Jul 8 07:35	test/

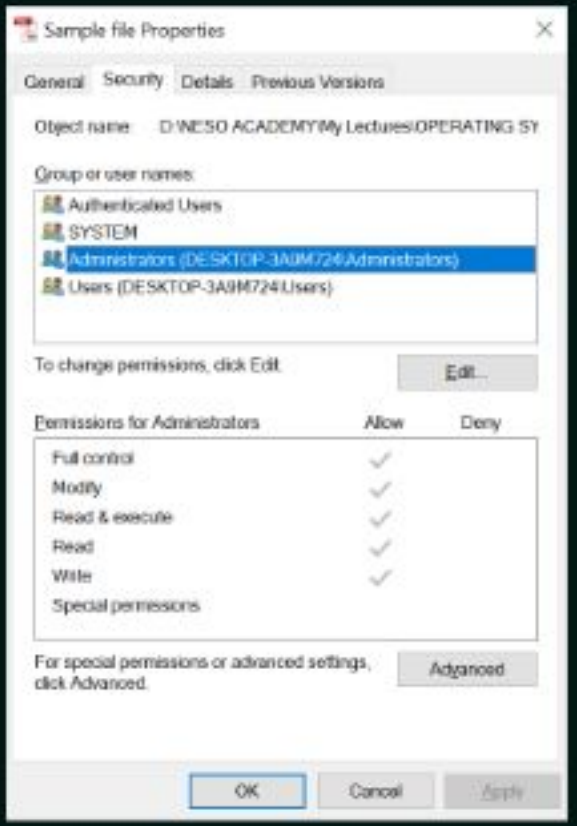
A sample directory listing

Example

In a Windows System







Example

In a Windows System