What is **state**?

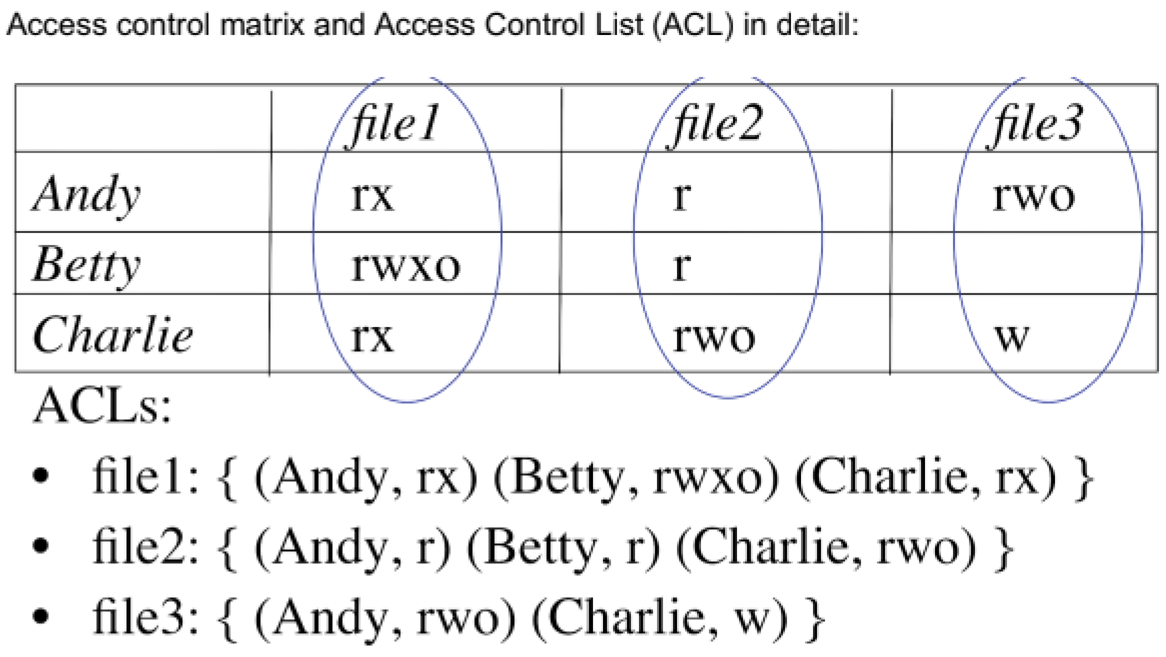
A state refers to the current condition of a system, which includes the present values of everything stored in memory, all data saved on secondary storage (like hard drives), and other system components.

What is **protection state**?

A protection state is a part of the overall system state that specifically focuses on aspects related to security and protection. It includes information related to how access to resources is controlled and protected.

what is the concept of **security policy** and **security mechanism**?

Security Policy: A security policy defines the rules and guidelines that specify what is allowed and what is not allowed in a system. It dictates how resources should be protected and who can access them.
Security Mechanism: A security mechanism is the method or tool used to enforce the security policy. It ensures that the rules defined by the policy are followed and that unauthorized access is prevented.



Access control matrix and Access Control List (ACL) in detail:

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

ACLs:

- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }

The set of rights is {read, write, execute, append, own, copy}

Own and copy rights are special rights.

What is **copy right**?

Copy right, also known as the grant right, is the ability of a person who holds certain rights to give or transfer those rights to someone else.

What is **own right**?

The **own right** is a special privilege that allows the owner to manage their own access permissions. This means they can add or remove their own privileges. Additionally, it gives them the ability to grant rights to others. Typically, the owner of an object is the one who created it or someone to whom the original creator transferred ownership.

What's the difference between **copy right** and **own right**?

The difference between **copy right** and **own right** is:

- **Copy Right** (grant right) allows the holder to give specific rights to others, but it doesn't allow them to modify their own permissions.
- **Own Right** allows the holder to manage their own permissions (add or remove privileges for themselves) and also grant rights to others. It's more powerful since it includes self-management of rights.

What is **Principle of Attenuation of Privilege**?

The **Principle of Attenuation of Privilege** states that a subject cannot grant rights to others that they themselves do not have. In other words, you can only pass on the rights that you personally possess.

Augmented/Extended/Full-blown Access Control List

In the traditional UNIX ACL model, Anne faces a challenge with granularity because she must grant permissions at a broad level rather than precisely as needed. For example:

- To allow Beth to read the file, she must assign a read permission.
- To allow Caroline to write to the file, she must assign a write permission.
- To allow Della to both read and write, she must assign both read and write permissions.

However, if there are overlapping or more complex permissions needed, Anne might end up giving more permissions than desired or fewer than needed due to the lack of granularity in the traditional ACL approach. This can lead to over-permissioning or under-permissioning of users.

Solution?

To address the limitations of traditional UNIX ACLs, we need an enhanced or extended access control mechanism. This improved system offers more precise and flexible control over permissions.

One example is IBM's AIX operating system, which augments traditional UNIX ACLs with additional features to provide more granular and effective access control. This extended ACL model allows for finer control over permissions and better management of access rights, addressing some of the issues associated with the basic UNIX ACL model.

to understand the three extended permissions: **permit**, **deny**, and **specify**.

attributes:

base permissions:

owner(frank): rw-

group(system): r-x

others: ---

extended permissions:

enabled

permit rw- u:dhs

deny r-- u:chas, g:system

specify r-- u:john, g:gateway, g:mail

permit rw- g:account, g:finance

1. **Permit**: Grants a user or group the specified access rights to the file. For example, `permit rw- u:dhs` allows the user `dhs` to read and write to the file.
2. **Deny**: Restricts a user or group from accessing the file with the specified rights. For instance, `deny r-- u:chas, g:system` prevents the user `chas` and the group `system` from reading the file.
3. **Specify**: Precisely defines the file access for a user or group. For example, `specify r-- u:john, g:gateway, g:mail` sets read-only access for the user `john` and the groups `gateway` and `mail`.

The `enabled` keyword must be used to activate these extended permissions in the ACL. By default, these permissions are not active (`disabled`).

If a user is denied a particular access by either a **deny** or a **specify** keyword, no other entry can override that access denial.

Deny>Specify>Permit
Always remember: restrictive entries take precedence over permissive modes.

Which subjects can modify an object's ACL?

- The owner of the object can modify its ACL.

Do the ACLs apply to a privileged user?

- ACLs are applied in a limited fashion to privileged users, such as the root superuser on UNIX systems and the administrator on Windows systems.

Does the ACL support wildcards (e.g., "everybody but user Fran")?

- No, ACLs do not support wildcards.

How is a conflict handled?

- In most cases, denial of access takes precedence over permission.