Infrastructure Security Using Linux

Computer science / Cybersecurity

The Process Control and Filesystem

4 - 5

Process Control

Process Structure (Components)

- **Address Space:**
  - Set of memory pages (usually **4KiB or 8KiB**).
  - Holds **code, data, and stack**.
- **Kernel Data Structures:**
  - **Address space map**
  - **Current status** (running, sleeping, etc.)
  - **Priority** and **scheduling parameters**
  - **Resource usage** (CPU, memory)
  - **Open files and network ports**
  - **Signal mask** (blocked signals)
  - **Owner** (user ID of process creator)

Components of a Process

- **Thread**
  - **Execution context** within a process.
  - Multiple threads in a process **share the same address space** and resources.
  - Enables **parallelism** within a process.
  - Called **lightweight processes** → **Cheaper** to create and destroy than full processes.

The PID: process ID number

- **Process Identifiers**
  - **PID (Process ID):**
    - Unique integer assigned to each process at creation.
    - Used in system calls (e.g., sending signals).
  - **PPID (Parent Process ID):**
    - PID of the process that created it.
  - **UID and EUID:**
    - **UID** = User ID of the user who started the process.
    - **EUID** = Effective User ID (determines access rights).

Lifecycle of a Process

Process Creation

- **fork** → Creates a copy of the original process:
  - New process = **distinct PID** and separate accounting info.
  - Copy is largely **identical** to the parent.
- **clone** → Superset of **fork**:
  - Handles **threads** and adds extra features.
  - **fork** internally calls **clone** for backward compatibility.

Lifecycle of a Process

- **Basic Process**
  - At boot, the kernel creates several autonomous processes.
  - **init** or **systemd** = **Process ID 1** (first user-space process).
  - Executes **startup scripts**.
  - All other user-space processes are **descendants** of this process.

Signals

- The signals **KILL**, **INT**, **TERM**, **HUP**, and **QUIT** all sound as if they mean approximately the same thing, but their uses are actually quite different.

```
kill [-signal] pid

killall firefox

pkill -u yazan          # kill all processes owned by user yazan
```

PS: Monitoring Processes

- The **ps** command is the **system administrator's main tool for monitoring processes**. and what its current status is (running, stopped, sleeping, and so on).

```
$ ps aux | head -8
$ ps lax | head -8
```

To look for a specific process, you can use **grep** to filter the output of **ps**.

```
$ ps aux | grep -v grep | grep firefox
```

We can determine the PID of a process by using **pgrep**.

```
$ pgrep firefox
```

ps lax | head -8

Interactive monitoring with **top**

top vs htop

- **top** – Real-time, dynamic system monitoring:
  - Displays **system summary** and **process list**.
  - **User-configurable** display (can be persistent).
  - Updates every **1–2 seconds**.
- **htop** – Enhanced version of **top**:
  - Interactive, supports **scrolling** (vertical + horizontal).
  - Displays **full command lines**.
  - Better **UI** and **more options** for managing processes.

Nice and renice: changing process priority

Niceness and Priority

- **Niceness** = Hint to the kernel about how to schedule a process relative to others.

Commands:

- **nice** – Start a process with a specific niceness:

```
nice -n 10 sh infinite.sh &
```

- **renice** – Change niceness of a running process:

```
renice -n 10 -p 1234
```

Priority Ranges:

- **0 to 99** → **Real-time** (higher priority).
- **100 to 139** → **User-space processes** (lower priority).

The /proc filesystem

- The Linux versions of **ps** and **top** read their process status information from the **/proc** directory, a pseudo-filesystem in which the kernel exposes a variety of interesting information about the system's state.

Table 4.4: Process information files in Linux /proc (numbered subdirectories)

File	Contents
cgroup	The control groups to which the process belongs
cmd	Command or program the process is executing
cmdline[a]	Complete command line of the process (null-separated)
cwd	Symbolic link to the process's current directory
environ	The process's environment variables (null-separated)
exe	Symbolic link to the file being executed
fd	Subdirectory containing links for each open file descriptor
fdinfo	Subdirectory containing further info for each open file descriptor
maps	Memory mapping information (shared segments, libraries, etc.)
ns	Subdirectory with links to each namespace used by the process.
root	Symbolic link to the process's root directory (set with **chroot**)
stat	General process status information (best decoded with **ps**)
statm	Memory usage information

a. Might be unavailable if the process is swapped out of memory

Strace and truss

- To figure out what a process is doing, you can use **strace** on Linux or **truss** on FreeBSD. These commands trace system calls and signals. They can be used to debug a program or to understand what a program is doing.

```
strace -p 5810
```

- **top** starts by checking the current time. It then opens and stats the **/proc** directory, and reads the **/proc/1/stat** file to get information about the **init** process.

df -h and **du** commands

- You can run a **df -h** to check the filesystem usage. If the filesystem is full, you can use the **du** command to find the largest files and directories.
- You can also use the **lsof** command to find out which files are open by the runaway process.

```
lsof -p pid
```

Periodic processes

- **cron: schedule command:** The **cron** (**crond** on RedHat: yeah weirddos!!) daemon is the traditional tool for running commands on a predtermined schedule. It starts when the system boots and runs as long as the system is up.
- **cron** reads configuration files containing lists of command lines and times at which they are to be invoked. The command lines are executed by **sh**
- Crontabs for individual users are stored under **/var/spool/cron** (Linux) or **/var/cron/tabs** (FreeBSD).

Examples

- The following schedule: 0,30 * 13 * 5 means that the command will be executed at 0 and 30 minutes past the 13th hour on Friday. If you want to run a command every 30 minutes, you can use the following schedule: */30 * * * *.

```
# Run a command at 2:30am every day
30 2 * * * command
# Run a command at 10:30pm on the 1st of every month
30 22 1 * * command
# Run a Python script every 1st of the month at 2:30am
30 2 1 * * /usr/bin/python3 /path/to/script.py
```

crontab management

crontab (Manage Cron Jobs)

- **Create/Modify/Delete crontabs**.

Common Options:

- **-e** → Edit crontab.
- **-l** → List crontab entries.
- **-r** → Remove crontab.

Systemd timer

systemd Timer

- **Unit file** ending in .timer (alternative to cron).
- More **flexible** and **powerful** than cron jobs.

Commands:

- **Manage timers** with systemctl:

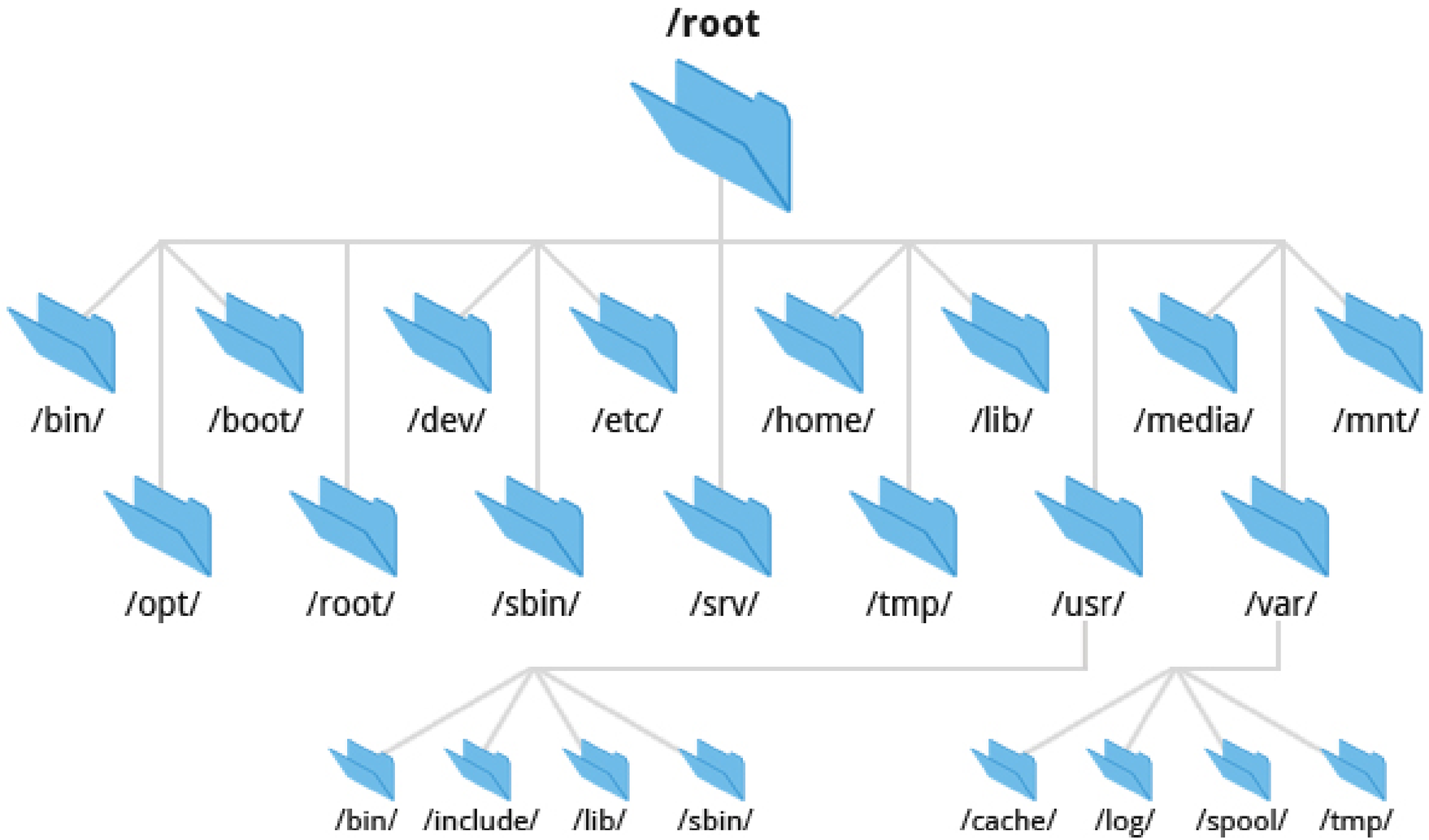
```
$ systemctl list-timers
```

Common Scheduled Tasks

- **Sending email** – Automate with cron/systemd timers.
- **Filesystem cleanup** – Use **cron** or **systemd timers** for automated cleanup.
- **Log rotation** – Split logs by size/date and keep older versions available.
- **Batch jobs** – Schedule long-running calculations as batch tasks.
- **Backup and mirroring** – Use **rsync** to sync files to a remote system.

The Filesystem

The Filesystem



The Filesystem

- **Filesystem Components**
  1. **Namespace** – Organizes files in a hierarchical structure.
  2. **API** – System calls for file navigation and manipulation.
  3. **Security models** – Control access, visibility, and sharing.
  4. **Implementation** – Software that links logical model to hardware.
- **Common Disk-Based Filesystems: ext4** (and ext2, ext3), **XFS, UFS, ZFS** (Oracle), **Btrfs**
- **Other Filesystems: VxFS** (Veritas), **JFS** (IBM), **FAT**, **exFAT**, **NTFS** (Windows), **ISO 9660** (CD/DVD)

Pathnames

Directory vs Folder

- **Directory** = Technical term (use in technical contexts).
- **Folder** = Informal term (avoid in technical writing).

Pathnames:

- **Absolute** → Full path from the root (/) (e.g., /home/username/file.txt).
- **Relative** → Path from the current location (./file.txt).

Filesystem Mounting and Unmounting

File Tree vs Filesystem

- **File tree** → Overall system layout.
- **Filesystem** → Individual branches attached to the file tree.

Mounting:

- mount command → Attaches a filesystem to a **mount point** (directory in the file tree).
- Maps the directory to the **root of the new filesystem**.

```
# Mount the filesystem on /dev/sda4 to /users
$ mount /dev/sda4 /users
```

Filesystem Mounting and Unmounting

Unmounting Filesystems

- **umount -l → Lazy unmount**:
  - Removes the filesystem from the namespace immediately.
  - Fully unmounts it once no processes are using it.
- **umount -f → Forceful unmount**:
  - Used for busy filesystems (can be risky).

Safer Alternative:

- Use **lsof** or **fuser** to identify processes using the filesystem, then terminate them before unmounting.

Organization of the file tree

The root filesystem includes at least the root directory and a minimal set of files and subdirectories. The file that contains the OS kernel usually lives under **/boot**, but its exact name and location can vary. Under BSD and some other UNIX systems, the kernel is not really a single file so much as a set of components.

/etc contains critical system and configuration files. **/sbin** and **/bin** for important utilities, and sometimes **/tmp** for temporary files. The **/dev** was traditionally part of the root filesystem, but these days it's a virtual filesystem that's mounted separately.

Some systems keep shared library files and a few other oddments, such as the C preprocessor, in the **/lib** or **/lib64** directory. Others have moved these items into **/usr/lib**, sometimes leaving **/lib** as a symbolic link.

Pathname	Contents
/bin	Core operating system commands
/boot	Boot loader, kernel, and files needed by the kernel
/compat	On FreeBSD, files and libraries for Linux binary compatibility
/dev	Device entries for disks, printers, pseudo-terminals, etc.
/etc	Critical startup and configuration files
/home	Default home directories for users
/lib	Libraries, shared libraries, and commands used by **/bin** and **/sbin**
/media	Mount points for filesystems on removable media
/mnt	Temporary mount points, mounts for removable media
/opt	Optional software packages (rarely used, for compatibility)
/proc	Information about all running processes
/root	Home directory of the superuser (sometimes just /)
/run	Rendezvous points for running programs (PIDs, sockets, etc.)
/sbin	Core operating system commands[a]
/srv	Files held for distribution through web or other servers
/sys	A plethora of different kernel interfaces (Linux)
/tmp	Temporary files that may disappear between reboots
/usr	Hierarchy of secondary files and commands
/usr/bin	Most commands and executable files
/usr/include	Header files for compiling C programs
/usr/lib	Libraries; also, support files for standard programs
/usr/local	Local software or configuration data; mirrors **/usr**
/usr/sbin	Less essential commands for administration and repair
/usr/share	Items that might be common to multiple systems
/usr/share/man	On-line manual pages
/usr/src	Source code for nonlocal software (not widely used)
/usr/tmp	More temporary space (preserved between reboots)
/var	System-specific data and a few configuration files
/var/adm	Varies: logs, setup records, strange administrative bits
/var/log	System log files
/var/run	Same function as **/run**; now often a symlink
/var/spool	Spooling (that is, storage) directories for printers, mail, etc.
/var/tmp	More temporary space (preserved between reboots)

a. The distinguishing characteristic of **/sbin** was originally that its contents were statically linked and so had fewer dependencies on other parts of the system. These days, all binaries are dynamically linked and there is no real difference between **/bin** and **/sbin**.

File types

- Most filesystem implementations define seven types of files:

1. Regular files
2. Directories
3. Character devices files
4. Block devices files
5. Local domain sockets
6. Named pipes(FIFOs)
7. Symbolic links

You can determine the type of a file by using the **file** command (type **man file** for more information).

```
$ file /bin/bash

bin/bash: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=33a5554034feb2af38e8c75872058883b2988bc5, for GNU/Linux 3.2.0, stripped
```

- You can determine the type of a file by using the **file** command (type **man file** for more information).

File types

- You can also use **ls -ld**, the **-d** flag forces **ls** to show the information for a directory rather than showing the directory's contents.
- **Regular files** consist of a series of bytes; filesystems impose no structure on their contents. Text files, data files, executable programs, and shared libraries are all stored as regular files.
- **Directories** are named references to other files.
- **Hard links** are a way to give a single file multiple names.
  The **ln** command creates a new hard link to an existing file. The **-i** option to **ls** causes it to display the number of hard links to each file.

File type	Symbol	Created by	Removed by
Regular file	–	editors, **cp**, etc.	**rm**
Directory	d	**mkdir**	**rmdir**, **rm -r**
Character device file	c	**mknod**	**rm**
Block device file	b	**mknod**	**rm**
Local domain socket	s	**socket** system call	**rm**
Named pipe	p	**mknod**	**rm**
Symbolic link	l	**ln -s**	**rm**

```
$ ln /etc/passwd /tmp/passwd
```

File types

- **Device Files**
  - Enable hardware and peripheral communication through drivers.
  - **Character Devices:** Stream-based data handling (e.g., keyboard).
  - **Block Devices:** Block-based data handling (e.g., disk).
- **Local Domain Sockets**
  - Confined to the local host, not network-wide.
  - Often used by syslog, X Window System for IPC.
- **Named Pipes**
  - IPC mechanism similar to local sockets.
  - Operates solely within the same host.
- **Symbolic Links (Soft Links)**
  - Reference files by name instead of inode.
  - Can link across filesystems and to directories.
  - Example: /usr/bin → /bin helps keep root filesystem small.

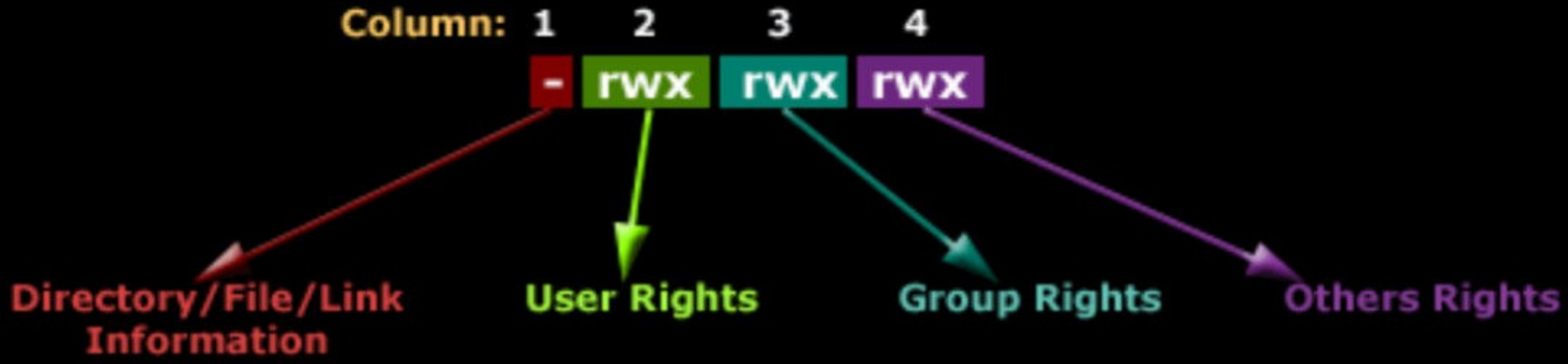
```
$ ln -s /bin /usr/bin

$ ls -l /usr/bin
lrwxrwxrwx 1 root root 4 Mar  1  2020 /usr/bin -> /bin
```

File attributes (File Permission Bits)

- **Unix/Linux Filesystem Model:**
  - Each file has nine permission bits controlling read, write, and execute permissions.
  - An additional three bits affect executable program operations.
  - These together form the file's mode.
- **Permission Bit Groups:**
  - Divided into three sets of three bits each:
    - **Owner (u)**: Controls what the file's owner can do.
    - **Group (g)**: Specifies permissions for members of the file's group.
    - **Others (o)**: Defines access for everyone else.
  - Use the acronym **Hugo** to remember the order: **u (owner), g (group), o (others)**.

Understanding The Linux File Permissions



While the first column defines a directory, file or link, the next 3 columns (2, 3, 4) defi... the permissions for the User, Group and Others (everyone else) groups.

chmod: change permissions

- The **chmod** command changes the mode of a file. You can use the octal notation or the symbolic notation.

Specifier	Meaning
u+w	Add write permission for the file's owner
ug=rw,o=r	Gives r/w permission to owner and group, and r permission to others
a-x	Remove execute permission for all users
ug=srx, o=	Set the setuid, setgid, and sticky bits for owner and group (r/x)
g=u	Make the group's permissions the same as the owner's

Table 5.3: Permission encoding for chmod

Octal	Binary	Perms	Octal	Binary	Perms
0	000	---	4	100	r--
1	001	--x	5	101	r-x
2	010	-w-	6	110	rw-
3	011	-wx	7	111	rwx

- **Tips**: You can also specify the modes to be assigned by copying the mode from another file with the **--reference** option. (e.g. **chmod --reference=sourcefile targetfile**)

chmod: change permissions

- **chown: change ownership**
  - The **chown** command changes the owner and group of a file. The **-R** option causes **chown** to change the ownership of the file's contents recursively.

```
$ chown -R abdou:users /home/abdou
```

- **chgrp: change group**
  - The **chgrp** command changes the group of a file. The **-R** option causes **chgrp** to change the group of the file's contents recursively.

```
$ chgrp -R users /home/abdou
```

umask: set default permissions

- The **umask** command sets the default permissions for new files and directories. The **umask** command is a bit mask that is subtracted from the default permissions to determine the actual permissions.
- Exemple:

```
$ umask 022
```

Octal	Binary	Perms	Octal	Binary	Perms
0	000	rwx	4	100	-wx
1	001	rw-	5	101	-w-
2	010	r-x	6	110	--x
3	011	r--	7	111	---

- For example, **umask 027** allows the rwx to owner, rx to group, and no permissions to others.

Access Control Lists

- **Traditional Unix Permissions:**
  - **Strengths:**
    - Straightforward and reliable.
  - **Limitations:**
    - Files typically have only one owner.
    - Groups are limited to a single set of permissions per file.
- **Access Control Lists (ACLs):**
  - **Purpose:**
    - Extend Unix permissions to support more granular access control.
  - **Benefits:**
    - Allow multiple owners per file.
    - Enable varied permissions for different user groups.
    - Provide flexibility that traditional models lack.

Access Control Lists

- **Components:**
  - **Specifier:** Identifies a user, group, or keyword (e.g., *owner* or *other*).
  - **Permission Mask:** Defines the access rights (read, write, execute).
  - **Type:** Specifies whether to *allow* or *deny* access.
- **Command Utilities:**
  - **getfacl:** Displays the current ACL of a file.
  - **setfacl:** Configures or modifies the ACL of a file.

```
$ getfacl /etc/passwd

$ setfacl -m u:TTU:rw /etc/passwd
```

Implementation of ACLs

- There are two types of ACLs: **POSIX ACLs** and **NFSv4 ACLs**. POSIX ACLs are the traditional Unix ACLs, and NFSv4 ACLs are a newer, more powerful type of ACL.
- **POSIX ACLs**
  - POSIX ACLs are the traditional Unix ACLs. They are supported by most Unix-like operating systems, including Linux, FreeBSD, and Solaris.

Format	Example	Sets permissions for
user::perms	user:rw-	The file's owner
user:username:perms	user:abdou:rw-	The user named username
group::perms	group:r-x	The file's group
group:groupname:perms	group:users:r-x	The group named groupname
mask::perms	mask::rwx	The maximum permissions
other::perms	other::r--	Everyone else

Implementation of ACLs

- **NFSv4 ACLs**
  - NFSv4 ACLs are a newer, more powerful type of ACL. They are supported by some Unix-like operating systems, including Linux and FreeBSD.
  - NFSv4 ACLs are similar to POSIX ACLs, but they have some additional features. For example, NFSv4 ACLs have a **default ACL** that is used to set the ACL of new files and directories.
- NFSv4 file permissions ….. **(Google it … !)**