Lecure 2. Symbolic and hard links. Permissions

root - the superuser

/sys is the virtual filesystem that displays some kernal confiuration

everything is a file so every kernal setting is also a file

regular files,
directories,
character devices (/dev/null)
block devices (/dev/sda)

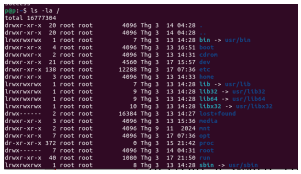
symbolic link

to create a symblic link file use

```
ln -s file1.txt file6.txt
```

file6.txt container address to file1.txt

#symbolic link is qite simple and only store path to the linked file



bin is symbolic link to file usr/bin

lib -> usr/lib

lib64 -> usr/lib64

to read the symbolic link of a file we can use following statement

readlink file6.txt

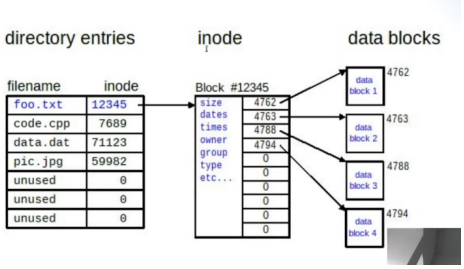
output: /home/p/os/file6.txt

$(pwd) : store the dir to the cur path such as $(pwd)/file1.txt

hardlink

filesystem = inodes + blocks

blocks store data
inodes file store metadata + pointer to data (but without filenames)



hardlink: contain make file point to the same inodes id

file permission

```
 stat file.txt
```

```
          File: a.cpp
          Size: 1440            Blocks: 8          IO Block: 4096   regular file
        Device: 10302h/66306d   Inode: 8388643     Links: 1
        Access: (0664/-rw-rw-r--)  Uid: ( 1000/       p)   Gid: ( 1000/       p)
        Access: 2025-03-17 07:16:09.667860730 +0700
        Modify: 2025-03-17 07:16:09.647860595 +0700
        Change: 2025-03-17 07:16:09.647860595 +0700
         Birth: 2025-03-17 07:04:42.055239147 +0700
```

Access: specifies what kind of access we are allowed

Uid: specifies the owner of the file

Gid: specifies the owner group of the file. So if we see for example stat bin/bash

Uid: user = p, Gid = p

```
p@p:~/cpp$ stat /bin/bash
  File: /bin/bash
  Size: 1396520         Blocks: 2728       IO Block: 4096   regular file
Device: 10302h/66306d   Inode: 3145823     Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2025-03-17 16:22:36.508302034 +0700
Modify: 2024-03-14 18:31:47.000000000 +0700
Change: 2025-03-13 14:28:12.109753666 +0700
 Birth: 2025-03-13 14:28:12.104755850 +0700
```

There are 3 types of permission: read, write, execute

the order of access folloing this: user/group/other

```
-rwxr-xr-x

- rwx: he can read write execute
  r-x: he can read and execute
  r-x: he can read and execute but not write the file
```

chmod, change file mode

```
chmod +x file.txt
then file will be added permission to execute

chmod -x file.txt
then file will be remove permission to execute
```

if we want to executable only for current user

user - u, group -g, other -o

```
chmod go-x file1.txt
```

+x will make your non-executable files executable - this is not what you want
+X will just allow to view files and directorires

ma chmod

-r will remove rea permissions
-R recursive

chown: change the owner of the file

chmod [user]:[group] [file]

sticky bit mean that only the file/directory owner can remove the file even you have write permisison

chmod -t

setgid

Lecture 3. Setuid and Setgid Bits. Processes

setuid bit - changes euid to the owner of the file(instead of the user who invoked it)

can we change uid, the answer is yes? but it's only if we are root

why are all these setuid and setgid bits needed

escalate: to become or make something become greater or more serious

/ˈes.kə.leɪt/

sudo has owner root and setuid bit, invoke sode,
-> euid is zero, sudo can call setuid() to change user

sudo checks permissions and asks for password if necessary

if success -> sudo just invokes your programs

setuid bit does work only on executabe programs and not on scripts

why?
./uids/py -> kernel read file metadata, check permissions, and executes/usr/bin/python3/uids.py

when /usr/bin/python3/ uids.py is executed, then python3 reads uids.py file again

suppose setuid works on sciprts, What happens?
1) we have setuid uids.py and invoke it
2) kernel reads our uids.py and invokes /usr/bin/python3 uids.py
3) kernel runs this program as setuid (as root)
4) we move uids.py (can do it if we have permissions on the directory) and replace it with our own contents
5) /usr/bin/python3 reads our uids.py (but now it's the wrong file)

thus we can execute any code as root
we dont often have write permissions on system directories
hardlinks

when you write setuid programs, you have to be really extra careful

when writing setuid programs hyou also need to be careful wht do the libraryies that you use do

when we execute the program -> **create a process**

has some computation resources (CPU, memory, etc)

then every process has exit code - integer show how the process exited

echo $? # return lasted exit code

&& invokes the next command only if the previous one succes

|| i invokes the next commnads only if the previous one failed

every process has its ID

ls /proc/

every process has command line

every process has its parent process id, its user, and group id(and effective user and group i

ps: process stat

pidof [name] : show the pid of process name

you can send signals to processes

nignal numbers -just integers
man sign