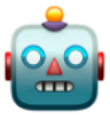


Sistemi Operativi 2

Breve storia di Linux

Il sistema operativo **Multics (Multiplexed Information and Computing Service)** fu uno dei primi sistemi operativi a condivisione di tempo (time-sharing), ossia multiprocesso e multi-utente, sviluppato attivamente a partire dal 1964 da parte dei centri di ricerca delle compagnie Bell Labs (AT&T Corp., una compagnia telefonica americana) e General Electric, assieme all'università MIT.
Multics mise sul campo tutta una serie di concetti e tecniche costruttive che sono ancora oggi elementi essenziali dei moderni sistemi operativi. Sebben rivoluzionario, il progetto Multics fu presto abbandonato da Bell Labs, poiché ritenuto troppo complesso da gestire.
A seguito di ciò, Ken Thompson e Dennis Ritche, due ricercatori di Bell Labs, svilupparono tramite un microcomputer PDP-7 la prima versione di Unics (Uniplexed Information and Computing Service), scritta totalmente in Assembly. Successivamente, sotto proposta di Brian Kernighan, il nome di Unics venne cambiato definitivamente in Unix.
Il sistema operativo Unix si diffuse rapidamente nei successivi 3-5 anni, portandolo allo sviluppo di versioni scritte tramite il linguaggio B e successivamente (e definitivamente) tramite il linguaggio C. Le versioni scritte in tali linguaggio permisero di portare Unix su varie architetture.
Il linguaggio C fu sviluppato da Dennis Ritchie stesso al fine di migliorare il linguaggio B precedentemente sviluppato dal suo collega Ken Thompson. La miglioria principale rispetto al B consiste nell'aggiunta dei tipi di dato (int, float, char, ...) rispetto alle sole generiche word da 4 byte del linguaggio B.
Successivamente, il codice sorgente del sistema operativo Unix venne distribuito ad università e centri di ricerca interamente assieme al proprio codice sorgente, il quale venne anche venduto ad aziende private, portando alla nascita di molte versioni.
Negli anni 80', Richard Stallman sviluppò il sistema operativo GNU (GNU is Not

Unix acronimo ricorsivo), basato su Unix ma diverso da esso in quanto non contenente codice del sistema operativo Unix, e inventò GPL (GNU General Public Licence), una licenza pubblica utilizzata per il software libero. Unix venne inoltre riscritto completamente, aggiungendo pacchetti importanti ad esso, molti presi direttamente da GNU (es: gcc, make, ...).
Negli anni 90', Linus Torvalds sviluppa il kernel Linux, il quale verrà poi utilizzato da altri sistemi operativi basati su Unix o derivati da esso. In particolare, nel 1994 viene definito lo standard Unix, dove un sistema operativo può avere marchio UNIX solo se esso rispetta le SUS (Single Unix Specification) e paga le royalties per l'uso del marchio.

Le caratteristiche dei sistemi Unix

Le caratteristiche di un moderno sistema operativo Unix, indipendentemente dalla sua categoria, sono:

- Multi-utente e multi-processo
- File system gerarchico
- Kernel in grafo di gestire la memoria principale, la memoria secondaria, i processi, le operazioni I/O e le risorse hardware in generale
- System call utilizzabili tramite funzioni C che possono essere chiamate per interfacciarsi con il kernel
- Possiedono una shell di sistema, ossia un programma che "esegue programmi" interpretando i comandi dell'utente
- Modularità, programmi di utilità e supporto ad ambienti di programmazione
- Composto da una serie di piccoli programmi che eseguono un compito specifico, limitato, ma in maniera esatta e semplice
- I programmi sono silenziosi, il loro output è minimale e ridotto a ciò che è stato esplicitamente richiesto
- Ogni lavoro complesso può essere svolto come articolazione del lavoro svolto da programmi semplici
- I programmi manipolano solo testo e mai i file binari (es: altri programmi)

Utilizzo della Shell di sistema

Shell di Sistema

Informalmente, una **shell di sistema** (spesso detta terminale) è un programma che "esegue programmi".
Più formalmente, invece, la shell è un programma interattivo e/o **batch** (ossia "a lotti") che accetta comandi da far eseguire al **kernel.**
Tali comandi non sono necessariamente dei programmi, bensì possono essere anche dei comandi definiti all'interno della shell stessa.

Prima di eseguire un comando, la shell stampa a video un prompt, ossia una stringa nel formato

`[nome_utente@nome_macchina cwd]$`

Ogni comando segue la seguente struttura:

`nome_comando [argomenti_opzionali] argomenti_obbligatori`

Ad esempio, nel comando `cp -r -i -a -u` file_sorgente file_destinazione, gli argomenti `-r, -i, -a, -u` sono opzionali, mentre i rimanenti sono obbligatori. Tipicamente, gli argomenti opzionali possono essere utilizzati anche con sintassi alternative (es: per il comando cp gli argomenti –interactive, –recursive sono uguali agli argomenti -i, -r). Inoltre, eventualmente essi possono avere un valore aggiuntivo in input (es: l'argomento –key=1 assegna il valore 1 all'argomento –key) e possono essere raggruppati (es: cp -ri è equivalente a cp -r -i).

Tutti i comandi lanciati nella shell vengono salvati in una **cronologia**.

Utilizzando le freccette su e giù della tastiera, è possibile scorrere i comandi presenti nella cronologia.

Super utente (root)

Ogni sistema operativo Linux-based possiede un **super utente** detto **root**, il quale possiede tutti i privilegi di sistema.
Pertanto, tale utente possiede accesso ad ogni operazione o comando possibile all'interno del sistema stesso.
L'utente root possiede sempre
UID pari a 0.

File System

Col termine **file system** si intende una struttura dati atta all'organizzazione di un'area di memoria di massa basata sul concetto di **file** e di **directory**, dove quest'ultime
possono contenere al loro interno dei file ed altre directory, creando così una struttura
gerarchica ad albero, dove solo le directory possono avere figli e i file corrispondono alle foglie dell'albero.

Come già espresso nel capitolo precedente, nei sistemi operativi Linux-based ogni cosa può essere rappresentata come un file o un processo. Per tanto, all'interno del file system è necessario distinguere tra file regolari, i quali contengono sequenze di bit dell'area di memoria sulla quale è installato il file system, e file non regolari, ad esempio utilizzati per l'accesso di basso livello a periferiche o dispositivi vari.
Inoltre, all'interno di una directory valgono le seguenti regole:

- Non possono esistere due file o due sotto-directory con lo stesso nome
- Non possono esistere un file ed una sotto-directory con lo stesso nome
- I nomi dei file e delle sotto-directory sono **case sensitive** (es: ciao.txt è diverso da Ciao.txt)

Nei sistemi operativi Unix-based e Linux-based vi è un solo file system principale avente una directory radice (root directory), ossia la directory /. Essendo la radice del file system, ogni altro file o directory è contenuto direttamente o indirettamente all'interno della root directory.

I Percorsi

Pertanto, ogni file o directory è raggiungibile dalla root directory mediante un percorso (path), il quale può essere di due tipologie:

- **Percorso assoluto**, ossia una sequenza di directory separate da uno / che specifica la posizione di un file o una directory a partire dalla root directory

  `(es: /home/utente/dir/subdir/file.pdf)`

- **Percorso relativo**, ossia una sequenza di directory separate da uno / che specifica la posizione di un file o una directory a partire dalla current working directory (cwd), ossia la cartella attualmente "aperta"
  (es: se la cwd è
  `/home/utente` allora il percorso relativo `dir/subdir/file.pdf` è equivalente al percorso assoluto `/home/utente/dir/subdir/file.pdf`)

Ogni **percorso relativo** risulta essere valido solo se la **cwd attuale è corretta**, mentre ogni **percorso assoluto** risulta essere **valido indipendentemente dalla cwd.**

All'interno dei percorsi (sia assoluti che relativi) è possibile utilizzare due directory speciali presenti all'interno di ogni directory:

- La **current directory** (ossia .), corrispondente alla directory stessa in cui ci si trova
  (es: il percorso
  `~/dir/./ciao.txt` è equivalente a `~/dir/ciao.txt`)
- La **parent directory** (ossia ..), corrispondente alla directory direttamente superiore a quella in cui ci si trova

(es: se la cwd attuale è `~/dir/subdir1/`, il percorso `../subdir2` è equivalente al percorso `~/dir/subdir2`)

Visualizzare i File

Per sapere la cwd attuale, è possibile utilizzare il comando `pwd`, mentre per cambiare cwd è possibile utilizzare il comando `cd [path]` (se l'argomento path viene omesso, la
cwd verrà impostata sulla home dell'utente, ossia ~/).
Per
visualizzare il contenuto di una directory, invece, è possibile utilizzare il comando `ls [path]`, restituente una lista dei file e le sotto-directory contenute in una directory (se l'argomento path viene omesso, viene utilizzata la cwd come path).

Se si vuole visualizzare **ricorsivamente** il contenuto della directory (dunque eseguendo ricorsivamente ls sulle sotto-directory), è possibile utilizzare l'argomento opzionale `-r`.

In una directory alcuni **file possono essere nascosti**, tipicamente file di configurazione o file usati come supporto a comandi ed applicazioni (es: il file .bash_history contiene la cronologia dei comandi eseguiti), i quali tuttavia non sono realmente invisibili, bensì essi vengono solamente omessi nell'output del comando ls, a meno che non venga utilizzato il parametro opzionale -a (dunque il comando `ls -a`). Un file può essere reso nascosto semplicemente aggiungendo un punto all'inizio del nome.

Creare Directory o File

Per **creare una directory**, è possibile utilizzare il comando **mkdir nome_dir**.

Utilizzando l'opzione -p, verranno create a catena tutte le sotto-directory del percorso indicato.
(es:
`mkdir -p dir1/dir2/dir3` crea anche le directory dir1 e dir1/dir2)

Per creare un
file vuoto, invece, è possibile utilizzare il comando `touch nome_file`, il quale potrà poi essere modificato utilizzando un editor di testo (come nano, vim, ...).
Per copiare un file, è possibile utilizzare il comando `cp [-r] [-i] [-u] src_file dest_file`, dove:

- L'opzione **[-r]** permette di effettuare una copia ricorsiva sulle directory
- L'opzione **[-i]** avvisa l'utente nel caso in cui il file di destinazione esista gia
- L'opzione **[-u]** effettua la sovrascrittura di un file già esistente solo se l'mtime del file sorgente è più recente di quello di destinazione

Spostare o Eliminare File

Per spostare (o rinominare) un file è possibile utilizzare il comando `mv [-i] [-u] [-f] src_file dest_file`, dove:

- Le opzioni **[-i]** e **[-u]** sono identiche a quelle del comando cp
- L'opzione **[-f]** forza l'operazione

Per eliminare un file è possibile utilizzare il comando `rm [-r] [-i] [-f] src_file dest_file`, dove:

- Le opzioni **[-r]** e **[-i]** sono identiche a quelle del comando cp
- L'opzione **[-f]** forza l'operazione è identica a quella del comando mv

A differenza del sistema operativo Windows, nei sistemi Linux-based non esiste il "cestino".
Per tanto, eseguendo il comando rm il file verrà completamente eliminato dal disco.

Per convertire o copiare file in modo avanzato, è possibile utilizzare il comando `dd [opt]` dove l'argomento [opt] è una sequenza di entrate nel formato variabile=valore.
Le variabili principali utilizzabili sono:

- **if**, ossia il file di input (se non specificato, l'input viene letto da tastiera)
- **of**, ossia il file di output (se non specificato, l'output viene scritto sul terminale)
- **bs**, ossia la dimensione di un singolo blocco in lettura/scrittura
- **count**, ossia il numero di blocchi da copiare
- **skip**, ossia il numero di blocchi da saltare nell'input prima di leggere effettivamente
- **seek**, ossia il numero di blocchi da saltare nell'output prima di scrivere effettivamente

Mounting, Partizioni e Tipi di file system

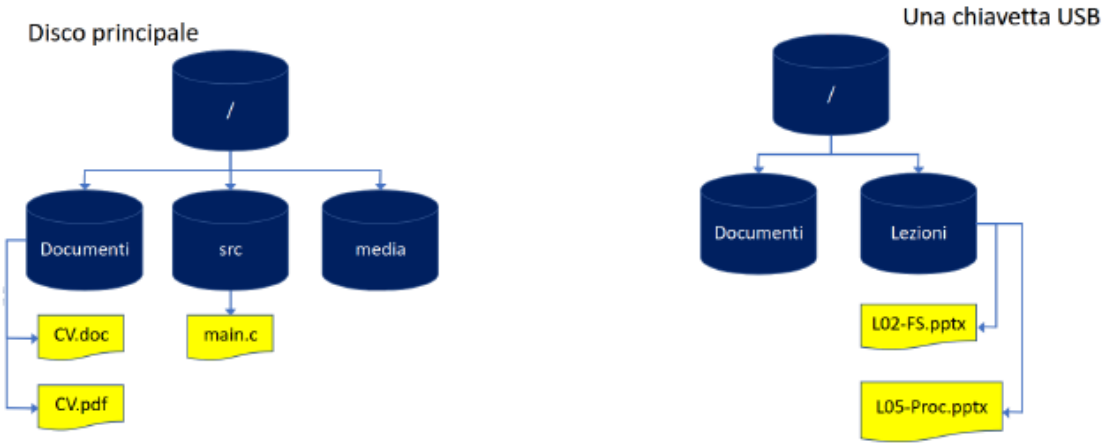
Il file system principale (ossia /) può contenere al suo interno **elementi** eterogenei tra loro, ad esempio:

- Dischi interni solidi o magnetici, solitamente contenenti il file system root
- File system su disco esterno
- File system di rete
- File system virtuali
- File system in memoria principale

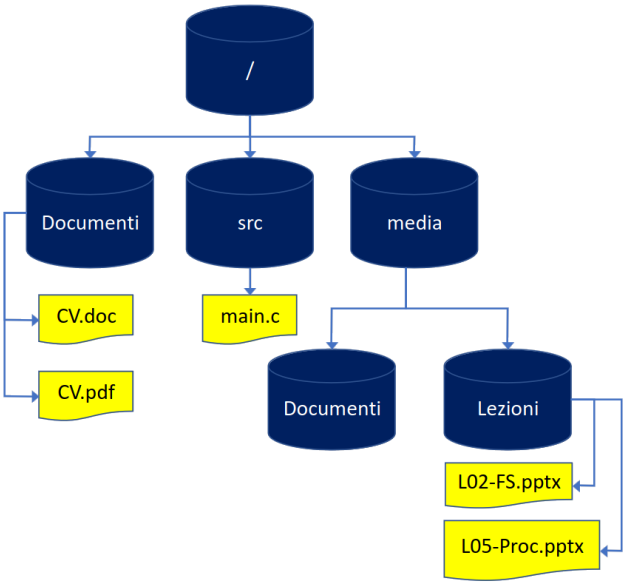
Una qualsiasi directory D all'interno del file system root può diventare il punto di mount per un altro file system F se e solo se la directory di root RF del file system F diventa accessibile da F.

Esempio:

- Consideriamo il seguente file system root e il seguente file system presente su una chiavetta USB



- Effettuando il mounting del secondo file system sulla directory `/mount` del file system root, esso diventa accessibile tramite la directory stessa



Effettuare il mounting di un file system F su una directory D non sovrascrive il contenuto della directory, bensì esso viene solamente temporaneamente sostituito.
Per tanto, se la directory D non è vuota, il suo contenuto originale sarà

nuovamente accessibile dopo
`l'unmount` di F

Per montare un file system e visualizzare i file system montati è possibile utilizzare il comando `mount` (consultare il manuale per le varie opzioni).
Inoltre, per visualizzare i file system attualmente montati è possibile esaminare anche il contenuto del file
`/etc/mtab`, mentre per visualizzare i file system montati all'avvio (bootstrap) è possibile esaminare il contenuto del file /etc/fstab.

Partizione

Un disco solido o magnetico può essere **suddiviso** in due o più partizioni, le quali possono essere gestite indipendentemente, come se fossero in realtà due dischi separati.

Tipi di file system

I tipi di file system Linux si differenziano in:

Nome	Dim_{max} Part.	Dim_{max} File	Lung_{max} Nome file	Journal
ext2	32 TB	2 TB	255 B	No
ext2	32 TB	2 TB	255 B	Si
ext4	1000 TB	16 TB	255 B	Sì
ReiserFS	16 TB	8 TB	4032 B	Sì

Dove un **journaling file system** tratta ogni scrittura su disco come transazione, tenendo traccia delle operazioni svolte su un file di log. Inoltre, ogni file system differisce per il modo in cui vengono codificati i dati al suo interno.

Per **formattare** un **disco o una partizione**, ossia creare su di essi un nuovo file system da zero, può essere utilizzato il comando `mkfs [-t type] device` (consultare il manuale).
Per
visualizzare la **dimensione e l'occupazione di un file system**, è possibile utilizzare il comando `df [-h] [-l] [-i] [file]` (consultare il manuale).

Directory di primo livello

Tipicamente, la directory **/**, contiene al suo interno le seguenti **directory di primo livello**, le quali possono essere montate o non:

- **/boot**, contenente il kernel e i file per il bootstrap. Non viene montata ed è per tanto salvata direttamente all'interno della root directory.
- **/bin**, contenente i file binari (ossia i file eseguibili) di base. Non viene montata ed è per tanto salvata direttamente all'interno della root directory.
- **/sbin**, contenente i file binari (ossia i file eseguibili) di sistema. Non viene montata ed è per tanto salvata direttamente all'interno della root directory.
- **/dev**, contenente i file non regolari relativi all'uso delle periferiche hardware e virtuali. Viene montata in fase di bootstrap.
- **/proc**, contenente i file relativi a dati e statistiche dei processi e ai parametri del kernel. Viene montata in fase di bootstrap.
- **/sys**, contenente i file relativi ad informazioni e statistiche dei dispositivi di sistema. Viene montata in fase di bootstrap.
- **/media** e **/mnt**, utilizzate come punto di mount per i dispositivi I/O (es: CD, DVD, USB, ...). Vengono montate solo quando necessario.
- **/etc**, contenente i file di configurazione di sistema. Non viene montata ed è per tanto salvata direttamente all'interno della root directory.
- **/var**, contenente i file variabili. Non viene montata ed è per tanto salvata direttamente all'interno della root directory.
- **/tmp**, contenente i file temporanei. Non viene montata ed è per tanto salvata direttamente all'interno della root directory.
- **/lib**, contenente le librerie necessarie ai file binari. Non viene montata ed è per tanto salvata direttamente all'interno della root directory.

In particolare, all'interno della directory **/etc** possiamo trovare due file di fondamentale importanza all'interno del sistema operativo:

- Il file `/etc/passwd`, contenente una lista di tutti gli utenti del sistema e informazioni ad essi associate. Ogni riga della lista possiede la seguente struttura:

  `username:password:uid:gid:gecos:homedir:shell`

- Il file `/etc/group`, contenente una lista di tutti i gruppi del sistema e informazioni ad essi associate. Ogni riga della lista possiede la seguente struttura:

  `groupname:password:gid:utente1,utente2,...`

  Anche in tal caso, il campo password viene censurato da una x.

Index Node (inode)

> 💡 All'interno di un file system Linux-based, ogni file (regolare e non, directory e non) è rappresentato da una struttura dati detta **index node (inode)**.

Tra i principali attributi di un inode troviamo:

- **inode number**, univoco per ogni inode
- **Type**, indicante il tipo di file
- **User ID (UID)**, ossia l'ID dell'utente proprietario del file
- **Group ID (GID)**, ossia l'ID del gruppo a cui è associato il file
- **Mode**, ossia i permessi di accesso al file per il proprietario, il gruppo associato ed ogni altro utente (vedi sezioni successive)
- **Size**, ossia la dimensione in byte del file
- **Timestamps**, ossia tre istanti di tempo:
  - **ctime** (change time), l'istante dell'ultima modifica di un attributo dell'inode
  - **mtime** (modification time), l'istante dell'ultima scrittura sul file associato

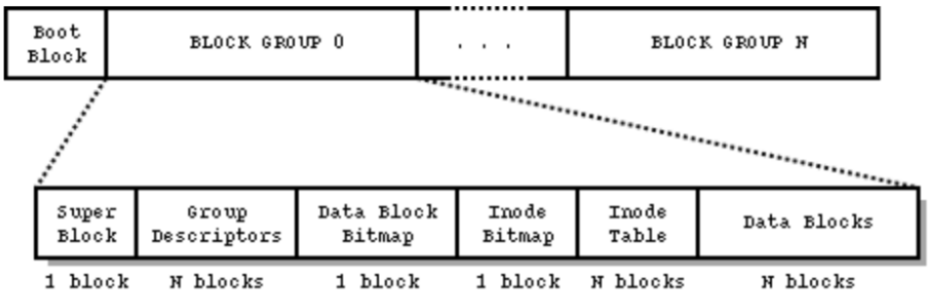
    - **atime** (access time), l'istante dell'ultima lettura del file associato
- **Link count**, ossia il numero di hard links dell'inode (vedi sezioni successive)
- **Data pointers**, ossia il puntatore alla lista dei blocchi su disco che compongono il file.

La vera funzionalità del comando `touch` risulta essere quella di **aggiornare tutti i timestamp** di un file **all'istante corrente**.

Se il file indicato non esiste, esso verrà **creato**. La creazione del file, dunque, è solo un **effetto secondario**.

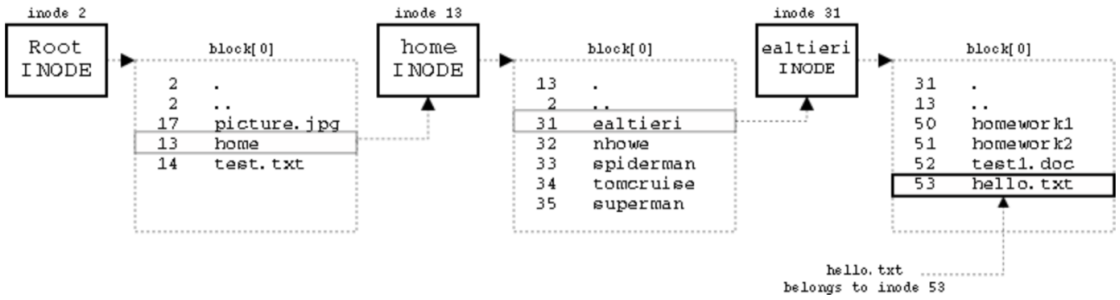
All'interno del file system (in particolare all'inizio del disco o partizione su cui è installato il file system) si trova una **tabella degli inode**.

Ad esempio, nei file system ext2 la tabella degli inode viene conservata all'interno del primo gruppo di blocchi:



Una **directory** non è altro che un **file speciale**; il cui contenuto è costituito da blocchi su disco contenenti tabelle formate da tuple nel formato (inode_file, nome_file).

Ad esempio, il path `/home/documenti/hello.txt` viene seguito esaminando uno ad uno il contenuto dei file puntati dagli inode delle directory intermedie:



Per visualizzare le informazioni contenute dell'inode di un file, il comando ls fornisce **numerevoli** opzioni:

- L'opzione **[-l]** permette di visualizzare permessi di accesso, numero di sottodirectory UID, GID, size, mtime dei file.
  Inoltre, all'inizio dell'output viene visualizzato il numero di blocchi totali occupati
  dalla directory (non è incluso il numero di blocchi occupati dalle sottodirectory),
  dove, normalmente, un blocco è grande tra 1kB e 4kB.
  - Se usato assieme a **[-c]**, viene visualizzato il **ctime** al posto dell'mtime
  - Se usato assieme a **[-u]**, viene visualizzato **l'atime** al posto dell'mtime
- L'opzione **[-i]** permette di visualizzare gli inode number dei file
- L'opzione **[-n]** permette di visualizzare i numeri associati all'UID e al GID, invece del loro nome

Hard link e Soft link

> 💡 Un **hard link** (o collegamento fisico) è un **collegamento che associa un nome ad un file** (e di conseguenza al suo inode). Ogni file possiede almeno un hard link.
> Un
> **soft link** (o shortcut), invece, è un **puntatore al path di un hard link** o un altro soft link.

Se un file possiede più hard link, esso sarà accessibile tramite ognuno di tali link e la dimensione e l'inode number di tali hard link saranno quelli del file stesso, risultando
dunque
identici.

Inoltre, eseguendo il comando `rm`, tale file verrà effettivamente rimosso dal disco solamente quando **verranno rimossi tutti i suoi hard link**

La **dimensione** di un **soft link** corrisponde al numero di **byte necessari a conservare il path puntato.**

Per creare un link è possibile utilizzare il comando `ln [-s] src_link new_link`. Se l'opzione **[-s]** non viene utilizzata, verrà creato un hard link, mentre in caso contrario
verrà creato un soft link.

Poiché un soft link è un puntatore ad un **path**, se il file relativo a tale path viene spostato, rinominato o rimosso, tale soft link non sarà più valido.

Permessi di accesso ai file

Come già accennato, all'interno di ogni inode vengono specificati i **permessi di accesso al file** associato a tale inode.
Tali permessi di accesso corrispondono ad una
terna di terne bit: tre bit per i permessi di accesso del **proprietario (user)**, tre bit

per i permessi di accesso del **gruppo associato**
(group)
e tre bit per **qualsiasi altro utente (other)**.
Per ognuna delle terne di bit, ognuno di essi corrisponde ad un permesso:

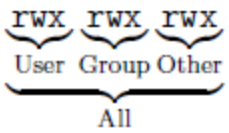
1. Il **primo bit** (partendo da destra) corrisponde al permesso di esecuzione (**execute**), indicato anche con una x. Se attivo, permette di **eseguire il file** (nel caso sia eseguibile).
2. Il **secondo bit** corrisponde al permesso di scrittura (**write**), indicato anche con una w. Se attivo, **permette di sovrascrivere, appendere in scrittura o cancellare** direttamente il file.
3. Il **terzo bit** corrisponde al permesso di lettura (**read**), indicato anche con una r. Se attivo, p**ermette di accedere al contenuto** del file.

Ognuno di tali bit può essere **impostato o non,** concedendo o meno il permesso ad esso associato.

Ad esempio, se i tre bit sono impostati su 011, vengono concessi solo i permessi di scrittura ad esecuzione. Difatti, utilizzando la **notazione alfabetica**, il permesso 011 corrisponde a -wx. Inoltre, trattandosi di terne di bit, il loro valore può essere interpretato anche in **base ottale**

Permesso	Valore binario	Valore ottale
- - -	000	0
- - x	001	1
- w -	010	2
- w x	011	3
r - -	100	4
r - x	101	5
r w -	110	6
r w x	111	7

Dunque, ogni terna di permessi associati ad un **inode** può essere interpretata come:



Nel caso delle **directory** gli effetti ottenuti in base ai permessi associati non risultano del tutto intuitivi:

Permesso	Ottale	Effetto sulla directory
`- - -`	0	Nessuna operazione concessa
`- - x`	1	La directory può essere impostata come `cwd`, ma solo se tale permesso è concesso per ogni directory del path. Inoltre, è possibile "attraversarla" se il contenuto è già conosciuto
`- w -`	2	Nessuna operazione concessa
`- w x`	3	È possibile aggiungere file e directory, cancellare file contenuti in essa (anche senza avere il permesso di scrittura su tali file), cancellare directory contenute in essa (se si hanno tutti i permessi su tali directory)
`r - -`	4	Può essere solo elencato il contenuto della directory (senza gli attributi dei file) e non può essere "attraversata"
`r - x`	5	È possibile elencare il contenuto della directory (attributi compresi), impostare come `cwd` ed "attraversare". Tuttavia, non è possibile cancellare o aggiungere file alla directory
`r w -`	6	Come il permesso 4
`r w x`	7	Come il permesso 3, ma si può anche elencare il contenuto della directory (attributi compresi)

Per **modificare i permessi di accesso** di un file è possibile utilizzare il comando `chmod perms nome_file`, dove nell'argomento **perms** possono essere specificati (in più formati) i permessi da aggiungere, rimuovere o impostare per il file.

Per **modificare il proprietario o il gruppo di appartenenza di un file**, invece, possono essere utilizzati i comandi `chown [-R] user nome_file` e `chgrp [-R] group nome_file`,
dove l'opzione
[-R] applica il comando ricorsivamente su tutte le sottodirectory nel caso in cui nome_file sia una directory.

Tali comandi possono essere utilizzati solo da **root**, richiedendo quindi l'uso di `sudo`