Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2024-2025

Corso di Laurea in Matematica

Permessi.

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Comandi sui file
   - I permessi sui file
2. Filesystem e file
   - File e i-node
   - File e link

Comandi sui file

I permessi sui file

I permessi dei file

- I permessi dei file servono a specificare chi può far cosa
  - **ogni file** ha associato un **utente** ed un **gruppo proprietari**
  - inizialmente, il proprietario è chi crea il file, ed il gruppo è il gruppo primario (ovvero, quello specificato per primo in `/etc/passwd`) di quell'utente
  - *inizialmente* perché si può usare il comando `chown` per cambiare il proprietario (come vedremo)
  - il proprietario decide cosa permettere agli altri utenti e agli altri gruppi, definendo i **permessi** di file e directory

I permessi dei file

- Per i **file** i permessi sono: lettura `r`, scrittura `w` ed esecuzione `x`

Permesso	Ottale	Significato
- - -	0	Non si può fare niente (è però possibile vedere gli attributi, se i permessi sulla directory lo consentono)
- - x	1	Non si può fare niente (per eseguire, si deve prima leggere)
- w -	2	Si può scrivere, ma solo da riga di comando, sovrascrivendo completamente il file o appendendo dati alla fine (per altre modifiche, bisognerebbe prima leggere); si può anche cancellare, ma occorrono opportuni diritti sulla directory
- w x	3	Come il permesso 2
r - -	4	Si può leggere
r - x	5	Si può leggere ed eseguire
r w -	6	Si può leggere e modificare a piacimento (ma attenzione alla cancellazione)
r w x	7	Si può fare tutto (ma attenzione alla cancellazione)

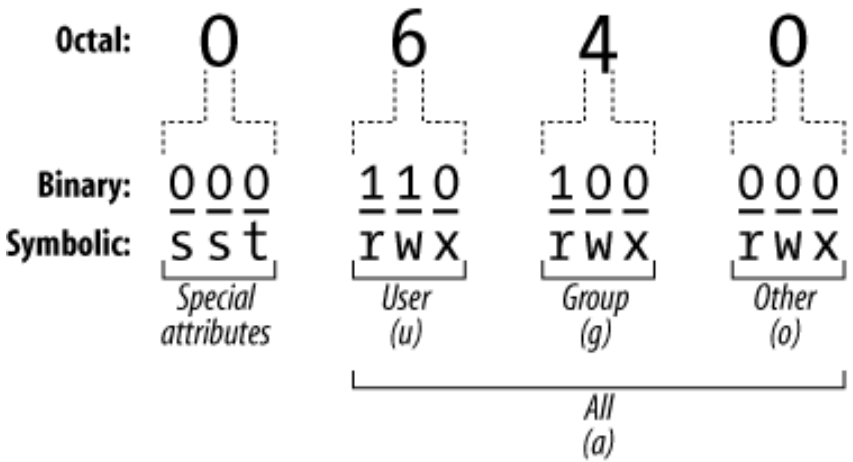
I permessi dei file

- Per le **directory** la cosa è un po' più complicata

Permesso	Ottale	Significato
- - -	0	Non si può fare niente
- - x	1	Si può settare come cwd (ma solo se c'è il permesso per *tutte* le directory nel path); si può anche *attraversare* (cioè vedere un file o una directory al suo interno, se c'è permesso in lettura)
- w -	2	Non si può fare niente (per fare veramente modifiche, occorrono i permessi di esecuzione)
- w x	3	Non si può listare il contenuto (altrimenti sarebbe come il permesso 7)
r - -	4	Si può solo listarne il contenuto, senza vedere gli attributi dei file/directory contenuti (si può sapere se si tratta di file o di directory); non può essere *attraversata*
r - x	5	Si può leggere (attributi compresi), settare come cwd ed attraversare; non è possibile cancellare o aggiungere file/directory
rw -	6	Come il permesso 4 (write senza execute è inutile)
r w x	7	Si può fare tutto: listare contenuto, aggiungere file e directory, cancellare file contenuti (anche senza permesso di scrittura sul file!) e directory contenute (ma occorrono tutti i permessi)

I permessi dei file

- Per ogni file/directory, sono specificati **3 insiemi di permessi** come quelli definiti sopra
  - il primo da sinistra è per l'**utente**: si applica se proprietario e utente utilizzatore coincidono
  - il secondo per il **gruppo**: si applica se l'utente utilizzatore appartiene al gruppo del file
  - il terzo per **tutti gli altri utenti**: si applica nei rimanenti casi
  - vengono mostrati da `ls -l` e `stat`
  - ci sono poi i permessi (o attributi) speciali



I permessi dei file

- Permessi speciali: **setuid bit** (s), **setgid bit** (s), **sticky bit** (t)
  - I permessi speciali consentono di impostare funzioni avanzate sui file o sulle directory
  - Sono rappresentati dai bit:
    - **setuid** - Set User Identification
    - **setgid** - Set Group ID
    - **sticky** - conosciuto come Sticky bit
  - I permessi speciali vanno usati con molta prudenza
  - Vengono **visualizzati al posto del bit di esecuzione**: il setuid nella terna utente, il setgid nella terna gruppo e lo sticky nella terna altro
  - Quando il permesso è impostato appare la lettera **s** per setuid bit e per setgid bit, mentre appare la lettera **t** per sticky bit: se è minuscola il permesso c'è, altrimenti è maiuscola (ed *inutile*)

I permessi dei file

- Il **setuid bit** si usa solo per i file **eseguibili**
- E' il permesso che consente l'esecuzione di un processo come proprietario del file e non come l'utente che ne richiede l'esecuzione
- Quindi impostando questa modalità su un file:
  - il file viene eseguito da qualunque utente del sistema con gli stessi privilegi dell'utente proprietario
  - i privilegi con cui opera il corrispondente processo sono quelli dell'utente proprietario del file e non sono quelli dell'utente che esegue il file,
  - ad esempio, se il proprietario è root, il file viene eseguito con i privilegi di root, indipendentemente da chi lo ha eseguito
  - Ad esempio, il comando `passwd` ha il setuid, che permette ad un utente di modificare la propria password

I permessi dei file

- Il **setgid bit** è l'analogo del *setuid bit*, ma vale per i gruppi
- I privilegi sono quelli del gruppo che è proprietario del file eseguibile
- Il **setgid bit** può essere applicato anche ad una directory e allora ogni file nella directory ha il gruppo della directory, anziché quello primario di chi crea files

- **Esercizio** provare a dare il comando `stat /tmp /usr/bin/passwd` e controllare cosa viene visualizzato

I permessi dei file

- Lo **sticky bit** inibisce il permesso di cancellazione dei file di un altro utente anche dove è autorizzata.
- Lo **sticky bit**, applicato su una directory, *corregge* il comportamento dei permessi write+execute (vedere Tabella directory): si possono cancellare file solo se si hanno i permessi di scrittura su quei file
- Per essere più precisi, lo sticky bit ha il seguente effetto:
  se una directory d appartiene all'utente u e un utente $u' \neq u$ cerca di cancellare un file f in d che non appartiene nè ad u' nè al gruppo cui appartiene u', allora
  senza sticky bit su d, per cancellare f è sufficiente avere i diritti di scrittura su d (nel gruppo *other*), anche se non si hanno i permessi di scrittura su f (sempre su *other*)
  con lo sticky bit per cancellare f sono necessari anche i permessi di scrittura su f

Il comando `chmod`

- Comando `chmod mode[, mode...] filename`: serve a cambiare i permessi
- Due modi di settare il *mode*: ottale o con lettere (simbolico)
  - **ottale**: si danno 4 numeri
  - Il primo numero indica setuid (4), setgid (2) e sticky (1), gli altri numeri vanno da 0 a 7, come nelle Tabelle, e sono per utente, gruppo ed altri
  - Si possono anche dare soltanto 3 numeri, e si intende che i bit speciali sono tutti a 0 (caso più comune)
  - **Esercizio** Creare un file e, usando la modalità ottale, settarne i permessi:
    prima a `rws r-S -w-`
    e poi a `rwx r-- -wT`

Il comando `chmod`

- Comando `chmod mode[, mode...]  filename`
  - **lettere**: qui se ne possono specificare molti, separati da virgole
  - il formato di ogni modo simbolico è :
    `[ugo] [+-=] [perms...]`
    dove `perms` è zero, una o più lettere nell'insieme {rwxst},
    oppure una lettera nell'insieme {ugo}
  - **Esercizio** Creare un file e, usando la modalità simbolica, settarne i permessi:
    prima a `rws r-S -w-`
    e poi a `rwx r-- -wT`

Il comando chmod: esempi ed esercizi

- **Esempio: Attivare suid**
  - Usando la modalità simbolica: `chmod u+s file.txt`
  - Usando la modalità ottale: `chmod 4755 file.txt`
  - Visualizzando si ha:
    `-rwsr-xr-x 1 utente root 500 2020-10-03 18:46 file.txt`
- **Esempio: Disattivare suid**
  - Usando la modalità simbolica: `chmod u-s file.txt`
  - Usando la modalità ottale: `chmod 0755 file.txt`
  - Visualizzando si ha:
    `-rwSr-xr-x 1 utente root 500 2020-10-03 18:46 file.txt`
- **Esercizio** togliere il permesso di lettura ad un file, e poi provare a visualizzarlo con `cat` o con `geany`
- **Esercizio** verificare che `chmod` modifica il ctime del file

Il comando `umask`

- Comando `umask [mode]`: è un comando della bash (non si trova in man, ma si può fare help umask)
  - Definisce quali permessi **negare** al momento della creazione di nuovi file e directory
  - Senza argomenti mostra l'umask corrente, altrimenti setta la maschera dei file al `mode` specificato
  - Si possono specificare solo le 3 terne, non i permessi speciali (che inizialmente sono tutti a 0)
  - I permessi predefiniti sono **666** per i file e **777** per le directory
  - Settando umask, alla creazione i permessi per un file saranno il risultato dell'operazione bit-a-bit *666 AND NOT(umask)*, mentre per una directory saranno *777 AND NOT(umask)*
  - `umask` ha effetto anche su `chmod`
  - **Esercizio:** modificare l'umask in modo che i permessi siano 664, sia che venga creata una directory che un file. Modificare poi in modo che il permesso per una nuova directory sia 775 e per un file sia 664.

I comandi `chown` e `chgrp`

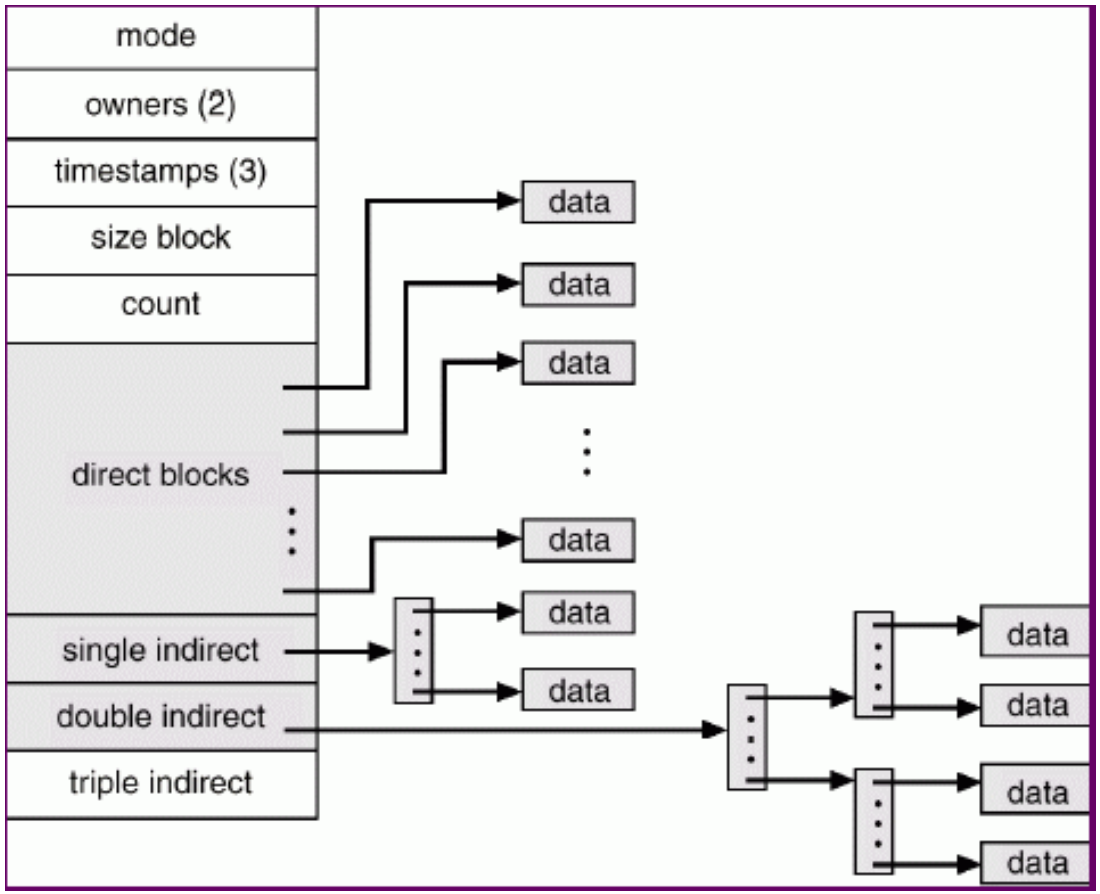
- Comandi `chown [-R] proprietario {file}` e `chgrp [-R] gruppo {file}`: servono a cambiare proprietario o gruppo
  - Possono essere usati solo da root, quindi ci vuole `sudo` (altrimenti chiunque potrebbe creare un file con *contenuti compromettenti* e darlo ad utente ignaro)
  - Se si passano delle directory e c'è l'opzione `-R`, si cambiano ricorsivamente tutti i file e le directory in esse contenute
  - **Esercizio** Creare l'albero di directory `/home/utente1/dir1/dir3/dir7/`
    Creare un file (vuoto) `file`*i* dentro ciascuna directory `dir`*i*
    Cambiare i proprietari in questo modo: la directory `dir3` diventa di `utente3`, tutti i file dentro `dir3` diventano di `utente2`, `dir7` diventa di `utente2` e `file7` diventa di `utente3`
    Infine, provare a cambiare i permessi di `file3` e `dir7`

Filesystem e file

File e i-node

- Per memorizzare file su disco Linux usa gli **i-node**
- Un **i-node** (o inode, abbreviazione di *index node*) è una struttura dati sul filesystem che contiene gli *attributi* base di un *filesystem object*, cioè file, directory o altri oggetti
- Le informazioni includono:
  - la dimensione del file e la sua locazione fisica (se risiede su un dispositivo a blocchi, come ad esempio un hard disk)
  - il proprietario e il gruppo di appartenenza
  - le informazioni temporali di modifica (mtime), ultimo accesso (atime) e di cambio di stato (ctime)
  - il numero di collegamenti fisici che referenziano l'inode
  - i permessi d'accesso
  - un puntatore allo spazio del disco che contiene i file veri e propri

- Ogni file del filesystem (directory comprese) è univocamente determinato da un `inode number` ed è rappresentato da una struttura dati `inode`



- Non ci sono mai contemporaneamente 2 file con lo stesso inode number
- Gli inode number *liberati* dalla cancellazione di un file verranno riusati alla prima occasione
- Esiste una **tabella di tutti gli inode**, che si trova solitamente all'inizio del disco

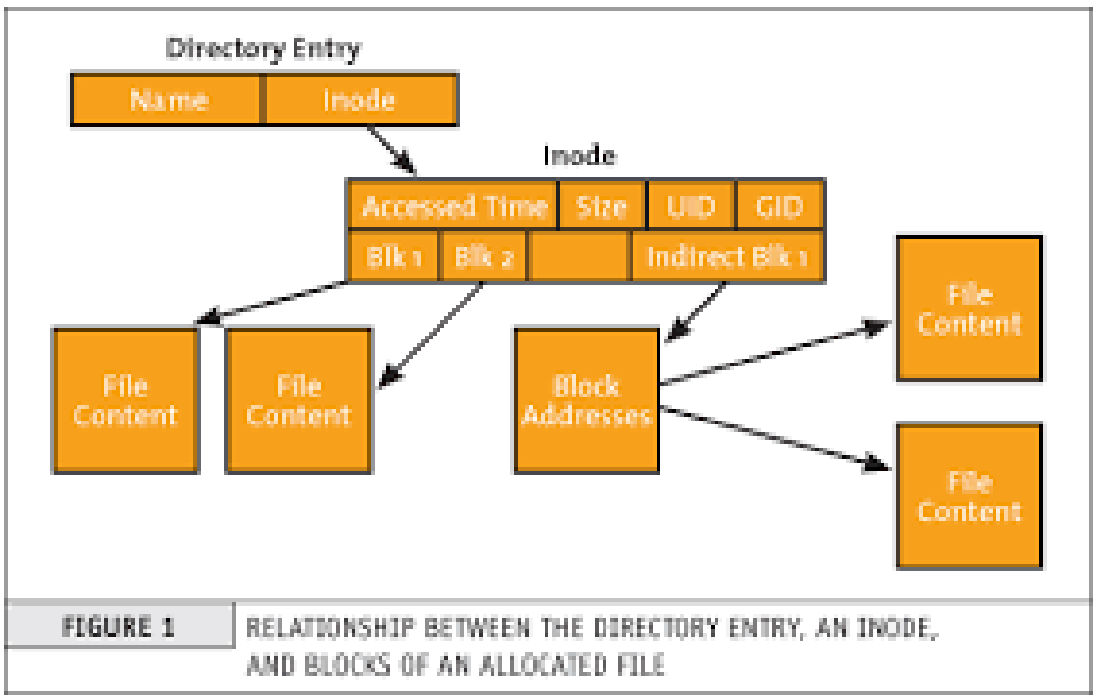


FIGURE 1 RELATIONSHIP BETWEEN THE DIRECTORY ENTRY, AN INODE, AND BLOCKS OF AN ALLOCATED FILE

- Tabella dei principali attributi della struttura dati `inode`

Attributo	**Spiegazione**
Type	Tipo di file (regular, block, fifo ...)
User ID	Id dell'utente proprietario del file
Group ID	Id del gruppo a cui è associato il file
Mode	Permessi (read, write, exec) di accesso per il proprietario, il gruppo e tutti gli altri
Size	Dimensione in byte del file
Timestamps	ctime (inode changing time: cambiamento di un attributo), mtime (content modification time: solo scrittura), atime (content access time: solo lettura)
Link Count	Numero di hard links
Data Pointers	Puntatore alla lista dei blocchi che compongono il file; se si tratta di una directory, il contenuto su disco è costituito di una tabella con 2 colonne: nome del file/directory e suo inode number

Il comando `ls`

- In riferimento alle opzioni del comando `ls`:
  - Per vedere l'i-node number dei file e delle directory, si può usare l'opzione `-i`
  - Per vedere quando i metadati associati al file, sono stati modificati l'ultima si può usare `ls -lc` che mostra il `ctime` e corrisponde all'ultima volta che è stato modificato l'i-node
  - Per vedere l'ID (numerico) di utente e gruppo, anzichè il corrispettivo nome, si può usare l'opzione `-n`

Filesystem e file

Soft link e hard link

Il comando `ln`

- Comando `ln [-s] sorgente [destinazione]`
  - `ln` è un comando dei sistemi operativi Unix (e Unix-like) che crea **collegamenti simbolici** e **collegamenti fisici** a file e directory
  - Se c'è l'opzione `-s` crea un **soft link**:
    - `ln -s file.txt softlink.txt`
  - altrimenti crea un **hard link**:
    - `ln file.txt hardlink.txt`
  - In pratica corrisponde a copiare un file, senza copiarne l'intero contenuto (che potrebbe essere grande)
  - Successive modifiche al *contenuto* del sorgente si rifletteranno sulla destinazione e viceversa
  - Se c'è un solo argomento, allora la sorgente dev'essere un file in un'altra directory e il link destinazione avrà lo stesso nome di questo file

Il comando `ln`

- Comando `ln [-s] sorgente [destinazione]`
  - **soft link**: viene creato un nuovo file (destinazione), il contenuto del quale coincide con `sorgente` (lo si può intuire guardando la dimensione del file con `ls -l` o con `stat`); da notare che spesso questo contenuto è direttamente negli attributi del file
  - Se si cancella il file sorgente, il link diventa *morto* e provare a visualizzare il file porta ad un errore
  - **hard link**: aumenta il link count della sorgente, e crea una destinazione con lo stesso link count
  - Se si cancella il sorgente, il link count decresce, ma la destinazione continua a mantenere il contenuto del file
  - Se si fa un hard link di un hard link, il link count cresce ancora (per tutti i file coinvolti)

Il comando `ln`

- Comando `ln [-s] sorgente [destinazione]`
  - Cancellare un file non vuol dire più automaticamente rimuovere il suo contenuto dal disco: potrebbero esserci hard links...
  - Per vedere a cosa *punta* un hard link, occorre visualizzare l'inode number
  - Per vedere a cosa punta un soft link, basta `ls -l`
  - Con i soft link c'è un *puntatore* ed un *puntato* e cancellare il puntatore non ha effetti sul puntato, mentre cancellare il puntato ha effetti sul puntatore
  - Con gli hard link è come se fossero tutti puntatori al file su disco (o nell'area di memoria collegata al rispettivo filesystem)
  - non si possono fare hard links a directory (tranne quelli predefiniti `..` e `.`), mentre si possono fare i soft links

Il comando `ln`

- **Esercizio**
  - Creare hard e soft link a file sia nella directory corrente che in sottodirectory e verificare con `ls -l`
  - Verificare con `ls -la` `ls -li`, con `ls -lh`
  - Provare a spostare il collegamento (cioè il softlink) in un'altra directory e a visualizzare il contenuto utilizzando il comando `less`
  - Provare a fare l'hard link di un soft link e viceversa