SC - 28/5/2025

28/05/25 10:15

COME È ORGANIZZATO INTERNAMENTE IL FILE SYSTEM

Consideriamo qui i FS della famiglia "ext".

Ogni file / directory è associato ad una struttura dati chiamata "inode" viene caricata in memoria e gestita dal SO. Attraverso questa struttura dati ogni "periferica" compatibile può essere utilizzata con le astrazioni proprie del FS.

L'inode associato ad un file contiene ESCLUSIVAMENTE metadati. I dati relativi al file sono memorizzati separatamente in BLOCCHI di dimensione fissa.

Cosa contiene un inode:

- Numero di inode : indice nella tabella degli inode
- Tipo di oggetto : file, directory, ...
- permessi di accesso : definiscono i diritti dei diversi utenti del sistema rispetto al file/directory

utente proprietario : leggere R , scrivere W , eseguire X
gruppo " : " R , " W , " X
tutti gli altri : " R , " W , " X

$$\underbrace{R\ W\ X}_{\text{utente}}\ \underbrace{R\ W\ X}_{\text{gruppo}}\ \underbrace{R\ W\ X}_{\text{altri}}$$

Es.: $\underbrace{\texttt{r - x}}\underbrace{\texttt{r - x}}\underbrace{\texttt{- - -}}$ ⇒ utente prop. legge ed esegue
gruppo " " " "
altri nessun diritto.

$\underbrace{\texttt{r w -}}\underbrace{\texttt{- - -}}\underbrace{\texttt{- - -}}$

Per modificare i permessi da terminale usiamo "chmod"
" " i proprietari " " " "chown"

I diritti possono anche essere espressi con notazione ottale

$\underbrace{\texttt{r - x}}\underbrace{\texttt{r - x}}\underbrace{\texttt{- - -}}$

$\underbrace{1\ 0\ 1}_{5}\ \underbrace{1\ 0\ 1}_{5}\ \underbrace{0\ 0\ 0}_{0}$ → chmod 550 <filename>

$\underbrace{r\ w\ -}_{6}\ \underbrace{r\ -\ -}_{4}\ \underbrace{r\ -\ -}_{4}$ → 〃 644 〃

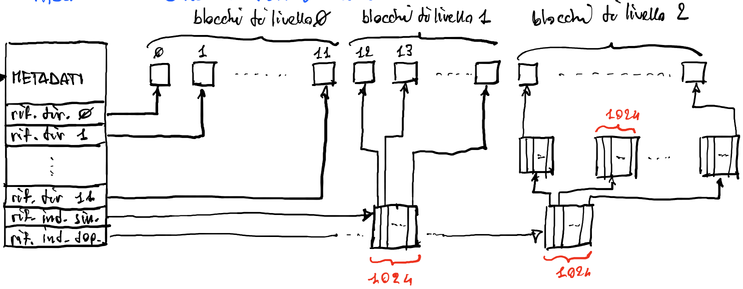
chmod 777 /etc → MOLTO MALE!
Tutti gli utenti possono fare tutto.

- Numero di hard-link : numero di collegamenti esistenti per questo file
- User ID : identificativo dell'utente proprietario
- Group ID : 〃 〃 gruppo 〃
- Size : Dimensione in byte del file
- Timestamp ultimo accesso al file
- 〃 ultima modifica 〃
- 〃 〃 〃 all'inode
- 〃 di creazione del file

} ci dicono cosa è successo al file

- riferimenti DIRETTI ai blocchi (12)
- riferimento INDIRETTO SINGOLO ai blocchi
- riferimento INDIRETTO DOPPIO ai blocchi

} puntatori ai blocchi contenenti i dati del file


blocchi di livello 0
blocchi di livello 1
blocchi di livello 2
0
1
11
12
13
METADATI
rif. dir. 0
rif. dir 1
rif. dir 11
rif. ind. sin.
rif. ind. dop.
1024
1024
1024

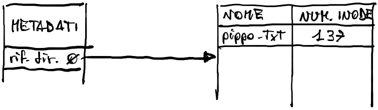
Tipicamente ogni blocco contiene 4KB

Qual'è la dimensione massima di un file con questa struttura?

MAX = 12 × 4096 + 1024 × 4096 + 1024 × 1024 × 4096 = 4 GB

Spazio per indici = (12+2) × 4 + 4096 + (1024+1) × 4096 ≈ 4 MB

E le directory?


METADATI
rif. dir. 0
NOME
NUM. INODE
pippo.txt
137

① Quanto spazio occupa su disco un file da 256 byte ?

Occupa 1 blocco da 4KB

② " " " " " " " " 16000 byte ?

Occupa 4 blocchi da 4KB

③ " " " " " " " " 4 MB ?

4MB = 4 194 304 byte

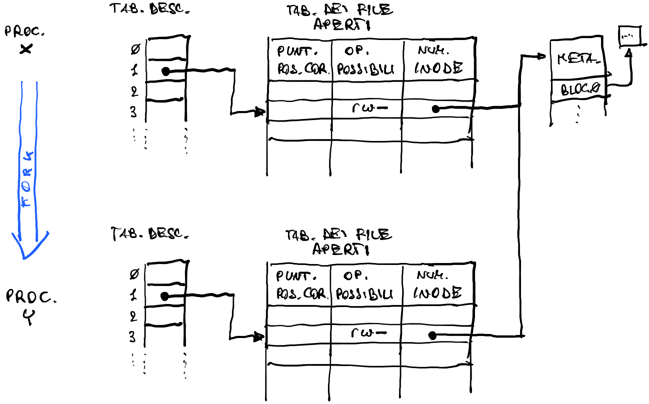
4MB/4KB = 1024 blocchi

- 12 indirizzati direttamente
- 1012 " a livello 1
- 1 blocco di indirizzi per rif ind. singolo

num totale $(12 + 1012 + 1) \times 4KB \approx$

TABELLA DEI DESCRITTORI

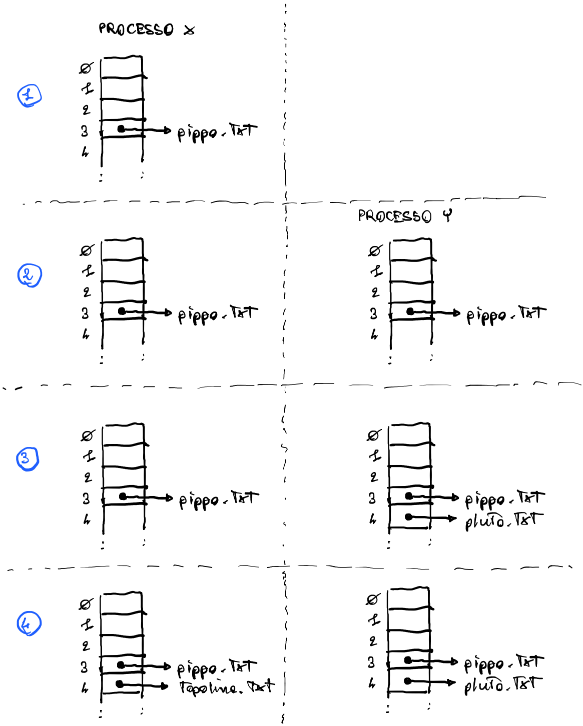
Per ogni processo il SO mantiene una tabella dei descrittori contenente la lista dei file (o meglio degli stream I/O) aperti da quel processo


TAB. DESC.
TAB. DEI FILE APERTI
PROC. X
0
1
2
3
PUNT. POS. COR.
OP. POSSIBILI
NUM. INODE
rw-
META
BLOCCO
FORK
TAB. DESC.
TAB. DEI FILE APERTI
PROC. Y
0
1
2
3
PUNT. POS. COR.
OP. POSSIBILI
NUM. INODE
rw-

① x = fopen ("pippo.txt");

```
② if (fork() == 0)
③        y = fopen("pluto.txt")
   else
④        z = fopen("topolino.txt")
```

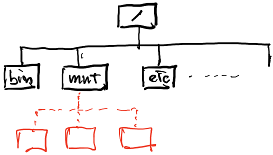
TABELLE DEI DESCRITTORI



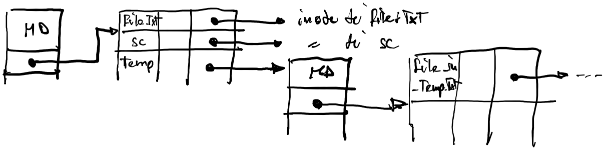
PROCESSO X
PROCESSO Y
①
②
③
④
0
1
2
3
4
pippo.txt
pluto.txt
topolino.txt

GESTIONE DEI DISPOSITIVI

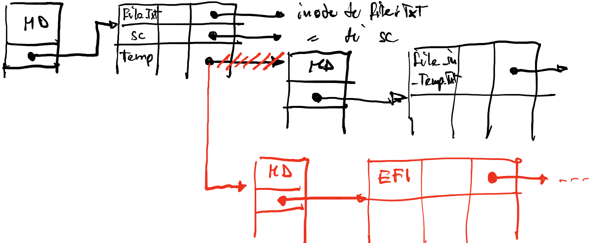
La funzione "mount" rende disponibile il FS di un dispositivo posizionandolo all'interno della struttura gerarchica generale.



Cosa c'è nell'inode di /mnt ?


HD
File.txt
sc
Temp
inode di file.txt
" di sc
HD
file_in_Temp.txt
- - -

mount /dev/sda1 /mnt/Temp


HD
File.txt
sc
Temp
inode di file.txt
" di sc
HD
file_in_Temp.txt
HD
EFI
- - -

inode di "/" in /dev/sda1

OPERAZIONI SU FILE E DIR

`# include <dirent.h>`

`DIR * opendir (const char * dirname);`

↑ struttura dati che ci permette di operare su dir

↑ path della directory

`struct dirent * readdir ( DIR * dirp );`

↳ permette di accedere al contenuto della directory puntata da dirp

↳ struct contenente informazioni sulla prossima entry contenuta nella directory

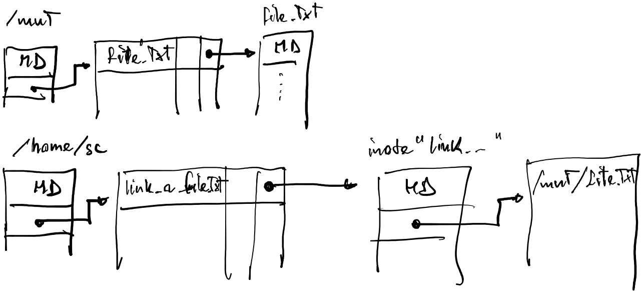
- ino_t d_ino → numero di inode
- char * d_name → nome file/dir

`int closedir (DIR * dirp)`

LINK A FILE/DIRECTORY

Un link è una scorciatoia o collegamento verso un file/directory

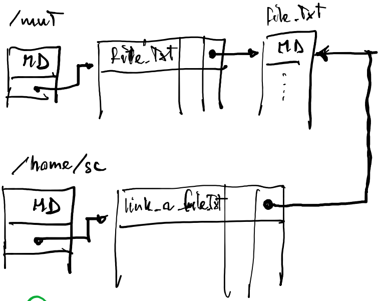
LINK SIMBOLICO : è costituito da un file contenente il path del file/dir collegato.


/mnt
MD
file.txt
file.txt
MD
/home/sc
MD
link_a_file.txt
inode "link.."
MD
/mnt/file.txt

⊕ i link simbolici sono "portabili" tra piattaforme diverse

⊖ il link è "fragile" : se sposto il file collegato il link si rompe

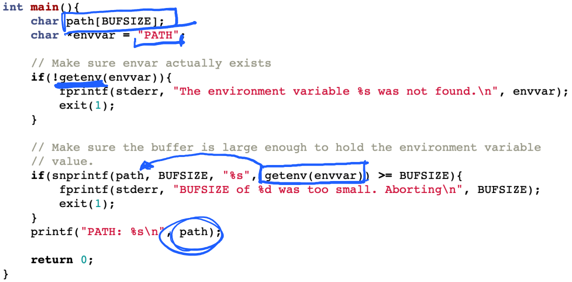
HARD LINK : è un nuovo riferimento ad un inode esistente


/mnt
MD
file.txt
file.txt
MD
/home/sc
MD
link_a_file.txt

⊕ gli hard link sono robusti rispetto ad operazioni di spostamento

⊖ non sempre sono compatibili con altri FS

VARIABILI D'AMBIENTE



```
int main(){
    char path[BUFSIZE];
    char *envvar = "PATH";

    // Make sure envar actually exists
    if(!getenv(envvar)){
        fprintf(stderr, "The environment variable %s was not found.\n", envvar);
        exit(1);
    }

    // Make sure the buffer is large enough to hold the environment variable
    // value.
    if(snprintf(path, BUFSIZE, "%s", getenv(envvar)) >= BUFSIZE){
        fprintf(stderr, "BUFSIZE of %d was too small. Aborting\n", BUFSIZE);
        exit(1);
    }
    printf("PATH: %s\n", path);

    return 0;
}
```