SC 27/5/2026

mercoledì 27 maggio 2026 10:16

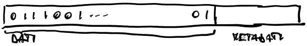
INTRODUZIONE AL FILE SYSTEM

È l'insieme degli schemi organizzativi dei dati e delle relative funzionalità attraverso cui il sistema operativo si interfaccia con le periferiche di archiviazione dati

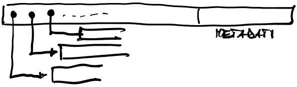
FILE E DIRECTORY

FILE: contenitore di dati (archivio) rappresentati come sequenze di byte. Oltre ai dati il file è caratterizzato da alcuni metadati necessari per il suo funzionamento:

- nome
- dimensione
- permessi di accesso
- Timestamp

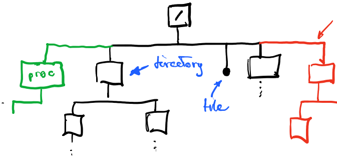

0 1 1 1 0 0 1 ---- 0 1
DATI
METADATI

DIRECTORY: contenitore logico che permette l'organizzazione dei dati in una struttura di qualche forma (tipicamente un albero)



STRUTTURA GERARCHICA DEI FILE SYSTEM

Nel mondo UNIX il contenuto del file system è organizzato in un unico albero. È presente una directory speciale detta "root directory" rappresentata "/"


/
proc
directory
file

periferica esterna aggiunta (es. USB Key). Vengono aggiunte alla struttura con il comando mount

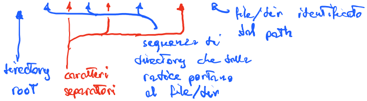
FILE SYSTEM HIERARCHY STANDARD (FSHS)

PATH / PERCORSO

Nel FS ogni file/directory è identificato UNIVOCAMENTE dal suo **PATH**

PATH ASSOLUTO : sequenza di tutte le directory presenti nel ramo del file system che partendo dalla root dir. arriva fino al file/dir considerato

Es.: /home/Tom/documents/text



file/dir identificato dal path
directory root
caratteri separatori
sequenza di directory che dalla radice portano al file/dir

ATTENZIONE : "/" è un path valido
"//" NON è un path valido

PATH RELATIVI : sequenza di directori nel FS che collegano attraverso un percorso valido il file/dir considerato con una directory di riferimento passando per la directory "antenato" comune ad entrambi i rami.

Per definire un path relativo usiamo dei percorsi speciali :

"." indica la directory corrente

".." " " " padre di quella corrente

"~" " " " home dell'utente

Es.: per individuare "other" partendo da "documents"

../../other

COME È ORGANIZZATO INTERNAMENTE IL FILE SYSTEM

Consideriamo solo i FS della famiglia "Ext"

Ogni file/dir è associato ad una struttura dati chiamata **INODE** che viene caricato in memoria e gestita dal SO. Attraverso l'inode 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 dell'inode : indice numerico nella tabella degli inode
- Tipo di oggetto : file, directory, ...
- Permessi di accesso : definiscono i diritti con cui i diverse utenti del sistema accedono al file/dir

diritti utente proprietario : leggere R, scrivere W, eseguire X
〃 gruppo 〃 : 〃 R, 〃 W, 〃 X
〃 di tutti gli altri : 〃 R, 〃 W, 〃 X

$$\underbrace{RWX}_{\text{utente}}\quad\underbrace{RWX}_{\text{gruppo}}\quad\underbrace{RWX}_{\text{altri}}$$

Es.: $\underbrace{r-x}\underbrace{r-x}\underbrace{r-x} \Rightarrow$ utente prop. legge ed esegue
gruppo 〃 〃 〃 〃
altri 〃 〃 〃

rw- - - - - - - - ⟹ solo utente prop. legge e scrive

Per modificare i permessi/diritti si usa "chmod"
〃 〃 utente/gruppo proprietario "chown"

Es.: chmod a+r Text.txt

chmod g-x file_Txt

I diritti possono essere rappresentati anche con notazione ottale

Es.:

$$\underbrace{\underbrace{r\ -\ x}_{4\ 0\ 1}}_{5}\ \underbrace{\underbrace{r\ -\ x}_{4\ 0\ 1}}_{5}\ \underbrace{\underbrace{-\ -\ -}_{0\ 0\ 0}}_{0} \longrightarrow \text{chmod 0550 nomefile}$$

$$\underbrace{r\ w\ -}_{6}\ \underbrace{r\ -\ -}_{4}\ \underbrace{r\ -\ -}_{4}$$

Es.: chmod 777 /etc PESSIMA IDEA!

Come faccio a definire i diritti di un file che creo in un programma?

Usato la syscall "open"

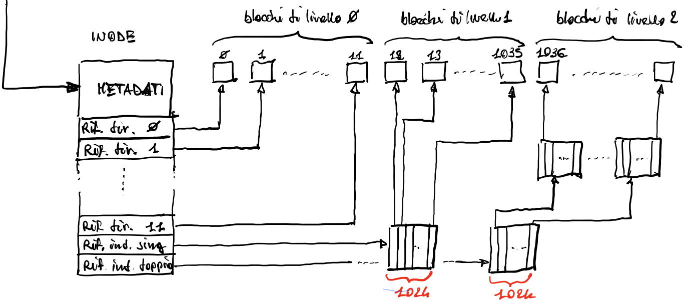
- Numero degli hard-link : numero di collegamenti esistenti per il file
- User ID : identificativo dell'utente proprietario

- Group ID : ″ ″ gruppo ″
- size : dimensione del file in byte
- Timestamp ultimo accesso
- ″ ″ modifica
- ″ ″ ″ all'inode
- ″ di creazione

ci dicono cosa è successo al file

- riferimenti DIRETTI a blocchi dati (12)
- riferimento INDIRETTO SINGOLO ″ ″ ″
- riferimento INDIRETTO DOPPIO ″ ″ ″

Puntatori ai blocchi contenenti i dati


blocchi di livello 0
blocchi di livello 1
blocchi di livello 2
INODE
METADATI
Rif. dir. 0
Rif. dir. 1
Rif. dir. 11
Rif. ind. sing
Rif. ind. doppio
0
1
11
12
13
1035
1036
1024
1024

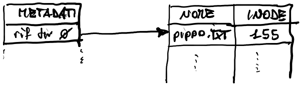
Tipicamente ogni blocco contiene 4KB

Qual'è la dimensione massima di un file?

$$MAX = 12 * 4096 + 1024 * 4096 + 1024^2 * 4096 = 4GB$$

$$\text{spazio indice} = (12+2) * 4 + 4096 + (1024+1) \times 4096 \simeq 4MB$$

Per le directory


METADATI
rif dir 0
NOME
INODE
pippo.txt
155

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

Occupa 1 blocco => 4KB

② ″ ″ ″ ″ ″ ″ ″ ″ 16000 byte?

Occupa 4 blocchi => 16KB

③ ″ ″ ″ 4MB?

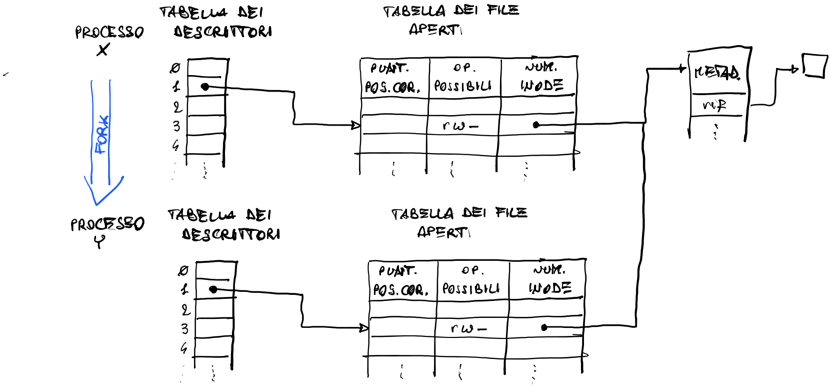
4MB = 4 194 304 byte

$4\,MB/4\,KB = 1024$ blocchi
- 12 indirizzati direttamente
- 1012 " a livello 1
- 1 blocco di nr. ind singoli

num Totale = $(12+1012+1) * 4096$

TABELLA DEI DESCRITTORI

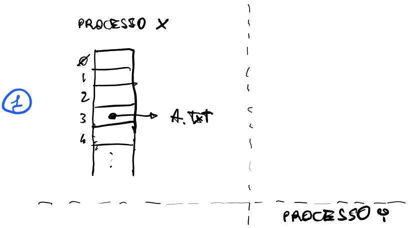
Per ogni processo il SO mantiene una Tabella dei descrittori contenente la lista dei file (stream I/O) aperti dal processo.


PROCESSO X
TABELLA DEI DESCRITTORI
TABELLA DEI FILE APERTI
0
1
2
3
4
PUNT. POS. COR.
OP. POSSIBILI
NUM. INODE
rw-
METAD.
nr
FORK
PROCESSO Y
TABELLA DEI DESCRITTORI
TABELLA DEI FILE APERTI
0
1
2
3
4
PUNT. POS. COR.
OP. POSSIBILI
NUM. INODE
rw-

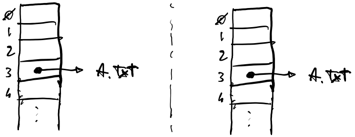
Esempio:

```
①  x = fopen ("A.TxT");
②  if (fork() == 0)
③            y = fopen ("B.TxT");
      else
④            z = fopen ("C.Txt");
```

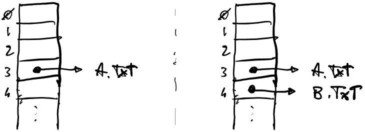
TABELLE DESCRITTORI


PROCESSO X
①
0
1
2
3
4
A.txt
PROCESSO Y

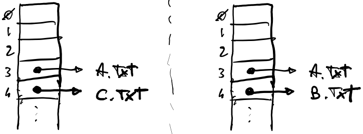
②


0
1
2
3
4
A.txt
0
1
2
3
4
A.txt

③


0
1
2
3
4
A.txt
0
1
2
3
4
A.txt
B.txt

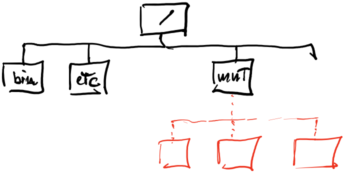
④


0
1
2
3
4
A.txt
C.txt
0
1
2
3
4
A.txt
B.txt

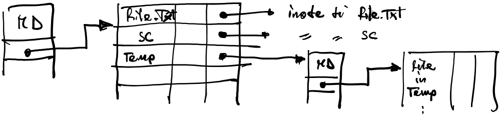
ATTENZIONE con thread multipli la tabella è unica.

GESTIONE (MOUNT) DEI DISPOSITIVI

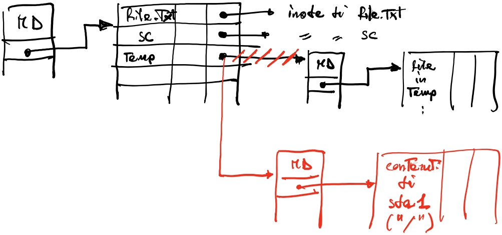
L'utility mount rende disponibile il FS di un dispositivo nella struttura gerarchica complessiva.



Immaginiamo il contenuto di mnt


MD
file.txt
sc
Temp
inode di file.txt
" " sc
MD
file in Temp

se eseguiamo "mount /dev/sda1 /mnt/Temp


MD
file.Txt
SC
Temp
inode di file.Txt
= = SC
MD
file in Temp
MD
contenuti di sda 1 ("/")

LINK

E' un collegamento a file/dir.

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

- + è portabile
- - non supporta spostamento del file

HARD LINK : è un nuovo rif. ad inode esistente

- + robusto a spostamento
- - non portabile

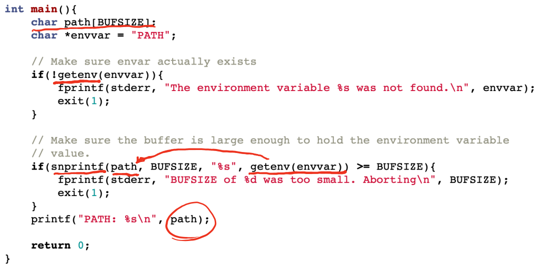
ATTENZIONE : un inode e relativi blocchi vengono cancellati quando non ci sono più link che lo puntano

VARIABILI DI AMBIENTE

env → stampa le variabili definite

export <nome> = <valore> → definisce una variabile

unset <nome> → elimina una variabile



```
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;
}
```