Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2017-18

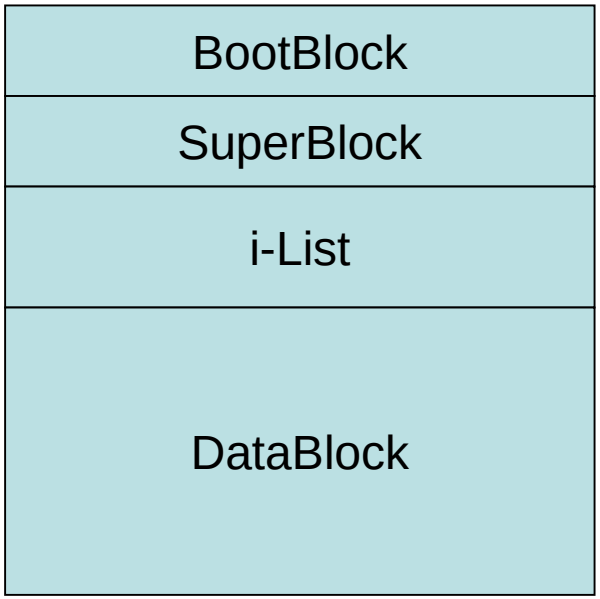
Pietro Frasca

Lezione 23

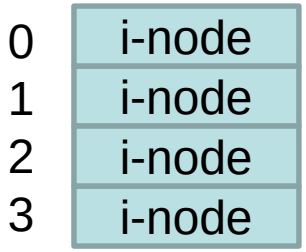
Giovedì 11-01-2018

Organizzazione fisica del file system

- Il file system di unix può essere allocato su vari dischi.
- Un disco prima del suo uso deve essere **formattato** in blocchi di dimensione fissa.
- Il disco viene suddiviso in 4 aree: **bootblock**, **superblock**, **i-list**, **datablock**.



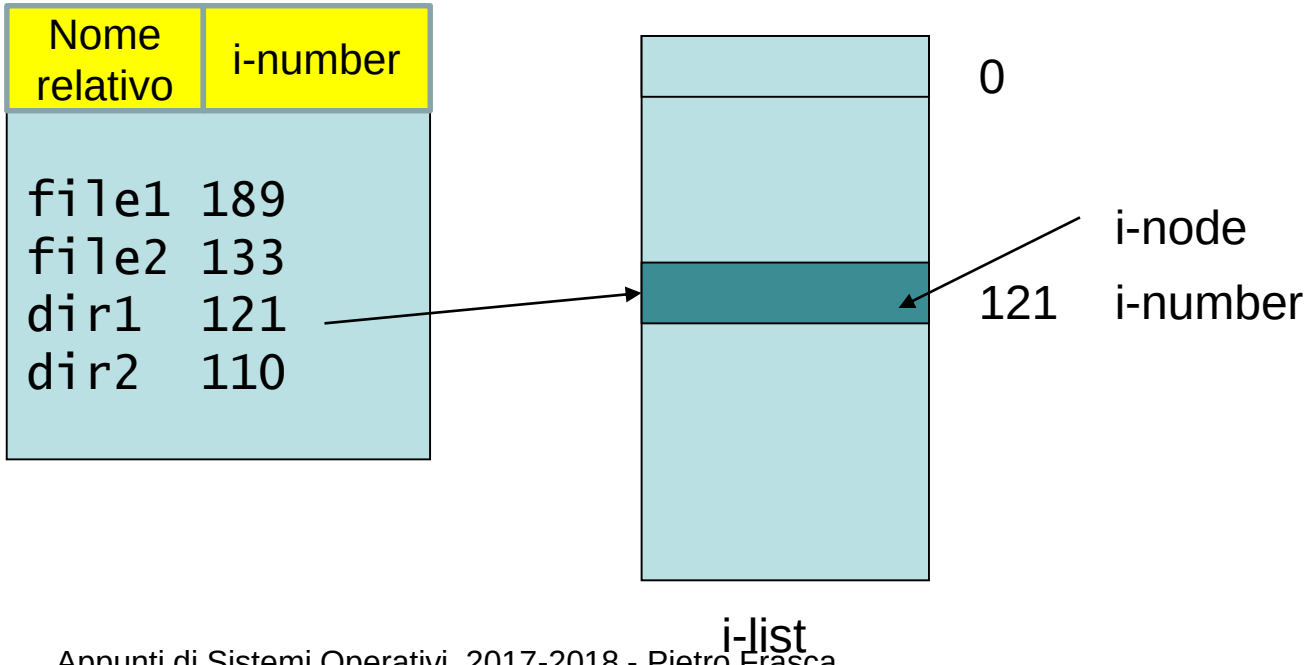
- L’area di **bootblock** ha dimensione di un blocco e contiene il programma di avvio (boot) del sistema.
- La **i-list** contiene la tabella di tutti i **descrittori** (**i-node**) dei file, directory e dispositivi contenuti nel file system. Ogni i-node è individuato mediante l’indice (i-number) della tabella (vettore).
- L’area **datablock** contiene effettivamente i file. I blocchi liberi di questa area sono organizzati in una lista collegata, il cui indirizzo è memorizzato nel superblock.
- L’area **superblock** ha dimensione di un blocco, descrive come è allocato il filesystem; contiene gli indirizzi delle 4 aree, il puntatore alla lista dei blocchi liberi e il puntatore alla lista degli **i-node** liberi.



- L’**i-node** è il descrittore del file e contiene le proprietà associate al file stesso. Tra le proprietà più importanti:
  - **Nome, Dimensione**, **Data**
  - **Tipo di file** (ordinario, directory, speciale..)
  - **Protezione** (i bit di protezione che ne indicano i diritti di accesso. Sono 12 bit: 9 per indicare la protezione e gli altri tre sono relativi a SUID, SGID e Sticky)
  - **Numero di link:** numero di nomi del file (numero di link hardware)
  - **Proprietario**, **Gruppo**
  - **Vettore di indirizzamento:** è costituito da un insieme di indirizzi (ad esempio 13 puntatori) che consente l’indirizzamento dei blocchi sui quali è allocato il file.

- Le prime 8 proprietà sopra elencate dei file (contenuti nella directory corrente) sono visibili con il comando
  ls –l.

- Il metodo di allocazione è ad indice, a più livelli di indirizzamento.
- **Directory.** La directory è rappresentata da un file, il cui contenuto ne descrive la struttura logica. Ogni record logico della directory contiene la coppia <***nome relativo***, ***i-number***> che identifica un file o una directory contenuti nella directory considerata.



Strutture dati del kernel per l'accesso ai file

- In Unix un file è organizzato come una sequenza di byte.
- E' possibile accedere al file nelle modalità: lettura, scrittura e scrittura in aggiunta *(append).*
- Prima di accedere ad un file è necessario eseguire **l'operazione di *apertura (open)***, mediante la quale sono aggiornate le strutture dati relative al file gestite dal kernel.
- Per l'accesso e la gestione dei file, il kernel mantiene alcune strutture dati specifiche.
- A ogni processo è associata una ***tabella dei file aperti del processo (TFAP)*** di dimensione limitata (tipicamente 20 elementi), nella quale ogni riga della tabella rappresenta un file aperto dal processo.

- L’indice di riga della ***TFAP*** è detto ***file descriptor****.*
- Le prime tre righe della *TFAP* sono inizializzate automaticamente per rappresentare **standard input** ***(file descriptor 0***), **standard output** ***(file descriptor 1***) e **standard error** ***(file descriptor 2*****).**
- La *TFAP* è una struttura dati accessibile soltanto dal kernel e fa parte della **User Structure** del processo.

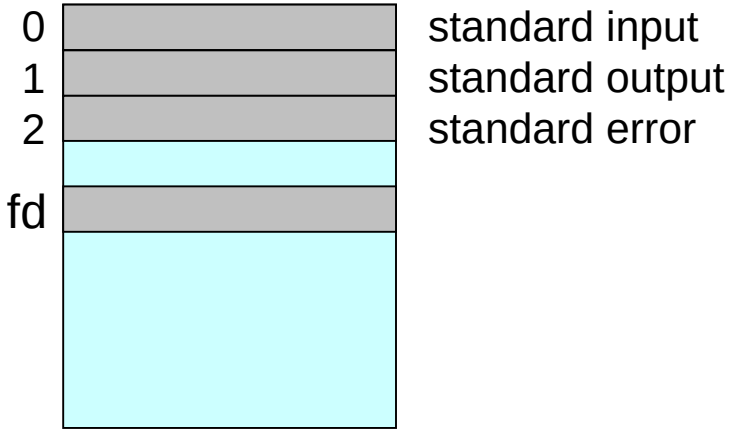
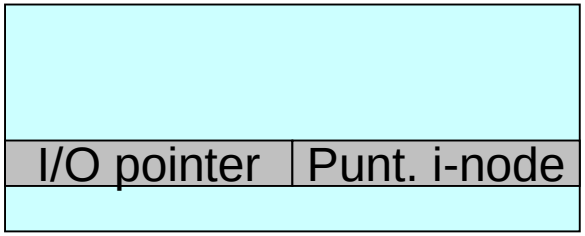


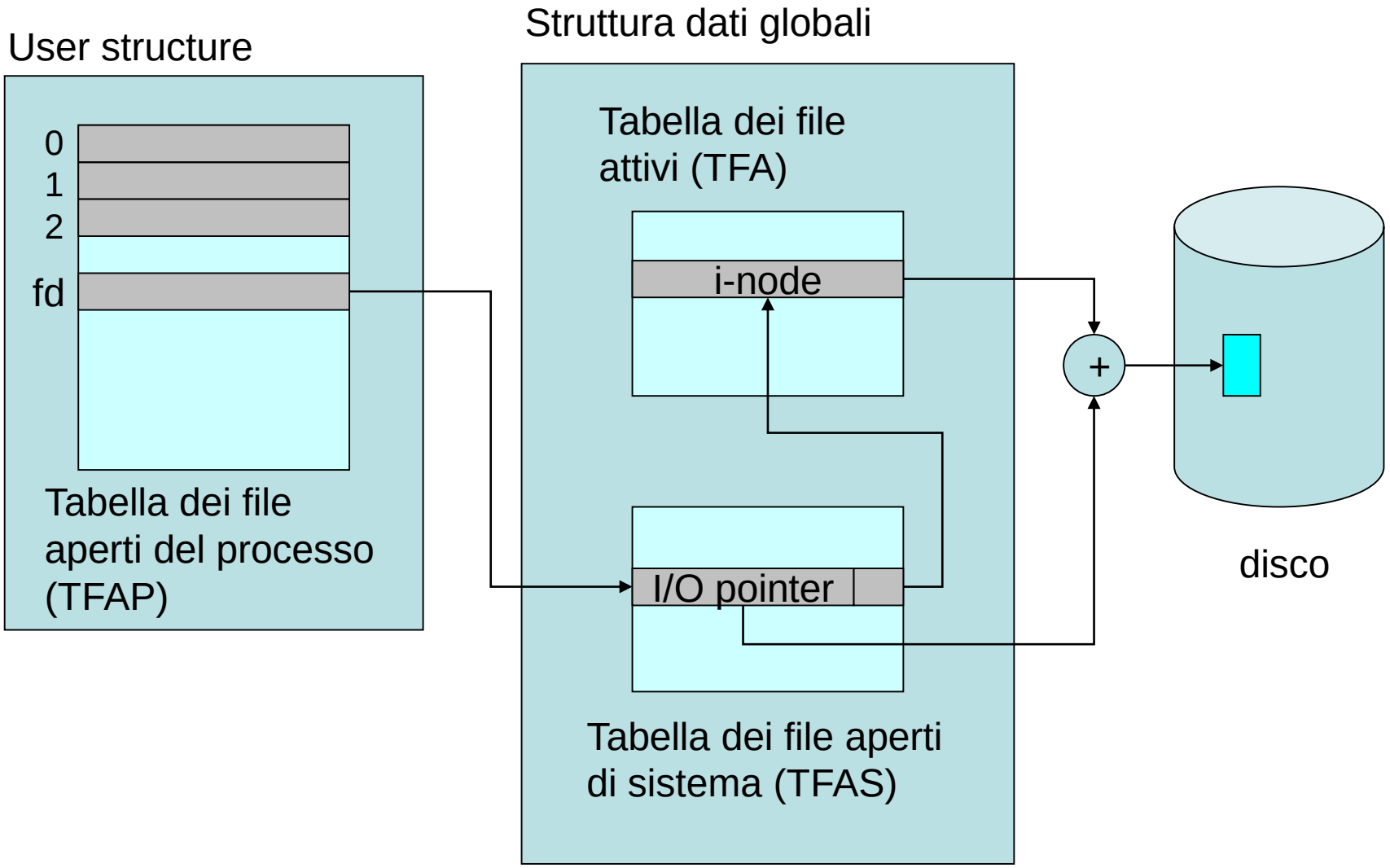
Tabella dei file aperti del processo (TFAP)

- A livello globale il kernel mantiene la ***Tabella dei File Aperti di Sistema*** (***TFAS****)* che contiene una riga per ogni operazione di apertura di file. Pertanto, se due processi aprono lo stesso file, nella ***TFAS*** saranno aggiunte due righe distinte.
- Ogni elemento della *TFAP* contiene un riferimento all'elemento corrispondente nella *TFAS.*
- Tra le informazioni contenute nell'elemento della *TFAS,* c'è l'***I/O pointer****,* che indica il prossimo byte da leggere e/o scrivere nel file aperto. Inoltre, è presente un **riferimento all'inode** (descrittore) del file aperto che il sistema carica e mantiene in memoria RAM sino a quando il file viene chiuso.



TFAS

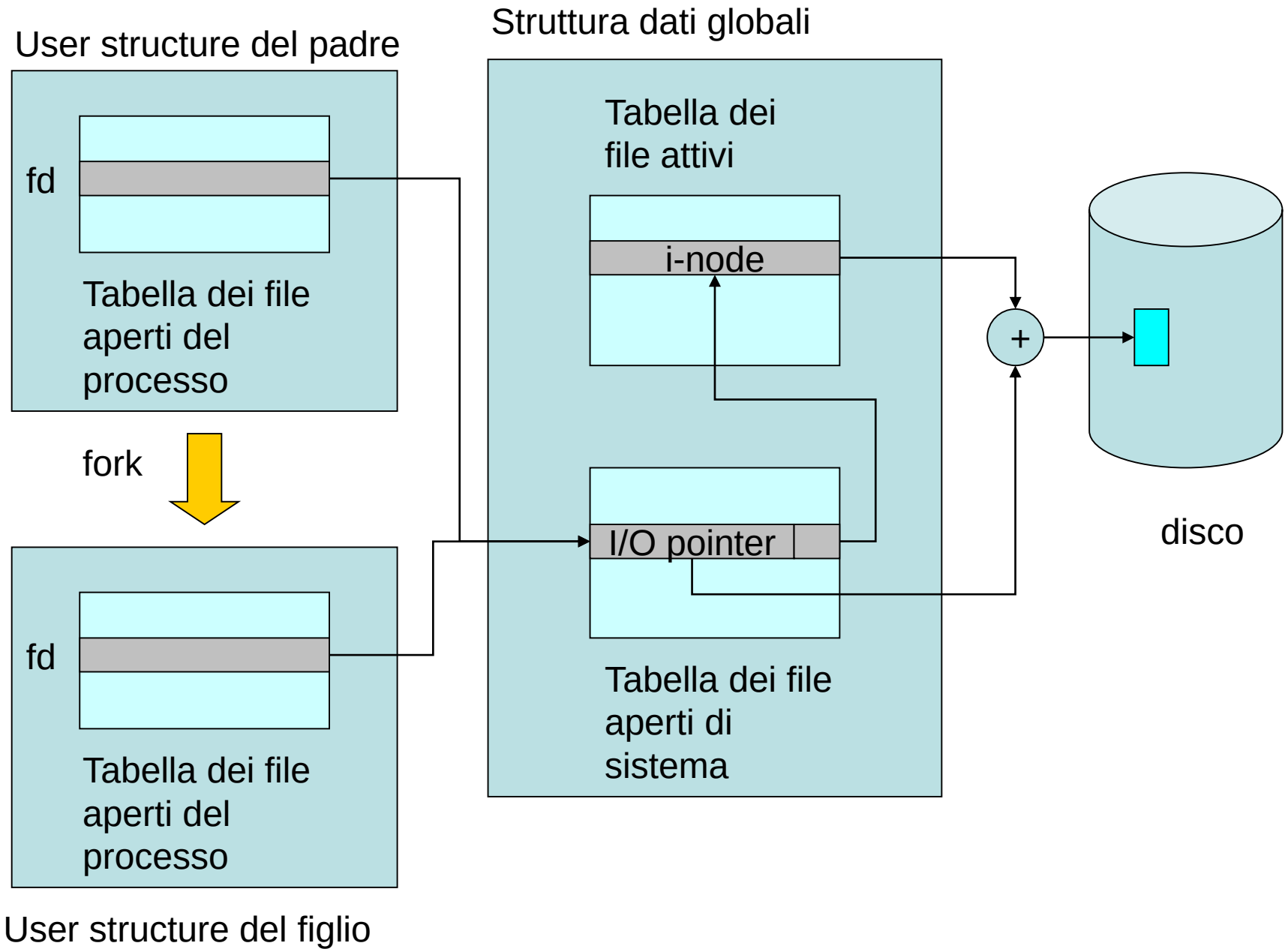
- Gli **i-node** dei file aperti sono inseriti all'interno di un'altra tabella globale: la ***Tabella dei File Attivi*** (***TFA****).*
- La figura seguente mostra come le tre tabelle sono tra loro in relazione. Si può vedere come, a partire dal file descriptor **fd**, si possa ricavare l'indirizzo del prossimo byte da leggere/scrivere sul file utilizzando i dati contenuti nelle tre strutture dati.



Strutture dati del kernel per l'accesso ai file

- Come già visto, infatti, l'i-node (memorizzato nella *TFA)* contiene il vettore di indirizzi che descrive l'allocazione del file sul disco; conoscendo inoltre il valore dell'I/O pointer, si può quindi calcolare l'indirizzo fisico <blocco, offset> del prossimo byte da leggere/scrivere sul disco.
- L'operazione di apertura di un file da parte di un processo *P* determina sulle strutture dati del kernel i seguenti effetti:
  - viene inserita una nuova riga (individuata da un file descriptor) nella prima posizione libera della *TFAP* relativa a *P;*
  - viene inserita una nuova riga nella tabella dei file aperti di sistema; se il file non è già in uso, l'i-node del file aperto viene copiato dalla i-list (in memoria secondaria) alla *TFA.*

- Considerando il comportamento della **fork()**, è da notare che, poiché ogni nuovo processo eredita dal padre una copia della ***User Structure****,* esso eredita quindi anche la tabella dei file aperti dal processo padre: pertanto se il padre ha aperto un file prima della chiamata **fork()**, il figlio ne eredita la riga corrispondente nella *TFAP,* e pertanto condivide lo stesso elemento della *TFAS* con il padre. Questa situazione, illustrata in figura, è l'unico caso in cui due processi che accedono allo stesso file, ne condividono anche l'I/O pointer.


User structure del padre
fd
Tabella dei file aperti del processo
fork
fd
Tabella dei file aperti del processo
User structure del figlio
Struttura dati globali
Tabella dei file attivi
i-node
I/O pointer
Tabella dei file aperti di sistema
+
disco

System call per i file

- La SC **open** consente di creare un file o aprirlo, se già esiste. La sintassi è:

  int open (char *nomefile, int modo, [int protezione])

  - **Nomefile** indica il nome del file
  - **Modo** specifica la modalità di accesso:
    - **O_RDONLY (lettura)**
    - **O_WRONLY (scrittura)**
    - **O_CREAT (creazione)**
    - **O_APPEND (scrittura in aggiunta)**
  - **Protezione** specifica la protezione del file. Il parametro può essere espresso in formato ottale: quattro cifre ottali da 0 a 7 che indicano rispettivamente: (1) i valori per i bit SUID SGID Sticky; (2) permessi utente 3) permessi del gruppo e (4) permessi per tutti gli altri utenti.

- La open ritorna un intero che rappresenta il descrittore del file associato al file aperto. Nel caso di errore ritorna il valore -1. Ad esempio:

  fd = open(“prova”,O_CREAT|O_WRITE,0755)

  apre il file **prova** in scrittura (se non esiste il file viene creato per via della presenza di O_CREAT). I diritti di accesso al file sono **rwx** per il proprietario e **r-x** per gruppo e tutti gli altri.

- La SC **close** chiude la sessione di accesso al file. La sintassi è:

  int close (int fd)

  l’argomento di **close** è il descrittore del file ottenuto dalla open.

- L’accesso al file avviene mediante le funzioni **read** (lettura) e **write** (scrittura):

  int read (int fd, char *buffer, int n)

  int write (int fd, char *buffer, int n)

- Le funzioni ritornano rispettivamente il numero di byte letti e scritti.
- Per l’accesso diretto (random) ai file si può usare la funzione lseek.

  int lseek (int *fd*, int *offset*, int *da_dove*);

  La funzione **lseek** riposiziona il puntatore del file avente descrittore *fd* alla posizione *offset* secondo il valore del parametro *da_dove,* il quale può assumere i seguenti valori:

SEEK_SET, il file pointer è impostato sulla posizione di *offset byte a partire dall'inizio del file;*

SEEK_CUR, il puntatore del file è impostato a offset byte a partire dalla sua posizione corrente;

SEEK_END, Il puntatore è impostato alla posizione *offset* byte a partire dalla sua dimensione (fine del file).

Esempi di SC dei file

- Il primo esempio mostra la realizzazione di un comando **cp** (copia di file).
- Nel secondo esempio, si illustra il caso in cui due processi, padre e figlio, che accedono allo stesso file, ne condividono anche l'I/O pointer.
- Nel terzo esempio si mostra come realizzare l'accesso diretto mediante la lseek.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#define DIMBUF 1024
#define PERMESSI 0755

int main (int argc, char **argv){
  int stato,fin, fout,n;
  char buffer[DIMBUF];
  if (argc != 3){
    printf("errore \n");
    exit(1);
  }
  if ((fin=open(argv[1],O_RDONLY))<0){
     printf ("errore lettura file");
     exit(1);
  }
  if ((fout=open(argv[2],O_CREAT|O_WRONLY,PERMESSI))<0){
     printf ("errore scrittura file");
     exit(1);
  }
```

```
while ((n=read(fin,buffer,DIMBUF))>0)
    if (write(fout,buffer,n)<n){
      close(fin);
      close (fout);
      exit(1);
    }
  close(fin);
  close(fout);
  exit(0);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#define N 10
int main(){
  int i,fd1,fd2,pid;
  fd1=open("pub.txt",O_CREAT|O_WRONLY,0777);
  pid=fork();
  if (pid==0) {
    fd2=open("priv.txt",O_CREAT|O_WRONLY,0777);
    for (i=0;i<N;i++) {
      write (fd1,"figlio",6);
      usleep(100);
      write (fd2,"figlio",6);
    }
    close (fd2);
  }
  else if (pid>0) {
    for (i=0;i<N;i++) {
      write (fd1,"padre",5);
      usleep(100);
    }
    close(fd1);  }}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
struct Persona {
   int id;
   char cognome[40];
   char nome[20];
   char tel[16];
} persona;
int main(){
   int fd,size,i=0,n;
   size=sizeof(persona);
   fd=open("./persone.db",O_CREAT|O_RDWR);
   persona.id=10;
   strcpy(persona.cognome,"Perin");
   strcpy(persona.nome,"Stefania");
   strcpy(persona.tel,"06102030");
   lseek(fd,size*i,SEEK_SET);
   write (fd,&persona,size);
   i++;
```

```
    persona.id=11;
    strcpy(persona.cognome,"Rossini");
    strcpy(persona.nome,"Mario");
    strcpy(persona.tel,"338112233");
    lseek(fd,size*i,SEEK_SET);
    write (fd,&persona,size);
    i++;
    persona.id=12;
    strcpy(persona.cognome,"White");
    strcpy(persona.nome,"Roger");
    strcpy(persona.tel,"021112233");
    lseek(fd,size*i,SEEK_SET);
    write (fd,&persona,size);
    close(fd);
    fd=open("./persone.db",O_RDONLY);
    i=1; // posiziona la lettura al primo record
    lseek(fd,size*i,SEEK_SET);
    read(fd,&persona,size);
    printf("cognome %s\n",persona.cognome);
  close(fd);
}
```

Protezione di file e directory

- Fondamentale requisito per i SO multiutente: consente di stabilire regole di accesso ai file e directory.
- Le soluzioni più comuni per l'accesso ai file si basano sui concetti di ***risorsa*** e ***dominio di protezione.***
- Le **risorse** sono gli oggetti da proteggere (file, directory, dispositivi …).
- I **domini di protezione** sono definiti come un insieme di coppie ***<risorsa, diritti>*** e generalmente sono associati ad un utente. I diritti specificano il tipo di operazione che è possibile eseguire per la risorsa. Ad esempio per i file i diritti specificano se si ha diritto di lettura, scrittura o esecuzione.
- Il sistema operativo mantiene in opportune strutture dati le risorse e i domini. Concettualmente si può pensare ad una **matrice di protezione**.

Utente\risorsa	file1	file2	file3	dir1	stampante1
Lino	rw	r	rw	rw	w
Mario	r	rwx		r	w
Eva				rw	

Matrice di protezione

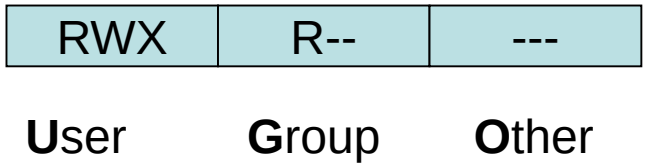
- La matrice non è una struttura dati adatta in pratica per realizzare la corrispondenza tra risorsa e diritti, per via dell'elevato numero di utenti e soprattutto di risorse che sono presenti in un sistema. Esse sarebbero matrici sparse e di enormi dimensioni.
- Le due soluzioni più comuni consistono in:
  - **Liste di controllo degli accessi (Access Control List – ACL)**
  - **Liste di capacità (Capability List o C-list)**

- Una **ACL** esprime la politica di protezione associata ad una risorsa, rappresenta quindi una **colonna della matrice di protezione**: per ogni risorsa **R** si elencano i permessi che ciascun utente possiede per essa. Ad esempio per la **risorsa file2** l'ACL è:

Utente\risorsa	file1	file2	file3	dir1	stampante1
Lino	rw	r	rw	rw	w
Mario	r	rwx		r	w
Eva				rw	

[lino:r,mario:rwx]

- Sia Windows che Unix adottano tecniche di protezione basate su **ACL**.
- Unix usa una tecnica di ACL più semplice poiché per ogni risorsa si specificano tre classi in cui gli utenti sono raggruppati: ***proprietario***, ***gruppo*** e tutti gli ***altri*** utenti. Pertanto, ogni ACL occupa solo 9 bit:



- Una **C-List**, invece corrisponde ad una riga della matrice di protezione. Quindi, il sistema mantiene una lista di permessi relativi alle risorse che il processo P ha diritto ad accedere. In relazione alla matrice di protezione dell'esempio si ha per un processo **P** dell'utente **lino**:

  [file1:rw, file2:r, file3:rwx, dir1:rw, stampante1:w]

Utente\risorsa	**file1**	**file2**	**file3**	**dir1**	**stampante1**
Lino	rw	r	rwx	rw	w
Mario	r	rw		r	w
Eva				rw	

Protezione

- La protezione in unix/linux avviene a livello di:
  - **autenticazione degli utenti**
  - **controllo dell'accesso alle risorse**

- L'accesso degli utenti al sistema consiste nella verifica di due parametri: **username** e **password**. Ogni utente è identificato nel sistema mediante un **nome utente** a cui corrisponde uno **user-id** (dato di tipo intero) e una **password** (dato di tipo string).
- L'utente **root** (detto super-user) è l'utente che ha i privilegi di amministratore di sistema, il suo **user-id** ha valore **0**.
- Gli utenti sono aggregati in gruppi. Ogni gruppo è identificato da un **nome logico** a cui è associato un numero intero: il **group-id**. Ogni utente deve appartenere almeno ad un gruppo, detto **gruppo di default**, e può appartenere a più gruppi.

Il file /etc/passwd

- L'elenco degli utenti è contenuto nel file di sistema **/etc/passwd**.
- Il file /etc/passwd ha un record composto da 7 campi; il carattere : (due punti) è il separatore di campo:
  1. User-name dell'utente
  2. Password crittografata
  3. User-id dell'utente
  4. Group-id a cui appartiene l'utente
  5. Descrizione dell'utente
  6. Home directory dell'utente
  7. Programma che viene eseguito al login

Esempio:

```
root:x:0:0:root:/root:/bin/bash
rossi:x:510:501:rossi lino:/home/rossi:/bin/sh
bianchi:x:511:501:bianchi eva:/home/bianchi:/bin/bash
```

Il file /etc/group

- definisce i gruppi ai quali appartengono gli utenti. Ogni gruppo è specificato da un record, avente il carattere : (due punti) come separatore di campo, che ha il seguente formato:

  nome_gruppo:passwd:GID:lista_utenti

- **nome_gruppo** indica il nome logico del gruppo;
- **password** se specificata (non è obbligatoria) indica la password (criptata) del gruppo;
- **GID (Group IDentifier)** è l'identificativo numerico del gruppo.
- **lista_utenti** elenco dei nomi logici degli utenti, appartenenti ad un diverso gruppo di default, separati da virgole.

Esempio di record di /etc/group:

```
studenti::501
tesisti::502,bianchi, rossi
```

- L'accesso al sistema avviene tramite **il login**. L'utente deve digitare al prompt username e password che vengono confrontate con i rispettivi valori presenti nel file **/etc/passwd e /etc/shadow**.
- In unix l'accesso alle risorse è basato sulle liste di controllo degli accessi **ACL** (**A**ccess **C**ontrol **L**ist): per ogni risorsa sono definiti i diritti di accesso associati agli utenti. In unix le risorse sono viste come file.

- In unix gli utenti, per quanto riguarda la sicurezza, sono distinti in tre gruppi:
  1. **Utente** (**User**) (costituito dal solo proprietario del file)
  2. **Gruppo** (**Group**) (costituito da tutti gli utenti appartenenti al gruppo del proprietario)
  3. **Altri** (**Other**) (costituito da tutti gli altri utenti del sistema che non appartengono agli altri due gruppi precedentemente descritti)
- Il **proprietario** del file stabilisce i diritti di accesso per i vari gruppi, mediante il comando **chmod**. Ad esempio il comando `chmod 755 test.c`.

- Per ogni file sono ammesse tre modalità di accesso:
  - **Lettura:** il contenuto del file può essere solo letto
  - **Scrittura:** il file può essere cancellato e il suo contenuto può essere modificato.
  - **Esecuzione**: il file può essere eseguito: in questo caso il file deve essere o un programma eseguibile o uno script. Nel caso di directory la **x** ha il significato di permettere la visualizzazione del listato della directory.
- L'ACL di ogni file è composta da 9 bit, che indica i diritti di accesso per gli utenti del sistema, visti come appartenenti ai tre gruppi: **U**ser, **G**roup e **O**ther (UGO).

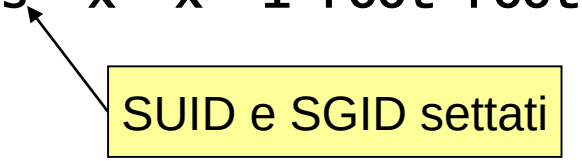
r	w	x	r	w	x	r	w	x
1	1	1	1	0	0	0	0	0
User			**G**roup			**O**ther		

- Per ognuno dei tre gruppi i diritti si esprimono con tre bit nel ordine lettura, scrittura, esecuzione (read, write, execute) i cui simboli sono rispettivamente r, w e x.
- Oltre ai 9 bit che stabiliscono i permessi di accesso, per ogni file sono specificati altri 3 bit, che hanno significato solo nel caso in cui i file sono eseguibili (programmi o script):
  - Il bit SUID (**S**et **U**ser **ID**)
  - Il bit SGID (**S**et **G**roup **ID**)
  - Il bit STIcky (**S**ave **T**ext **I**mage)
- Infatti, quando si avvia un programma il sistema inserisce nel descrittore del nuovo processo che si crea lo **user-id** e il **group-id**, appartenenti all'utente che lancia il programma.
- E' possibile cambiare il proprietario del file settando i bit **SUID** e **SGID**. Infatti se il bit SUID è posto ad 1, al processo che esegue il file sarà assegnato lo **user-id** del proprietario del file (analogamente per SGID). In tal modo l'utente che lancia il file eseguibile assume, temporaneamente, l'identità del proprietario del file.

- Un esempio d'uso di SUID e SGID si ha nel programma **passwd**.
- Il comando passwd consente a un utente di cambiare la propria password, modificando il proprio record all'interno del file /etc/passwd, che per sicurezza appartiene all'utente root (superuser) e può essere modificato solo da questo. Il file eseguibile /bin/passwd ha entrambi i bit SUID e SGID settati ad 1 consentendo così ad un qualsiasi utente di assumere l'identità di root per la durata dell'esecuzione del programma e quindi di modificare il file /etc/passwd.
- Se si esegue il list del file si può vedere che è presente il carattere **s** minuscolo al posto della **x**.

- **[frasca@localhost]$ ls -l /usr/bin/passwd**

-r-s--x--x 1 root root 16336 13 feb 2003 /usr/bin/passwd



```
[frasca@localhost]$ ls -l /etc/passwd
-rw-r--r--  1 root root 21907 Dec  1 13:36 /etc/passwd
```

- Sui vecchi sistemi unix, se un file ha lo **sticky bit** settato ad 1, consente di mantenere memorizzato il codice del processo nell'area di swap, anche dopo la sua terminazione. Questo consentiva al programma di avviarsi più rapidamente. Questa caratteristica non è più usata nei moderni sistemi. Linux, ad esempio, ignora lo sticky bit sui file. Altri sistemi unix possono usare lo sticky bit sui file per scopi particolari. In alcuni sistemi, solo il superuser può settare lo sticky bit sui file. Quando lo sticky bit è usato su una directory, i file in quella directory possono essere eliminati o rinominati solo dal proprietario o da root. Senza lo sticky bit anche gli utenti che hanno accesso in scrittura a quei file possono cancellarli e rinominarli.