Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2016-17

Pietro Frasca

Lezione 23

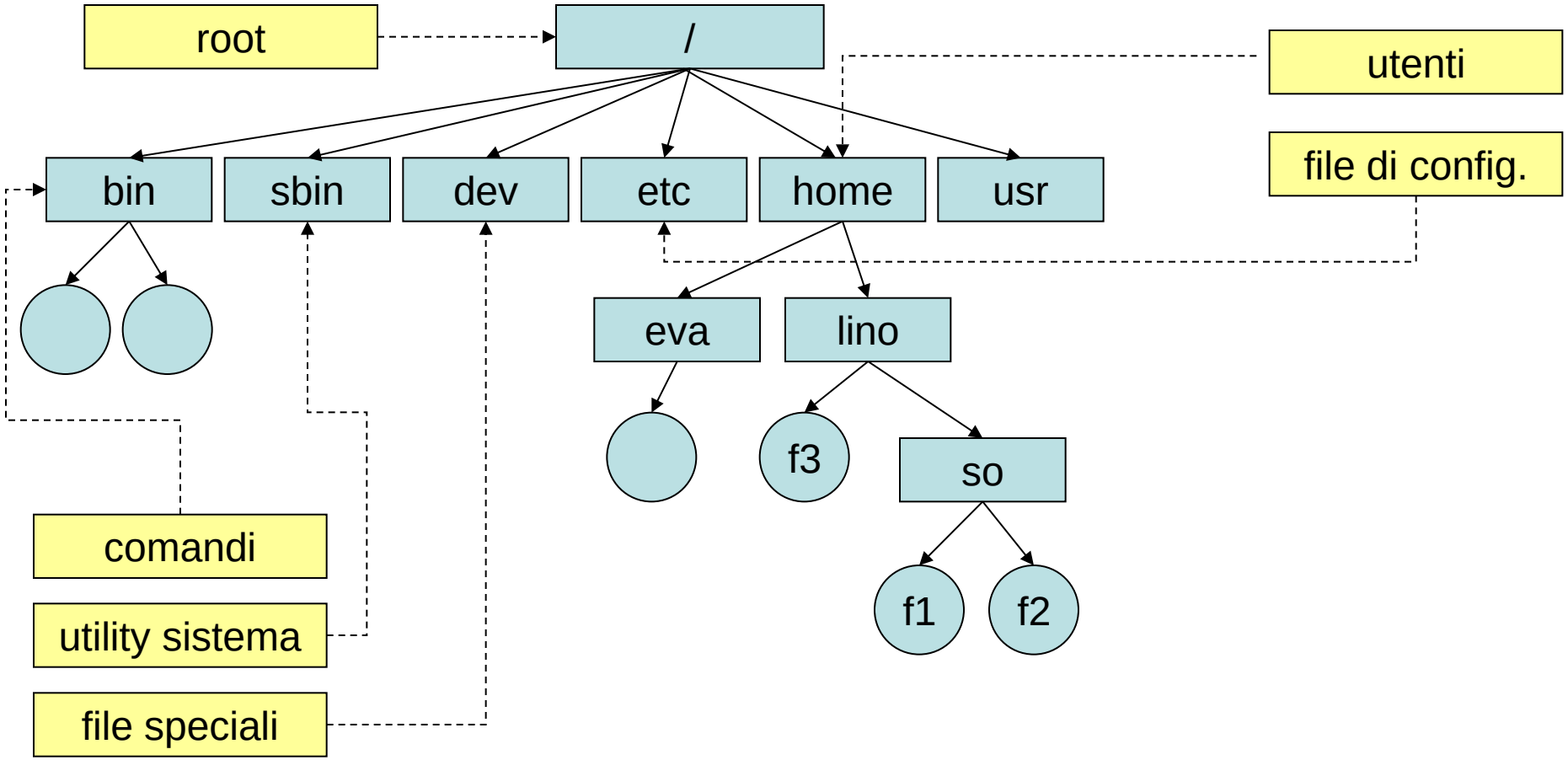
Martedì 17-01-2017

Il file system

- Il file system si basa sulle astrazioni di file e directory. Il file è l'unità logica di memorizzazione dei dati, mentre la directory è la struttura che consente di raggruppare file e anche altre directory.
- Il file system di unix considera nello stesso modo sia risorse hardware che software. In particolare esistono tre tipi di file:
  - **Ordinario**
  - **Directory**
  - **Speciale**
- Il file speciale rappresenta un dispositivo fisico, come ad esempio un disco, una stampante, una porta seriale, etc.
- In unix tutti i file speciali sono memorizzati nella directory **/dev**

Struttura logica del file system

- una tipica organizzazione logica del file system di unix è mostrata nella figura seguente.
- La directory radice è indicata con il carattere **/** (barretta o slash).
- La navigazione nel file system, cioè l'operazione per passare da una directory corrente ad un'altra si ottiene mediante il comando **cd (change directory)**.
- A ogni shell in uso è associata una directory corrente che specifica la locazione corrente nel file system.
- I nomi dei file possono essere espressi in **formato assoluto** e in **formato relativo**. Il nome assoluto del file individua il percorso che è necessario compiere per giungere ad esso a partire dalla root. Il nome relativo indica il percorso che è necessario compiere a partire dalla directory corrente per arrivare al file.



Tipica organizzazione del file system di unix

- Ad esempio, in riferimento alla figura il nome del file assoluto di **f1** è **/home/lino/so/f1** mentre quello relativo, se la directory corrente è **lino**, è **so/f1**
- La directory corrente è indicata con il carattere **.** (punto), mentre la directory padre (parent) è indicata con **..** (punto punto). Quindi il nome relativo del file **f3**, supponendo che la directory corrente sia **so**, sarà **../f3**.
- La struttura del file system di unix è a **grafo aciclico**, dato che ad un file possono essere assegnati più nomi logici (file linkati). Il comando che consente di realizzare un link è **ln**.
- Ad esempio i seguenti comandi creano rispettivamente un link software e un link hardware al file **/usr/local/bin/pro1** assegnandogli i nomi **pro2** e **pro3** nella directory corrente.

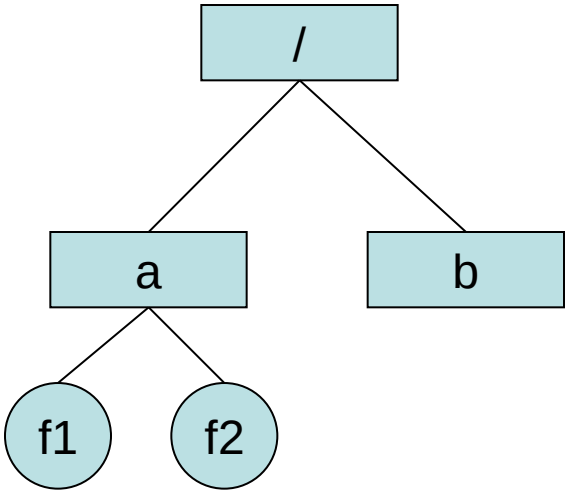
```
ln -s /usr/local/bin/pro1 pro2

ln /usr/local/bin/pro1 pro3
```

- il nome di un file può quindi non essere unico, ma ad ogni file è associato un solo descrittore (chiamato **i-node**) che è univocamente identificato da un numero intero (detto **i-number**).
- La variabile di ambiente **PATH** indica la lista di directory nella quale deve essere ricercato il programma che si vuole eseguire. Essa è una stringa formata da un sequenza di directory, ciascuna delle quali è separata dalla successiva da un carattere separatore (carattere **:**). Ad esempio

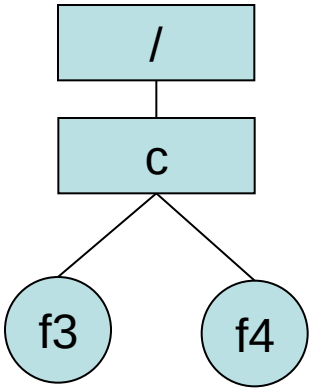
  `PATH="/bin:/usr/local/bin:."`

- Unix permette di montare un disco nel file system di un altro disco. Nell'esempio in figura il file system del disco B è montato sulla directory b del disco A. Il comando è **mount**.

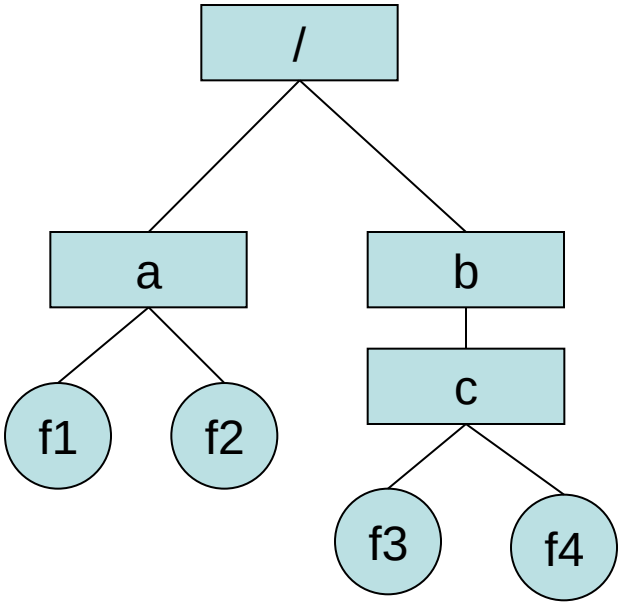
Disco rigido



CD B



Disco rigido + CD

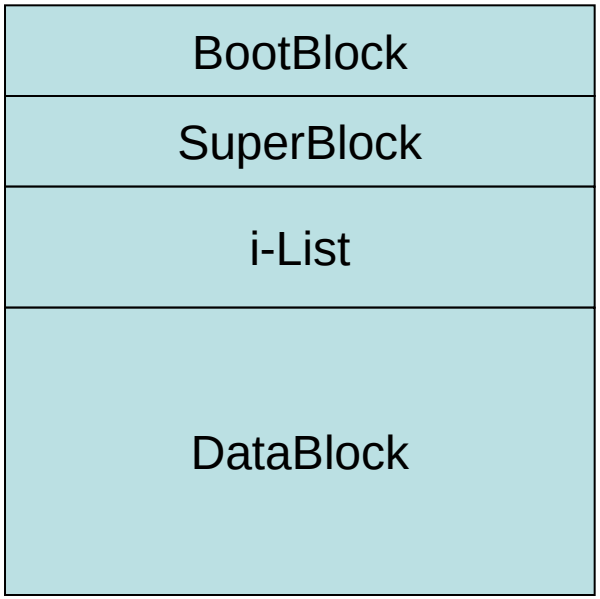


Quindi per copiare il file f3 contenuto nel CD, dopo il mount si può fare:

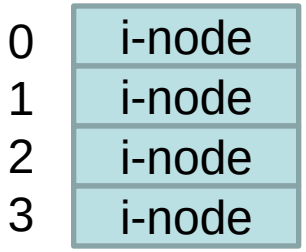
cp /b/c/f3 .

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**.



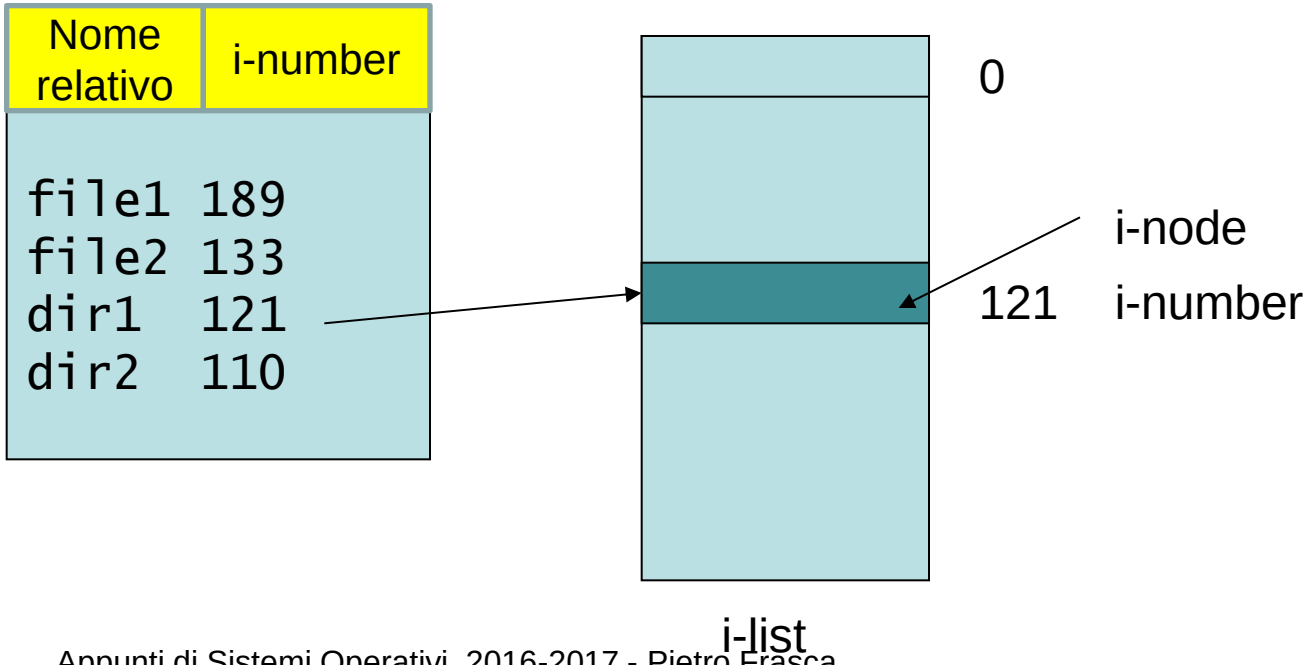
- L’area di **bootblock** ha dimensione di un blocco e contiene il programma di avvio (boot) del sistema.
- La **i-list** contiene la tabella di tutti i **descrittori** (**i-node**) dei file, directory e dispositivi contenuti nel file system. Ogni i-node è individuato mediante l’indice (i-number) della 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.

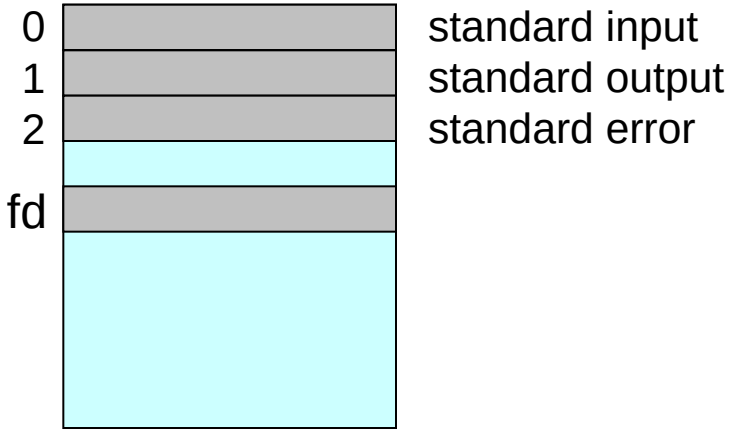
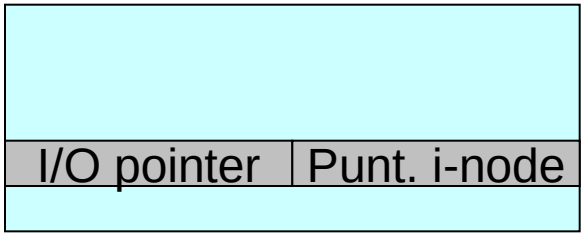


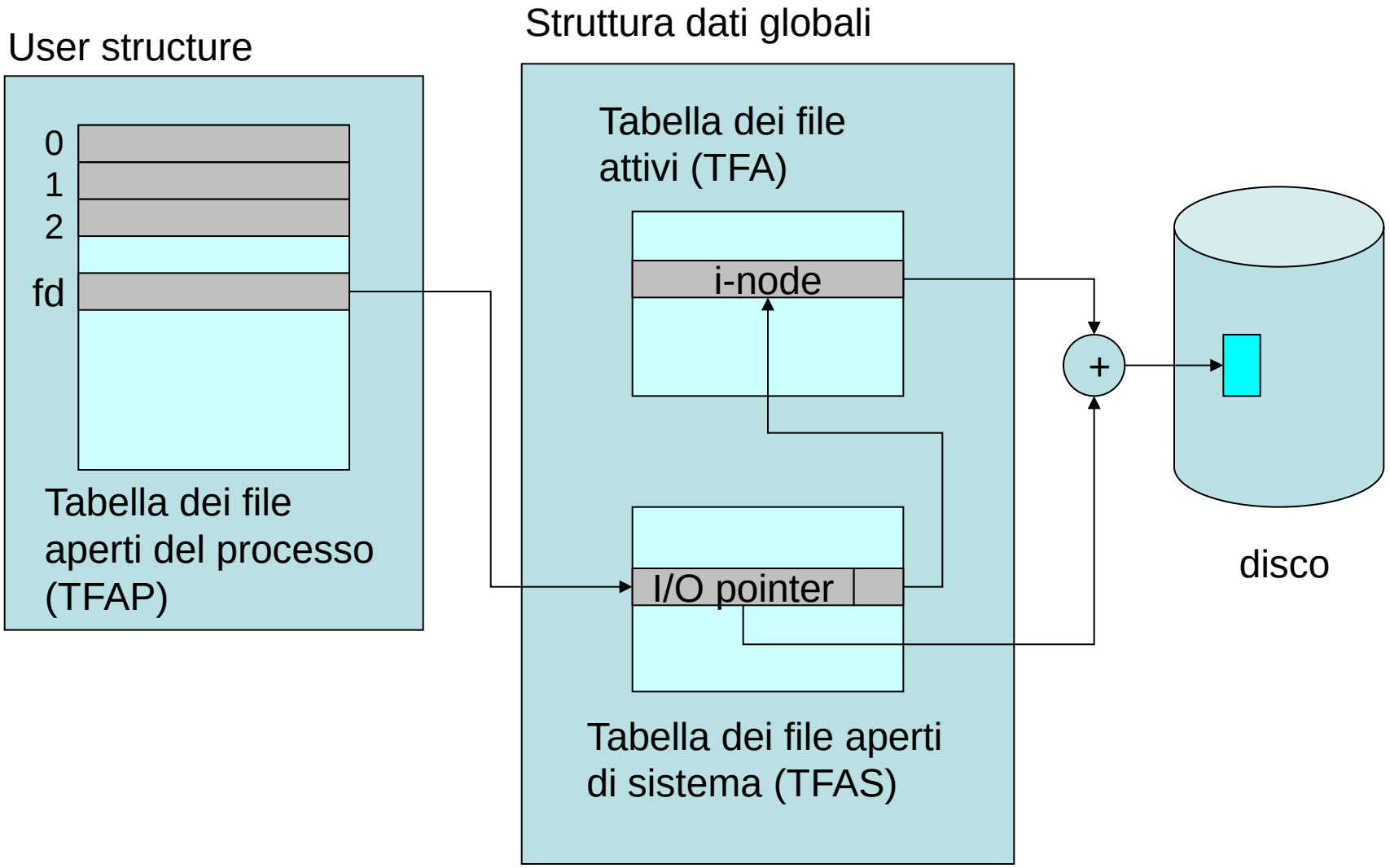
Tabella dei file aperti del processo (TFAP)

- 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

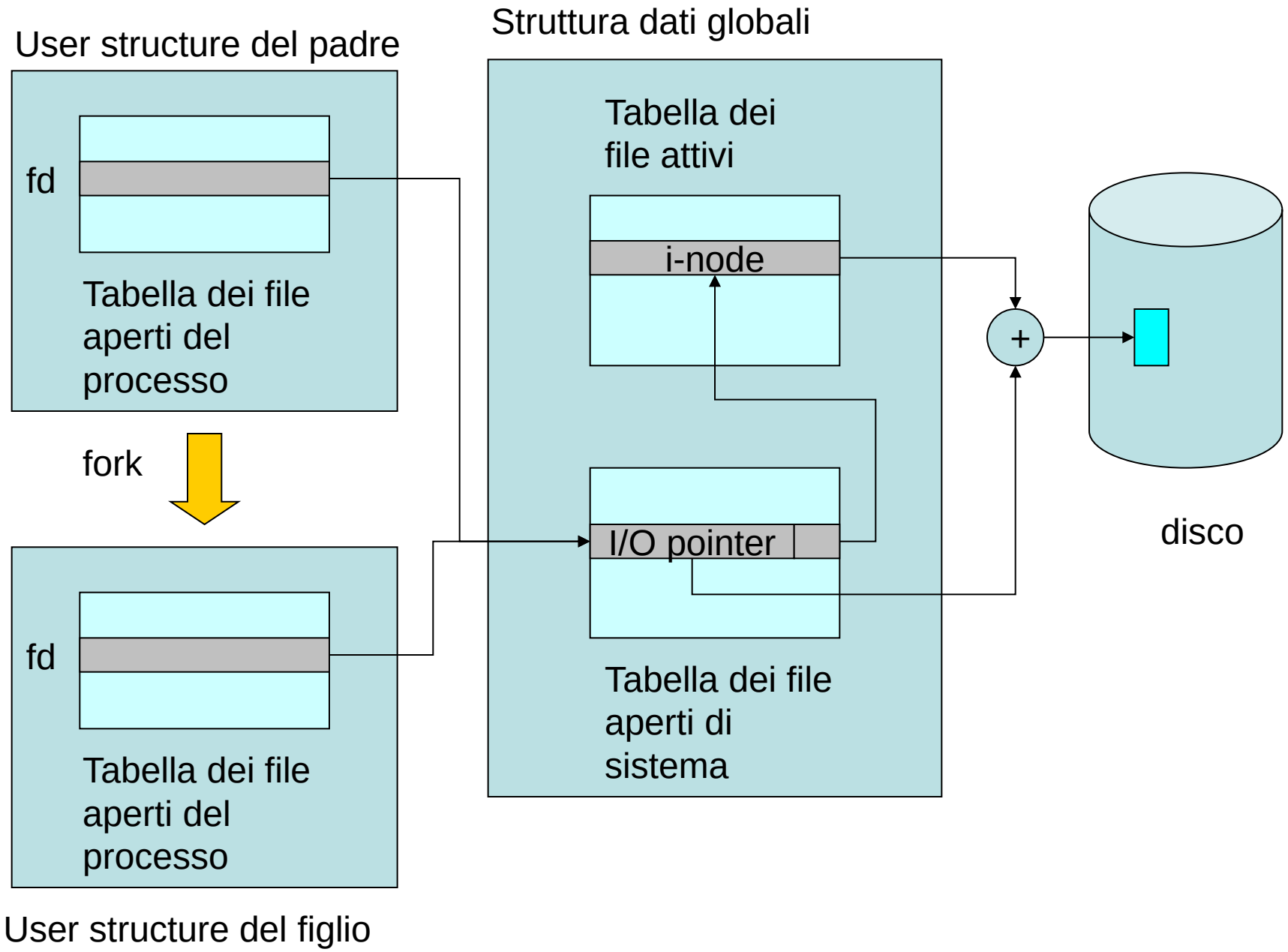
- Gli **i-node** dei file aperti sono inseriti all'interno di un'altra tabella globale: la ***Tabella dei File Attivi*** (***TFA***).
- La figura seguente mostra come le tre tabelle sono tra loro in relazione. Si può vedere come, a partire dal file descriptor **fd**, si possa ricavare l'indirizzo del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