I seguenti appunti non sono appunti ufficiali del corso di Sistemi Operativi e non sono stati revisionati da alcun docente. Nelle pagine seguenti troverete degli appunti riepilogativi degli argomenti trattati a lezione dal docente riguardo il File System e sul libro di testo consigliato. Tutte le immagini presenti sono soggette a Copyright. Buona lettura!

D. B.

File System

È la parte del sistema operativo che ***fornisce i meccanismi necessari per l'accesso e l'archiviazione*** delle informazioni in memoria secondaria.
È organizzato in livelli:

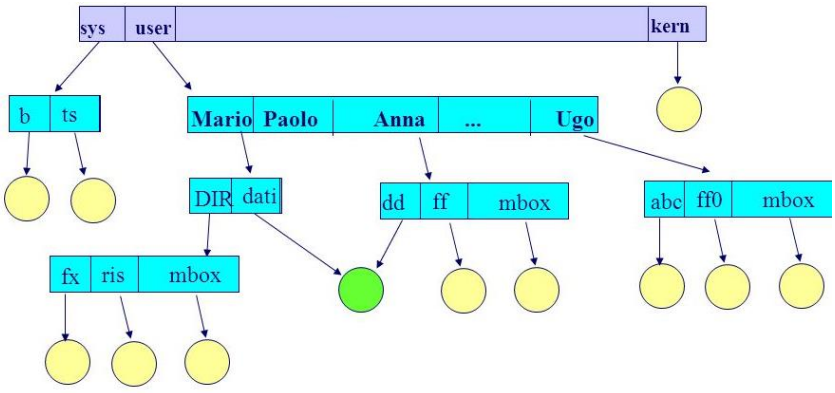
- **Struttura logica:** presenta le *primitive* che permettono l'interazione con il file system e presenta all'utente una *visione astratta delle informazioni che prescinde dalle caratteristiche del dispositivo* o dalle tecniche di allocazione
- **Accesso:** definisce e realizza i meccanismi mediante i quali è possibile eseguire le operazioni sul file System.
  Realizza le operazioni elementari di accesso al file in conformità alla modalità di allocazione e ai diritti di accesso:
  - A questo livello il file è visto come ***un insieme di record logici*** (i quali sono caratterizzati da tipo e dimensione, in Unix il record logico è il byte), i quali possono essere allocati contiguamente o in maniera casuale.
  - Si occupa dei meccanismi di ***protezione***
- **Organizzazione fisica:** a questo livello il dispositivo virtuale è visto come un array di blocchi fisici (che differiscono dai blocchi o record logici: i secondi sono molto più piccoli ed è legato al file e non al dispositivo).
- **Dispositivo virtuale:** permette di agire indipendentemente dallo specifico disco fisico. *Permette di vedere il dispositivo come un array di blocchi fisici.*

Struttura Logica

Definizione di FILE: unità logica di memorizzazione all'interno del file system. Rappresenta un contenitore di informazioni, le quali possono essere di varia natura.
Ogni file è caratterizzato da:

- Nome simbolico
- Tipo (o estensione), può essere utilizzato per indicare il tipo di informazioni contenute in un file
- Attributi come dimensione, data e ora creazione, data e ora modifica, proprietario e diritti di accesso.

Definizione di directory: è un'astrazione che consente di raggruppare più file. Può essere visto come un file "particolare" nel quale è contenuto l'elenco di file o directory al suo interno.
L'utilizzo delle directory ci permette di strutturare ***il file system attraverso una struttura ad albero***, e nel caso in cui alcune directory condividano lo stesso file (linking) l'albero diventa un grafo diretto aciclico.



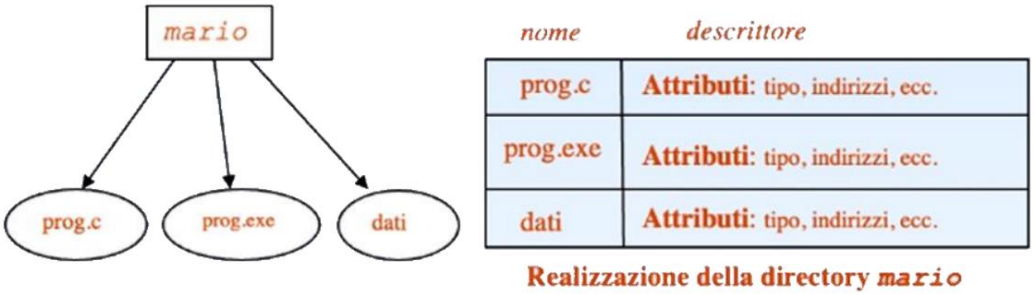
Definizione di partizione: un disco logico mappato su un disco fisico, che coincide con tutto il dispositivo o una parte dello stesso. Al momento dell'installazione del sistema operativo UNIX si definiscono le partizioni e solitamente abbiamo sempre due partizioni indipendenti: una usata come area di swap e un'altra che consiste nelle partizioni rimanenti, unificate a livello di file system (una radice unica).

Interfaccia utente: solitamente il sistema operativo mette a disposizione dei comandi per la gestione del file system ,come il listing, creazione o eliminazione di directory, attraversamento di directory, ecc.

Accesso

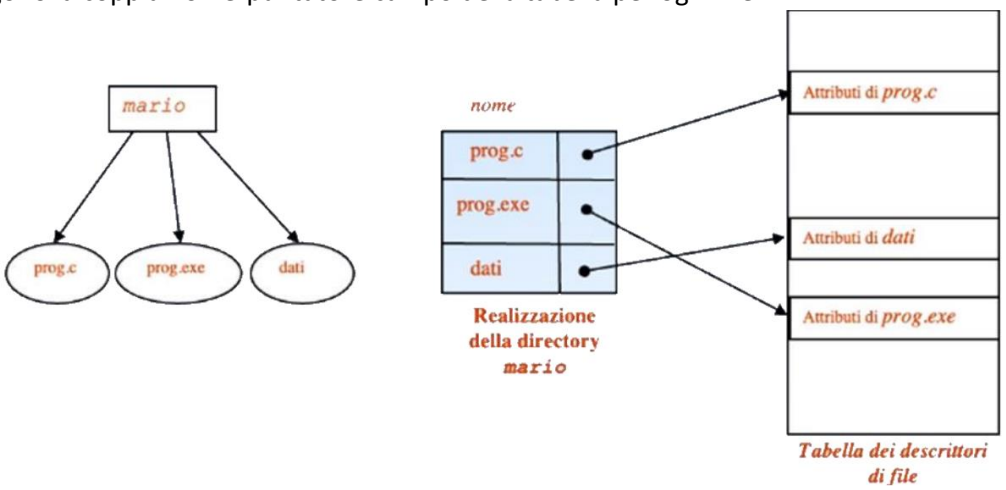
Tutte le informazioni relative ai file citate in precedenza sono mantenute in una struttura detta ***descrittore del file.***
Le directory a loro volta necessitano dei collegamenti ai descrittori dei file che contengono.
Le directory possono essere realizzate in due modi:

- ***Struttura dati tabellare (Windows):*** ogni directory contiene un elemento formato dalla coppia nome-descrittore per ogni file.



Realizzazione della directory `mario`

- Struttura dati centralizzata: è presente la tabella dei descrittori, mentre le directory mantengono la coppia nome-puntatore campo della tabella per ogni file.



Realizzazione della directory `mario`

Tabella dei descrittori di file

Nel secondo caso, l'operazione di linking è sicuramente più efficiente.

Indipendentemente dai metodi di allocazione in memoria secondaria però, l'esecuzione delle **operazioni** (lettura, scrittura, append) richiede **preventivamente l'accesso al descrittore di file**, il quale è memorizzato su disco: quindi bisognerebbe fare due accessi al disco! Per questo motivo, è tenuta in memoria una **tabella dei file aperti** nei quali è contenuto:

- **Una copia del descrittore di file**
- Informazioni relative al processo che accede al file
- Puntatore al prossimo record logico in caso di accesso sequenziale

Inoltre per velocizzare le operazioni, alcuni sistemi operano un ***memory-mapping*** del file, ovvero copiano il file in memoria centrale e le operazioni vengono fatte sulla copia, così alla apertura o chiusure del file le operazioni sono: inserimento/rimozione dell'elemento dalla tabella dei file aperti, memory-mapping/salvataggio modifiche.

Metodi di accesso: sono indipendenti dal dispositivo usato e dalla tecnica di allocazione, a questo livello i file sono visti come un insieme di record logici.

Metodo di accesso sequenziale:
Per accedere al record logico K-esimo, è necessario accedere ai K-1 record precedenti.
Durante una sessione di accesso a un file il sistema registra la posizione del prossimo elemento della sequenza da leggere attraverso **un puntatore a file (detto I/O pointer**).
Le primitive utilizzabili sono:

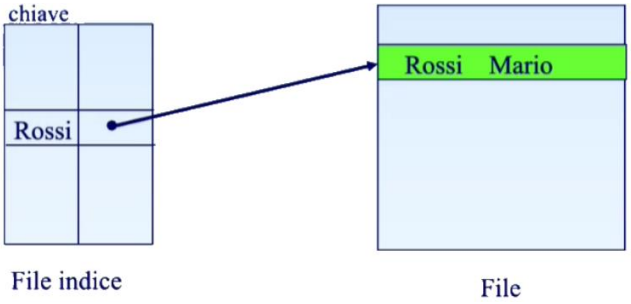
- *readnext(f, &V)*
  Lettura del prossimo record logico della sequenza.
  Assegno il valore del prossimo record logico del file f alla variabile V
  Posiziono il puntatore al file f sull'elemento successivo a quello letto.
- *writenext(f, V)*
  Scrittura del prossimo record logico della sequenza.
  Scrive nel prossimo record logico del file f il valore della variabile V.
  Posiziona il puntatore al file f sull'elemento successivo a quello scritto

Metodo di accesso diretto:
Si può accedere a un qualunque record logico del file ed inoltre permette di non dover memorizzare IO-pointer.

- *readd(f, i, &V)*
  Lettura del record logico Ri dal file f, il valore letto viene assegnato alla variabile V
- *writed(f, i, V)*
  Scrittura del valore della variabile V nel record logico Ri del file f

Può essere conveniente nel caso in cui si voglia accedere o si vogliano modificare pochi record all'interno di un file di grandi dimensioni. (L'efficienza però dipende anche dal metodo di allocazione)

Metodo di accesso a indice:
Anche qui decade la sequenzialità permettendo l'accesso ad un record qualsiasi del file, ma viene utilizzate una "hash-table": la tabella degli indici, alla quale viene associata una chiave ad un record all'interno del file.

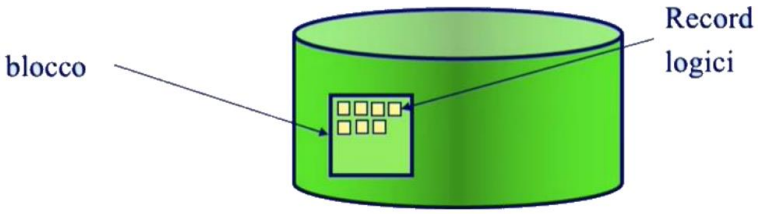


Mantenere la tabella degli indici ha però un costo: per ogni file va memorizzata la tabella degli indici e vi si deve accedere in memoria secondaria (o facendo il memory mapping in memoria centrale).

- *readk(f, key, &V)*
  Lettura del record logico con chiave uguale a key dal file f, il valore letto viene assegnato alla variabile V.
- *writek(f, key, V)*
  Scrittura del valore della variabile V nel record logico del file f avente chiave key

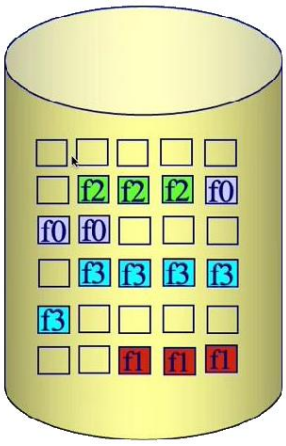
Tecniche di allocazione fisica dei file

Innanzitutto, distinguiamo il record logico (unità di accesso al file) dal record fisico (o blocco, unità di accesso al disco fisico): la prima è di dimensioni molto minori della seconda.



Metodo di allocazione contigua:

Questo metodo prevede che ogni file occupi un **insieme di blocchi fisici contigui**.



Vantaggio: rende particolarmente efficiente la realizzazione del metodo di accesso sequenziale ed anche l'accesso diretto (infatti basta memorizzare l'indirizzo del primo blocco per poi fare (B + i)/NumRecordPerBlocco).

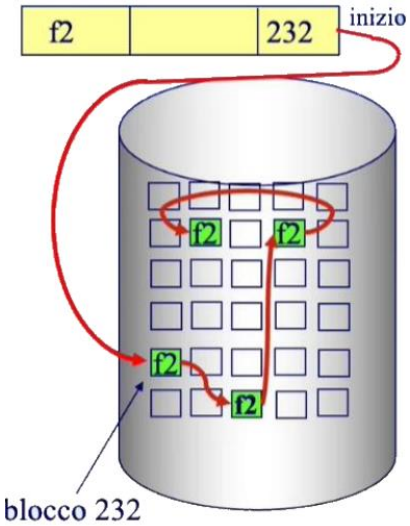
Svantaggio: legato alla gestione dello spazio disponibile, infatti la tecnica è del tutto simile a quella utilizzata per la gestione della memoria a partizioni variabili.

Per ogni nuova allocazione è *necessario trovare su disco un insieme di blocchi contigui* di dimensione sufficiente a contenere il file. Inoltre, poiché solitamente i blocchi che si trovano hanno dimensione maggiore, il disco si va a ***frammentare,*** per cui è necessario periodicamente fare un'operazione di *compattamento* (ma questa operazione è molto onerosa in termini di overhead! Basti pensare al fatto che quando si fa il compattamento della memoria centrale ci si appoggia al disco fisico, che in questo vogliamo compattare).

Infine è necessario mantenere una struttura dati in memoria per sapere quali sono i blocchi disponibili: la gestione dell'assegnamento dei blocchi disponibili può avvenire secondo tre criteri:

- **Best Fit:** si ordinano i frammenti in ordine *crescente* per dimensione e quando si vuole allocare un nuovo file, si assegna la prima partizione nel quale il file è contenuto. (Va a creare delle partizioni a volte troppo piccole)
- **First Fit:** si ordinano i blocchi in ordine *crescente* per indirizzo del blocco e quando si vuole allocare un nuovo file, si assegna la prima partizione nel quale il file è contenuto. Migliore la gestione del rilascio dei blocchi e dell'assegnamento.
- **Worst Fit: :** si ordinano i blocchi in ordine *decrescente* per dimensione e quando si vuole allocare un nuovo file, si assegna la prima partizione nel quale il file è contenuto, quindi a quello di dimensione massima, così da favorire la creazione di frammenti di dimensioni maggiori ed eventuale allocazione di altri file nella stessa zona.

Metodo di allocazione a lista concatenata:



Questa tecnica memorizza ogni file in un insieme di blocchi non contigui e organizzati in una lista concatenata.

Cade la necessità di dover trovare dei blocchi contigui e non esiste più il fenomeno della frammentazione: quando un blocco si libera può essere utilizzato indipendentemente dalla zona in cui si trova!

La lista è realizzata:

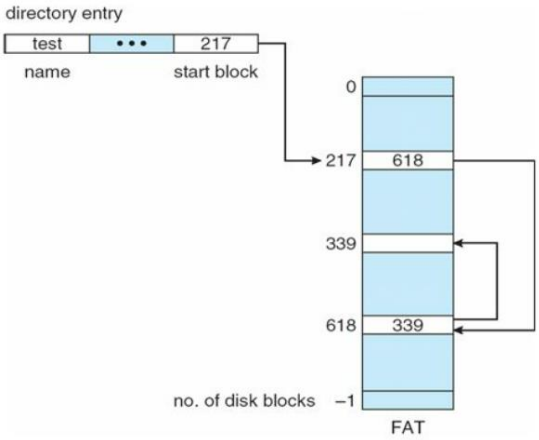
- Mantenendo un puntatore al primo blocco nel descrittore del file
- Riservando un po' di spazio nel blocco per memorizzare il puntatore al successivo

Vantaggio: come detto decade il fenomeno della frammentazione e si realizza in maniera efficiente il metodo di accesso sequenziale.

Svantaggio: è poco efficiente realizzare il metodo di accesso diretto (necessita di i/NumRecordPerBlocco accessi in memoria).

Inoltre la *robustezza in caso di perdita di informazioni è scarsa*: se per un guasto hardware o un errore

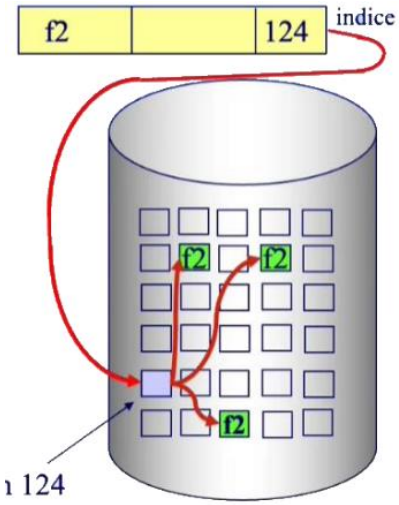
software il valore di un puntatore presente in un blocco si corrompe, è impossibile accedere a tutti i blocchi successivi dello stesso file!



Per arginare il problema si può realizzare una lista a doppio collegamento (ma occupa più spazio in memoria, sia nel descrittore del file in quanto bisogna memorizzare anche un puntatore alla coda del file, sia nel riservare spazio nei blocchi) e non si è comunque risolto al 100%.

Un altro metodo per arginare il problema è l'utilizzo della **FAT (File Allocation Table)**: al posto della lista a doppio collegamento, viene salvata in una posizione prefissata sul disco virtuale questa tabella, che contiene per ogni blocco il riferimento al blocco da lui puntato. Per velocizzare le operazioni in caso di perdita dei dati la FAT può essere copiata in RAM e/o in cache.

Metodo di allocazione ad indice:



Anche questo metodo si basa sull'allocazione di blocchi non contigui in memoria, ma invece che creare una lista utilizza un blocco indice nel quale vengono memorizzati gli indirizzi dei blocchi utilizzati.

Vantaggi: no frammentazione e accesso sequenziale o diretto efficienti.

Inoltre, nel descrittore di file basta salvare il puntatore solo al blocco indice.

Svantaggi: occupazione di un blocco intero per l'indice e poiché la dimensione del blocco è limitata, viene limitata anche la dimensione del file.

Ad esempio:

Supponiamo di avere un disco da 1TByte, e blocchi grandi 128Byte (2^8 byte per blocco).

Gli indici dei blocchi saranno in numero $2^{40} / 2^8 = 2^{32}$ blocchi.

Mi servono 32 bit per rappresentare l'indice di ogni blocco.

Quanti indici entrano nel blocco indice?

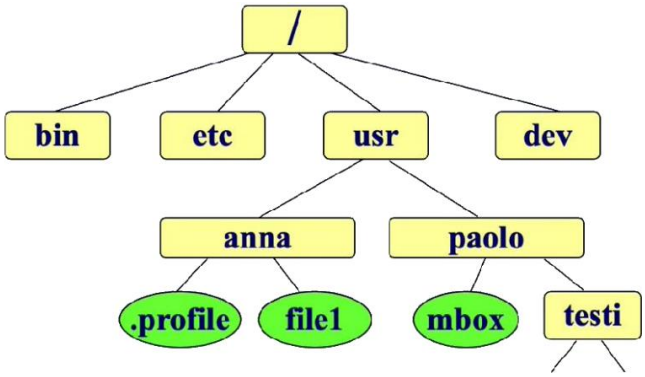
$$\frac{8 * 128Byte}{32} = \frac{2^3 \cdot 2^7}{2^5} = 2^5 \; indici \; dei \; blocchi$$

Significa che potremmo avere al massimo 32 Blocchi da 128Byte, ovvero file di dimensione max 4 GiB.

Per evitare questa pesante restrizione si utilizzano indici su più livelli.

File System in Unix

Il file system Unix è un grafo aciclico.
Una delle caratteristiche del file System Unix è l'***omogeneità***: ogni risorsa all'interno del sistema è rappresentata sotto forma di ***file***.
In particolare i file possono essere:

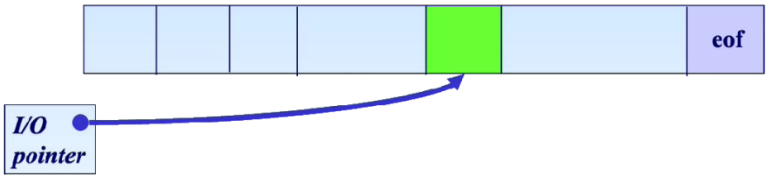


- **Ordinario**
- **Directory**, astrazione che contiene informazioni riguardo a file e directory contenute
- **Speciale**, rappresenta solitamente un dispositivo fisico. Tutti i file speciali sono contenuti nella directory */dev.*

La rappresentazione omogenea permette di avere le stesse modalità di Input/Output sia per i file che per le periferiche.

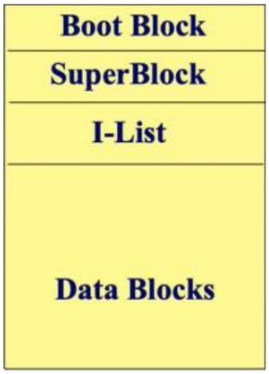
Livello di accesso

In UNIX l'accesso è sequenziale. Il file è una sequenza è visto come una sequenza di byte, il record logico dunque coincide con il byte.
Il sistema operativo mantiene un puntatore detto I/O pointer che indica la posizione corrente del record nel file aperto.



Organizzazione fisica

La superficie del disco viene suddivisa in 4 regioni:



- **Boot Block:** occupa un blocco fisico allocato ad un indirizzo prefissato. Contiene il programma di inizializzazione del sistema, eseguito in fase di bootstrap.
- **Super Block:** occupa un blocco fisico e descrive l'allocazione del file system. In particolare contiene:
  - Il limite delle 4 regioni
  - Il puntatore alla lista dei blocchi liberi
  - Il puntatore alla lista degli i-node liberi
- **I-List:** Contiene il vettore di tutti i descrittori dei file ***(i-node),*** le directory e i dispositivi presenti nel file system.
  Ogni I-node è identificato da un indice (i-number) che lo identifica univocamente.
- **Data Blocks:** la regione più grande e che contiene effetivamente i file.

Il descrittore di file i-node:

Contiene:

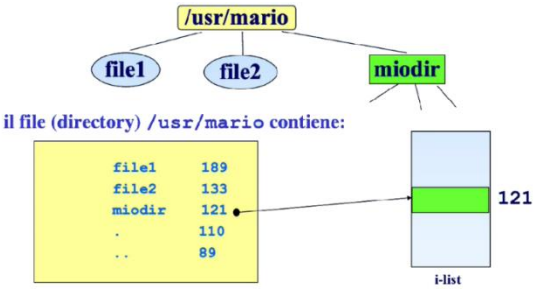
- il tipo di file (ordinario, directory o speciale)
- utente e gruppo proprietario
- i 12 bit di protezione (9 per i permessi utente proprietario, gruppo proprietario, altri e 3 SUID, SGID, STIcky)
- data di creazione e modifica
- dimensione (numero di blocchi occupati dal file)
- **vettore di indirizzamento:** in particolare il metodo di allocazione usato in Unix è un metodo a indice su più livelli.
  Il vettore può contenere da 13 a 15 elementi:
  - i primi 10 elementi contengono riferimenti diretti ai blocchi utilizzati per memorizzare i dati
  - l'11 contiene il riferimento ad un blocco indice che a sua volta contiene i riferimenti ai blocchi (indirizzamento diretto di primo livello)
  - il 12esimo contiene riferimento ad un blocco indice che a sua volta contiene riferimenti ad altri blocchi indice, ognuno dei quali contiene riferimenti ai blocchi dei dati (indirizzamento indiretto di secondo livello)
  - dal 13esimo in poi aumenta il livello di indirizzamento indiretto

  Questo ovviamente limita il limite della dimensione del file, però porta ad un overhead spaziale non indifferente.

La directory

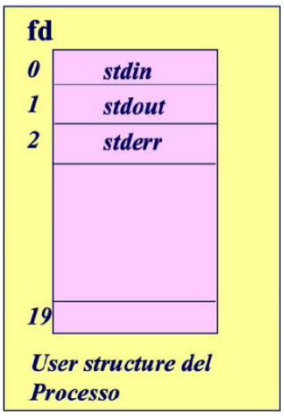
In Unix è una tabella dove ogni elemento è caratterizzato da due campi:

- il nome simbolico
- l'i-Number, così da permettere di raggiungere l'i-Node nell'i-List



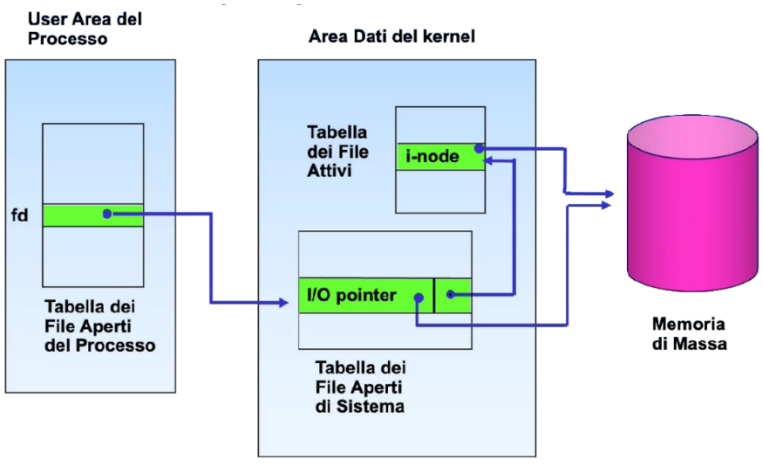
Strutture dati del kernel per l'accesso ai file

La questione principale è come gestire l'iNode di un file aperto poiché possono essere tanti in un sistema multi-programmato?
A ***ogni processo*** è associata ***una tabella dei file aperti***, dove ogni elemento è **un file descriptor**. La tabella è raggiungibile dalla **user structure di un processo (dal kernel).**
Al momento dell'inizializzazione del processo la tabella contiene automaticamente 3 file descriptor: stdin, stdout, stderr.



Nell'area dati del kernel troviamo anche altre due strutture globali:

- **tabella dei file attivi,** (una sorta di cache, per ogni file aperto è presente una copia dell'i-node)
- **tabella dei file aperti di sistema,** ogni elemento di questa tabella è composto da IO-Pointer e collegamento a I-node



Grazie all'IO-pointer e all'i-node che contiene il vettore di indirizzamento è possibile calcolare l'indirizzo fisico del prossimo byte a cui accedere in memoria secondaria.
Cosa succede quando si fa un'operazione di apertura?

1) Viene inserito un nuovo elemento nella prima posizione libera della tabella dei file aperti di processo
2) Viene inserito un nuovo elemento nella tabella dei file aperti di sistema
3) Se non è già presente viene copiata l'i-node (dalla memoria secondaria, i-list) alla tabella dei file attivi

Ricordiamo che in caso di fork da parte di un processo viene copiata la tabella dei file aperti del processo (unico caso in cui due processi condividono l'IO-Pointer).

Cosa succede quando si fa un'operazione di chiusura?

1) Elimina l'elemento dalla tabella dei File aperti del processo e (eventualmente) di sistema.
2) Verifica se l'iNode posto nella tabella dei file attivi dovrà essere rimosso o meno (verifico il numero di istante aperte dello stesso file, abbiamo un contatore memorizzato nel descrittore).

System Call per accesso ad un file

1) int open(char filename[], int mode, int prot)
    a. filename → nome simbolico del file da aprire
    b. va specificato il modo di accesso se O_WRONLY, O_CREAT, ecc
    c. nel caso in cui il file si stia creando si possono specificare i bit di protezione nel campo prot
2) int close(int fd)
3) int read(int fd, char *buf, int nbytes)
4) int write(int fd, char*buf, int bytes)

Le ultime due sono le systema call con le quali effettivamente si accede al file, restituiscono i numeri di byte letti/scritti e vengono messi/presi su/da buf. Restituiscono -1 in caso di errore.