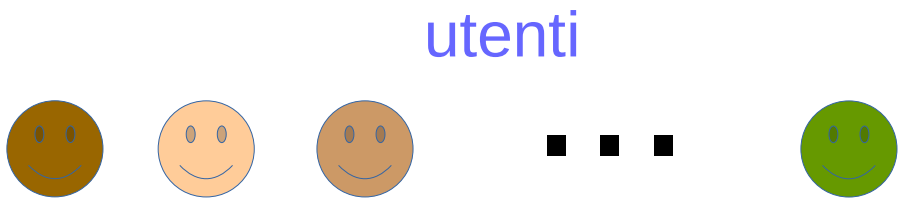
Implementazione del file system

Capitolo 11 del libro (VII ed.)

Introduzione

- Programmi e dati sono conservati in **memoria secondaria**
- la memoria secondaria ha le seguenti caratteristiche:
  - è **organizzata in blocchi** (un blocco può comprendere più settori)
  - è possibile **accedere direttamente** a qualsiasi blocco
  - è possibile leggere un blocco, modificarlo e riscriverlo esattamente **nella stessa posizione** di memoria
  - Si accede alla memoria secondaria attraverso un **FILE SYSTEM**

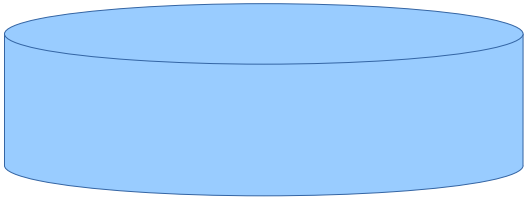
Il FS sta fra gli utenti e la memoria secondaria



fornisce una vista sulla memoria come organizzata in file e directory, definisce i metadati, fornisce operazioni permesse su file e directory

File System

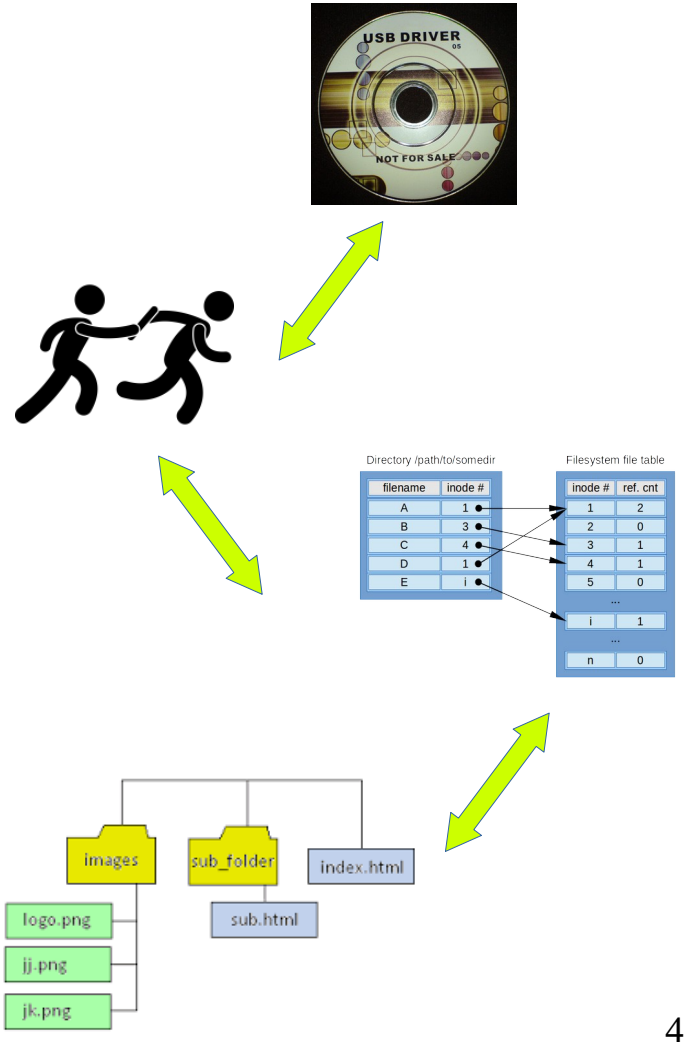
algoritmi e strutture dati che fanno corrispondere tutti gli elementi del FS visto dall'utente (FS logico) ai supporti di memoria su cui i dati sono conservati



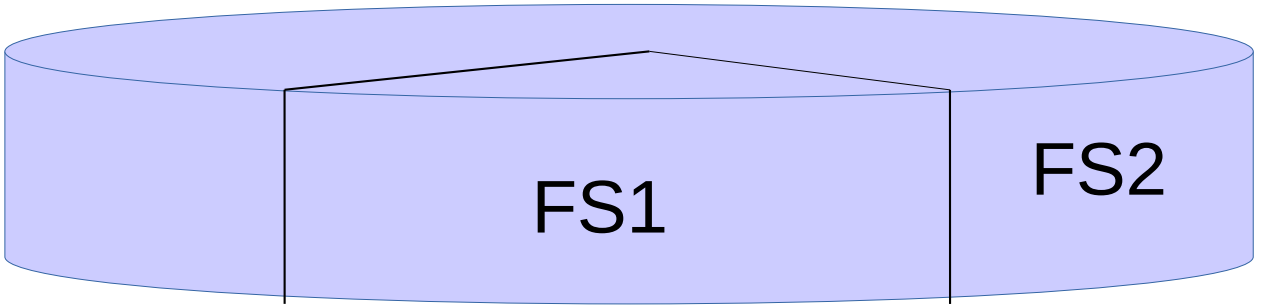
Memoria secondaria

Livelli del file system

- il FS di solito è strutturato in una sequenza di livelli, dal basso verso l'alto:
  - **driver di dispositivo**: si occupa del trasferimento dei dati da dispositivo di memoria secondaria a RAM e viceversa. È il gestore del dispositivo
  - **file system di base**: è proposto a passare comandi al driver di dispositivo
  - **modulo di organizzazione dei file**: è a conoscenza di come i file sono memorizzati su disco, è in grado di tradurre indirizzi logici in indirizzi fisici. Mantiene e gestisce anche l'informazione relativa ai blocchi liberi
  - **file system logico**: gestisce il FS a livello di metadati. Ogni file è rappresentato da un **File Control Block** (FCB)



Struttura del disco



- Un disco può essere diviso in più **partizioni**
- Ogni partizione è organizzata in **un solo FS**
- Un FS contiene il SO e/o i file degli utenti
- I diversi FS sono composti in una sola struttura (operazione **mount**)

Struttura su disco

- Un FS è una sequenza di blocchi di memoria secondaria
- Blocchi speciali:
  - **boot control block**: se il FS non contiene un SO è un blocco vuoto, se invece contiene un SO, questo blocco contiene info necessarie in fase di bootstrap

Struttura su disco

- Blocchi speciali:
  - boot control block: ...
  - **volume control block** (o **superblocco**): descrive lo stato del FS stesso, quant'è grande, quanti file può contenere, ecc. ... In particolare, contiene il numero dei blocchi appartenenti alla partizione, la loro dimensione, il numero di blocchi liberi (e loro riferimenti)

Struttura su disco

- Blocchi speciali:
  - boot control block: ...
  - volume control block (o superblocco): ...
  - **struttura delle directory**: organizza i file, è implementata in modi diversi e contiene coppie <nome file, FCB>

Struttura su disco

- Blocchi speciali:
  - boot control block: ...
  - volume control block (o superblocco): ...
  - struttura delle directory: ...
  - una **lista di FCB** (file control block): contengono i metadati relativi ai file e consentono di accedere ai loro contenuti;gli FCB sono anche detti **inode** (index-node), ciascuno identificato da un numero

INODE / FCB

- Un **FCB** (o **inode**) mantiene le informazioni relative ai file
- Gli inode sono conservati in **memoria secondaria**
- **Esempio**: in Unix un inode su disco può contenere queste informazioni

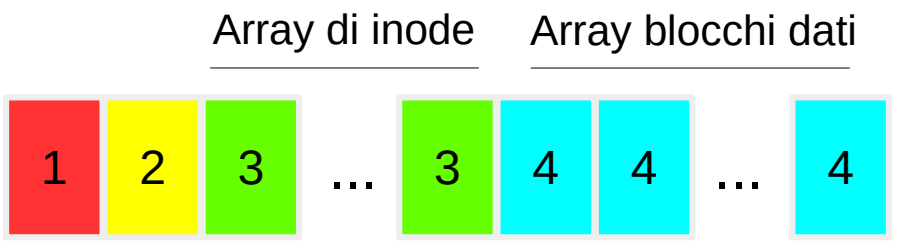
• Identità del proprietario del file	(User ID)
• Tipo del file	(regolare, directory, link, device, ...)
• Diritti di accesso	(r w x per proprietario, gruppo, altri)
• Tempi di accesso e modifica	(data/ora ultimo accesso/ultima modifica) # di link al file
• Dimensione del file numero di byte	
• Tabella per l'accesso ai dati	(indirizzi dei blocchi di mem. secondaria contenenti i dati)

NB: per semplificare l'accesso gli inode hanno **dimensione fissa** -> es. è prevista una lunghezza massima per i nomi dei file, l'indirizzamento della memoria avviene attraverso parole di lunghezze prefissata

Esempio: FS del SO MINIX

- Minix è un SO scritto in C da Andrew Tanenbaum (Vrjie University) per scopi didattici (fine anni '80)
- I nomi di file sono al più di 14 caratteri
- i blocchi possono essere riferiti da parole di 16 bit (2^{16} blocchi) - questo impone un limite importante, se i blocchi sono grandi 1KB non è possibile gestire memorie secondarie più grandi di 64 MB
- Linux è nato come tentativo di superare questi due limiti di Minix

FS di MINIX



1 Boot block: contiene un codice di bootstrap usato per avviare il SO

2 Super block: descrive lo stato del FS (quant'è grande, quanti file può contenere, ecc.)

3 Lista degli inode: ogni inode corrisponde a un file (esistente o potenziale), uno in particolare corrisponde alla radice del FS

4 Blocchi dati: ogni blocco può appartenere a uno e un solo file

NB: in Unix le **directory** sono implementate come **file speciali,** quindi alcuni inode corrispondono a directory

Strutture dati in RAM

- All'avvio di un elaboratore viene “montato il FS” e vengono caricate in RAM delle informazioni che riguardano la sua struttura
- Lo scopo è consentire l'accesso (efficiente) ai file
- Le strutture mantenute in RAM sono:
  - **mount table**: mantiene informazioni su ciascuna partizione montata, ogni operazione di mount/unmount la modifica
  - **directory**: mantiene informazioni sulle directory a cui si è avuto accesso di recente (utile in particolare per quei SO che usano descrittori diversi per i file e per le directory. Alcuni SO, e.g. Unix, invece implementano le directory come file speciali)
  - **tabella dei file aperti**: mantiene una copia arricchita dei FCB di tutti i file aperti al momento tabelle dei file dei processi: si ha una tabella di questo tipo per ogni processo; per ogni file aperto viene mantenuto un riferimento all'opportuno elemento della tabella (globale) dei file aperti

In-core inode (FCB in RAM)

Quando un file viene aperto per la prima volta il suo inode viene caricato in RAM ed arricchito da qualche informazione aggiuntiva

Esempio: in Unix un inode su ram può contenere queste informazioni

- **INODE COPIATO DA DISCO**
- **Stato dell'in-core inode**: dice se
  - l'inode è locked (il file non è disponibile)
  - la copia dell'inode in RAM è diversa da quella su disco,
  - il file è un punto di mount
  - ...
- **device number**: identificatore del FS a cui appartiene il file
- **inode number**: identificatore dell'inode nella struttura dati su disco
- **contatore** del numero di riferimenti all'inode (#utilizzi del file attuali)
- ...

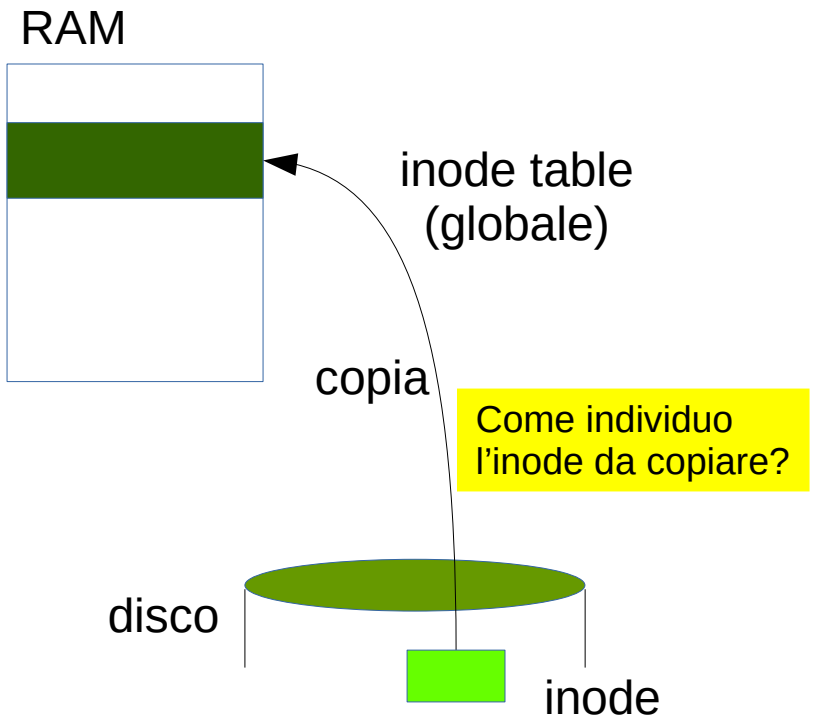
In Unix gli node caricati sono detti **incore inode** e sono conservati in una tabella globale che si chiama **inode table**

Apertura di un file

- supponiamo che un processo esegua la sys. call **open("prog.c", O_RDONLY)**

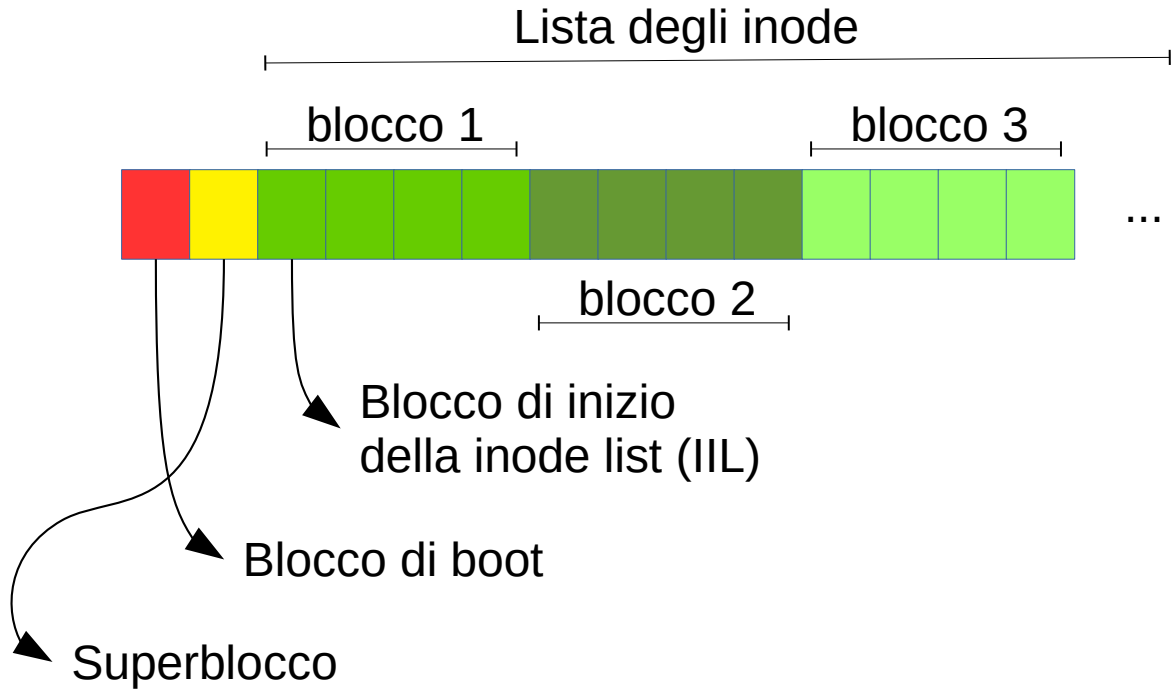
- Il kernel mantiene una **inode table** di dimensione finita
- Nell'eseguire la open: controlla se l'inode corrispondente al file è già stato caricato, **se sì userà l'incore inode trovato** altrimenti:
- Se c'è spazio, crea un **nuovo incore inode**, lo "locka" e va a copiarvi l'inode su disco corrispondente al file da usare
- Se non c'è spazio l'operazione fallisce e viene restituito un errore (così il processo richiedente non rischia di rimanere sospeso per tempi lunghi)
- *se esisteva già .... lo vedremo fra poco*

FAIL



- **Nota**: il lock (esclusivo e obbligatorio) serve per evitare inconsistenze. Si attiva all'inizio di certe system call e si disattiva al loro termine

Esempio: UNIX



La lista degli inode è contenuta in una **sequenza di blocchi**

Gli inode hanno dimensione fissa

Tutti i blocchi contengono lo stesso numero di inode
Esempio #inode-per-blocco: 4

Per accedere alla porzione di disco che memorizza un inode basta conoscerne: 1) il numero, 2) la dimensione degli inode (la stessa per tutti) e 3) quella dei blocchi

Il SO ha tutte queste informazioni

Proviamo a identificare l'indirizzo di inizio dell'inode con indice 5 (partendo a contare dall'indice 0)

Esempio: UNIX

L'inode è identificato da un indirizzo <B, D>:

B = blocco
S = displacement nel blocco

[1] Individuare il blocco che contiene l'inode
B = (inode-number / #inode-per-blocco) + IIL
/ : divisione intera

[2] Individuare il displacement dell'inode nel blocco
D = (inode-number % #inode-per-blocco) X dim-disk-inode
% : resto della divisione intera

- #inode-per-blocco: 4
- inode-number: 5
- B: IIL + 1
- D: 1 x dim-inode

Algoritmo NAMEI

- Nei lucidi precedenti abbiamo implicitamente supposto di avere a disposizione il numero dell'inode corrispondente al file di interesse
- Di solito però avremo il **nome del file** … o più in un cammino, es.

  mieiFile/Documenti/anno2020/slideSisOp.odp
- Per identificare l'inode relativo a un file devo avere a disposizione un algoritmo che è in grado di percorrere il cammino fino a identificare il numero dell'inode:

1) accedo alla directory *mieiFile*
2) cerco fra i suoi contenuti *Documenti*
3) accedo alla directory *Documenti*
4) cerco fra i suoi contenuti *anno2020*
5) accedo alla directory *anno2020*
6) cerco fra i suoi contenuti *slideSisOp.odp*
7) trovando il numero di un inode

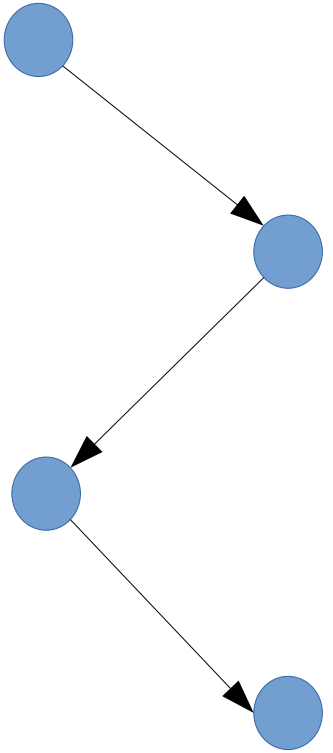
Algoritmo NAMEI

inode-number NAMEI(string cammino)

if (la prima directory del cammino è “/”)
    current = root inode
else current = inode della working directory

repeat
  el = leggi prossimo elemento dall'input;
  if (el is null) return current;
  else if (el è contenuto in current)
                    current = inode associato a el
        else return (no inode)
until (el is null);

return current;



Per verificare se el è contenuto in current occorre accedere ai blocchi dati relativi a current