Il File-System

I direttori in ambiente Linux

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Direttori

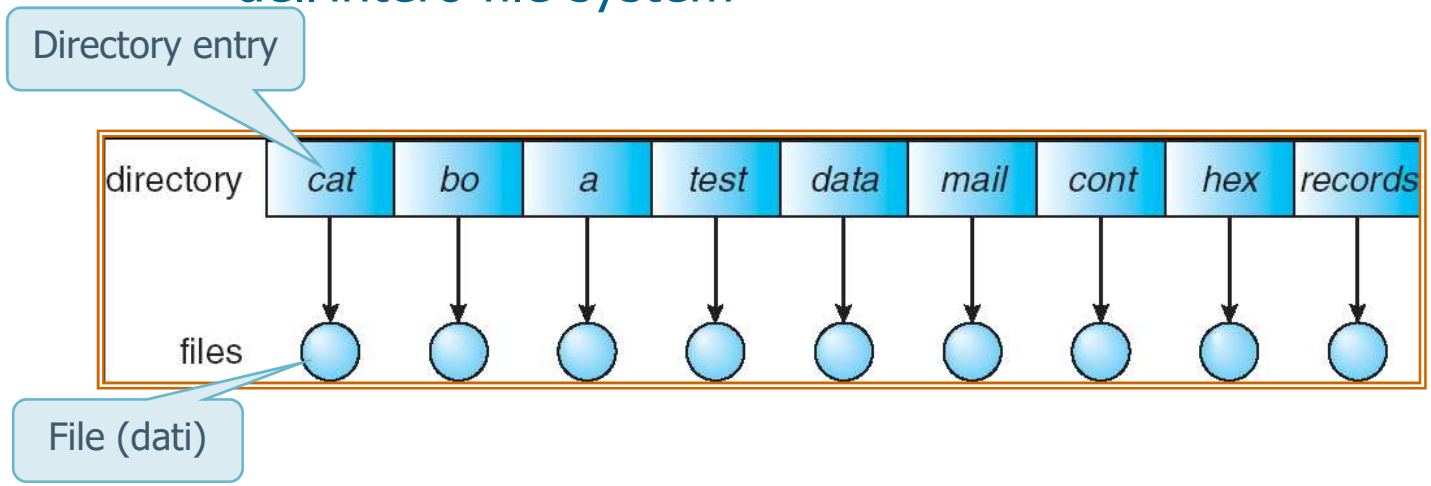
- Nessun sistema di memorizzazione contiene un unico file
- I file sono organizzati in direttori
  - Un direttorio è un nodo (di albero) o vertice (di grafo) contenente informazioni sugli elementi in esso contenuti
  - Direttori e file risiedono entrambi su memoria di massa
- Su un direttorio sono effettuabili operazioni simili a quelle dei file
  - Creazione, cancellazione, elenco del contenuto, ri-denominazione, visita, ricerca, etc.

Struttura

- La struttura di un direttorio dipende da ragioni di
  - Efficiency (efficienza)
    - Velocità nel manipolare il file system, e.g., localizzare un file
  - Naming (convenienza)
    - Semplicità per un utente di identificare i propri file
    - Evitare che lo stesso nome attribuito a più file crei problemi
  - Grouping (organizzazione)
    - Raggruppare le informazioni in base alle relative caratteristiche
      - Programmi di editing, compilatori, giochi, etc.

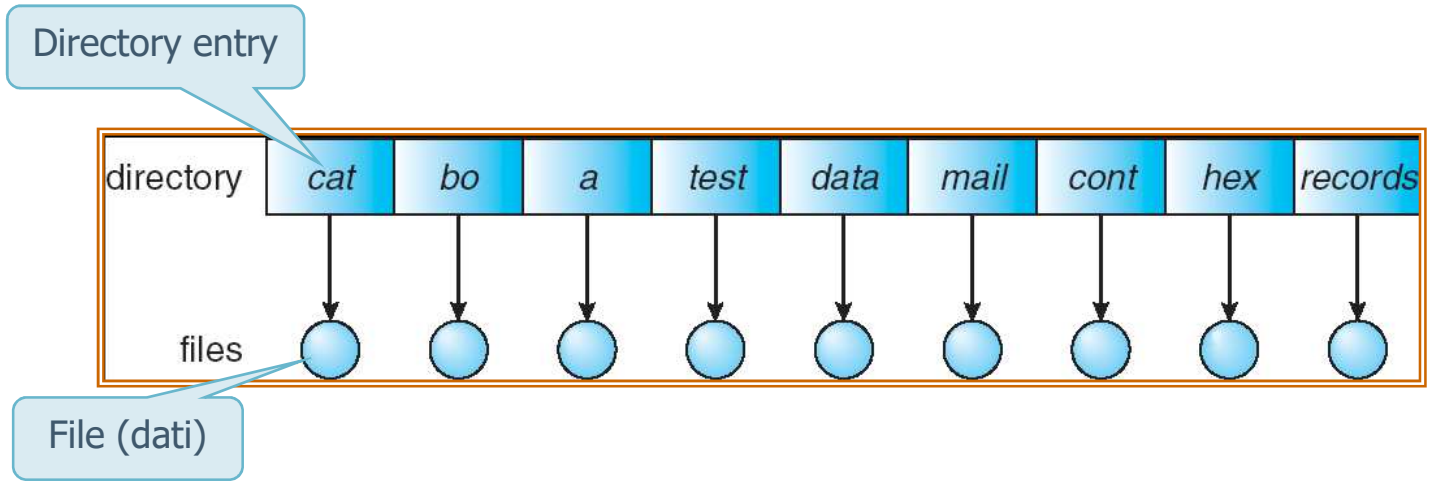
Direttori a un livello

- La struttura più semplice è quella a livello singolo
- Tutti i file del file system sono contenuti all'interno dello stesso direttorio
  - I file sono differenziati dal solo nome
  - Ciascun nome deve essere univoco all'interno dell'intero file system



Direttori a un livello

- Per ciascun file si evidenziano
  - La directory entry: individua il nome e eventualmente altre informazioni del file
  - I dati: separati dalla directory entry, sono da essa individuati tramite puntatore

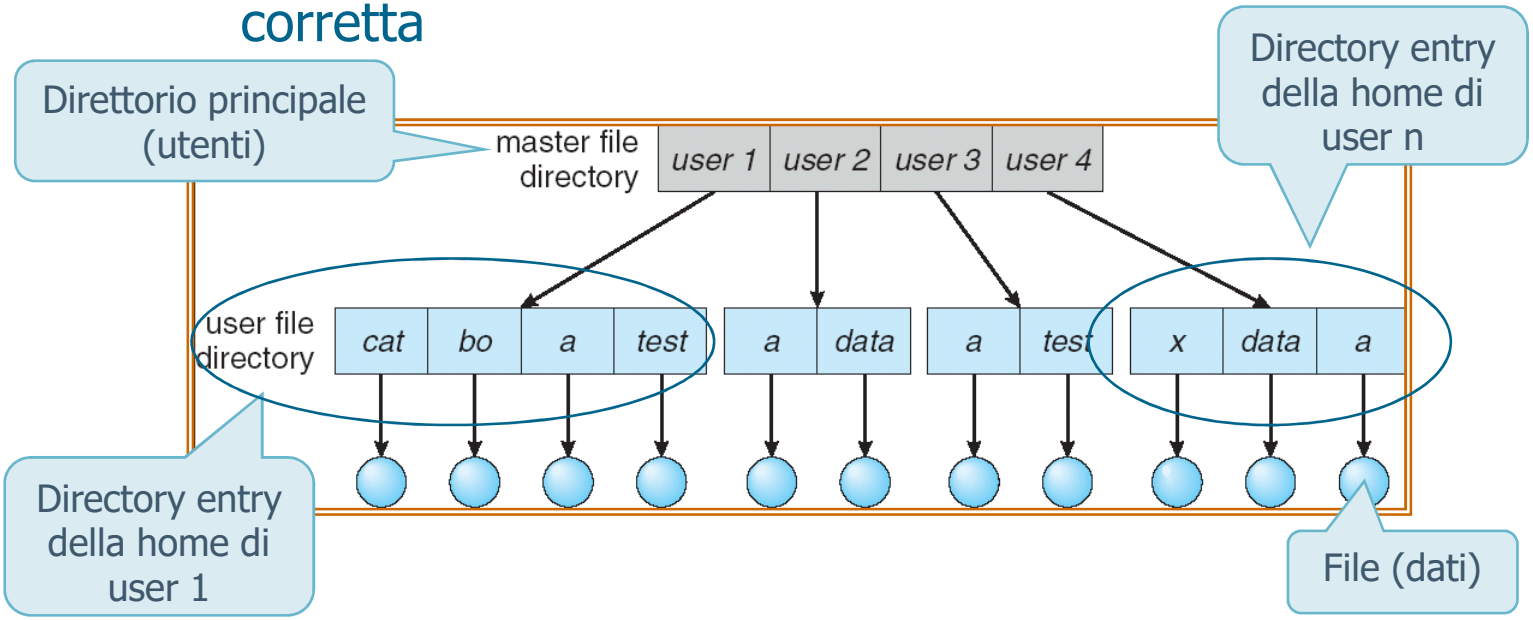


Direttori a un livello

- Prestazioni
  - Efficiency
    - Struttura facilmente comprensibile e gestibile
    - Gestione del file system semplice ed efficiente
  - Naming
    - I file devono avere nomi univoci
    - Presenta limiti evidenti all'aumentare del numero di file memorizzati
  - Grouping
    - La gestione dei file di un utente singolo è complessa
    - La gestione di utenti multipli è praticamente impossibile

Direttori a due livelli

- I file sono contenuti in un albero a due livelli
- Ogni utente può avere il proprio direttorio
  - Ogni user ha la sua home directory
  - Tutte le operazioni sono eseguite solo sulla home corretta

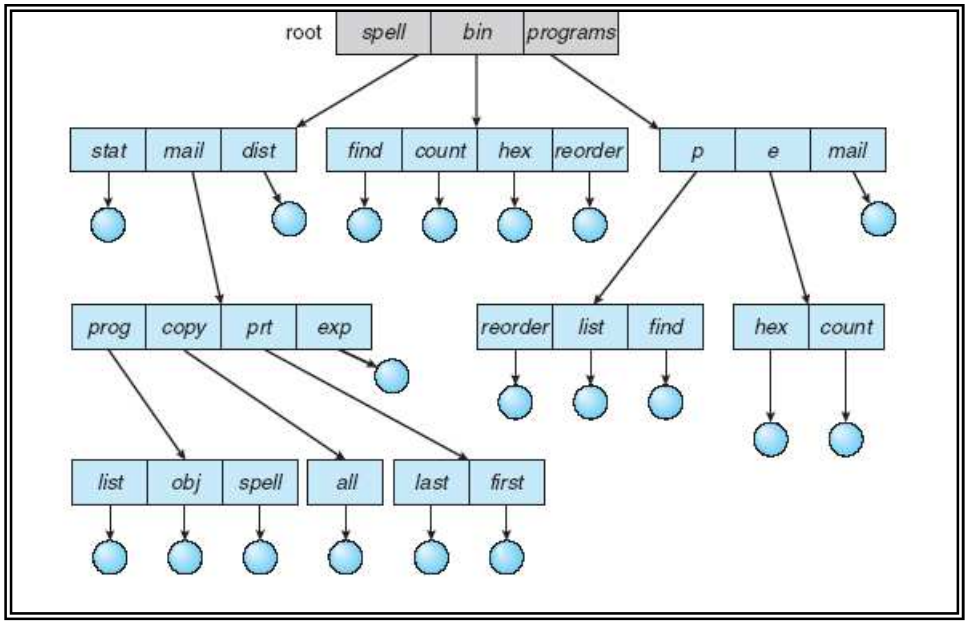


Direttori a due livelli

- Prestazioni
  - Efficiency
    - Visione del file-system "user oriented"
    - Ricerche semplificate e efficienti agendo su utenti singoli
  - Naming
    - È possibile avere file con lo stesso nome purchè appartenenti a utenti diversi
    - Occorre specificare un path-name per ogni file
  - Grouping
    - Semplificato tra utenti diversi
    - Complesso per ciascun utente singolo

Direttori ad albero

- Generalizza i precedenti
- I file sono contenuti in un albero
  - Ogni nodo/vertice può contenere come entry un altro nodo/vertice dell'albero



Direttori ad albero

- Ogni utente può gestire tanto file quanto direttori e sotto-direttori
  - Nascono i concetti di working directory, cambio direttorio, path assoluto e relativo, etc.
- Prestazioni
  - Efficiency
    - Ricerche vincolate alla struttura ad albero e quindi alla sua profondità e ampiezza
  - Naming
    - Permesso in maniera estesa
  - Grouping
    - Permesso in maniera estesa

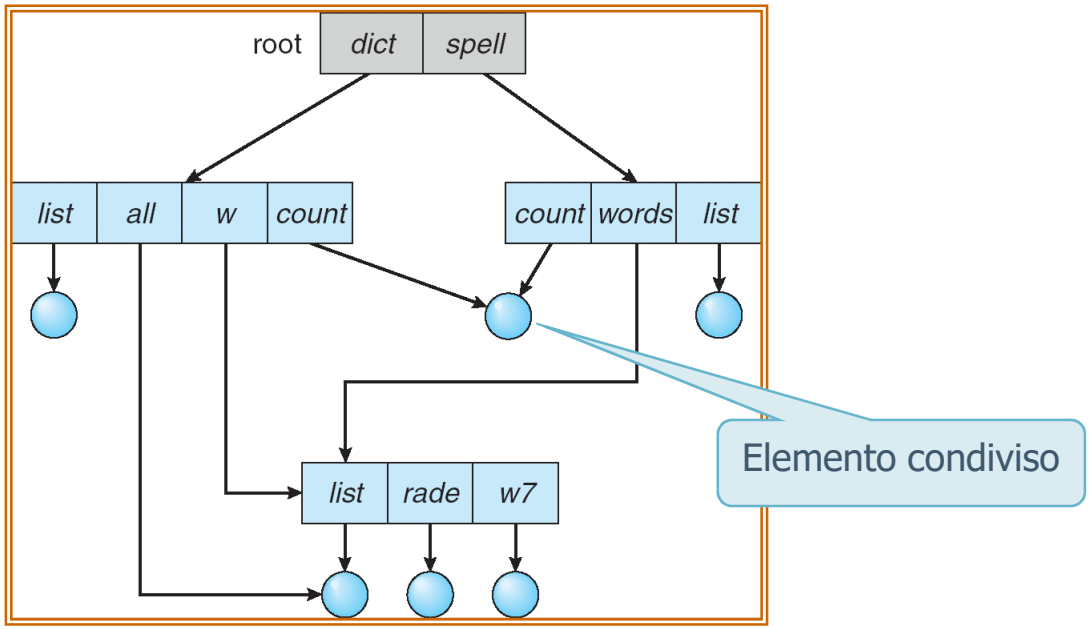
Concetti analizzati nella parte sperimentale Linux

Direttori a grafo aciclico

- File system a albero non permettono la **condivisione** di informazioni
- Spesso è utile individuare lo stesso oggetto con nomi diversi
  - Da parte dello stesso utente con path diversi
  - Da parte di utenti diversi
  - Si osservi che la duplicazione delle informazioni (copia) non risolve tale problema a causa di
    - Aumento dello spazio occupato dal file system
    - Coerenza delle informazioni presenti nelle varie copie

Direttori a grafo aciclico

- I file system a grafo orientato aciclico (cioè senza cicli)
  - Permettono di condividere informazioni, rendendola visibile con path diversi



Direttori a grafo aciclico

- La presenza di link aumenta la difficoltà di gestione dei file system
  - Occorre distinguire gli oggetti nativi dai relativi collegamenti in fase di creazione, manipolazione e cancellazione
  - Creazione
    - La condivisione di una entry può essere ottenuta in diversi modi
    - Nei sistemi UNIX-like la strategia standard è quella della creazione di **collegamenti** o **link**
      - Un link è un riferimento (puntatore) a un'altra entry (pre-esistente)

Direttori a grafo aciclico

- Visita e ricerca
  - Se la entry è un link occorre accorgersene e effettuare un indirizzamento indiretto, ovvero "risolvere" il collegamento, utilizzandolo per raggiungere l'entry originaria
  - Tramite link ogni entry del file system può essere raggiungibile con più path assoluti (e con nomi diversi)
    - Analisi del file system (statistiche, e.g., quanti file di estensione ".c" sono presenti?) diventano molto più complesse

Direttori a grafo aciclico

- Cancellazione
  - Occore stabilire come gestire il link e come gestire l'oggetto riferito
    - La cancellazione di un link in genere viene effettuata in maniera immediata e non influisce sull'oggetto originale
    - Occorre però decidere come effettuare la cancellazione dell'oggetto
      - Se si cancella l'oggetto che cosa si fa dei link che lo riferiscono?
      - Quando si riutilizza lo spazio ad esso riservato?

Direttori a grafo aciclico

- Cancellazione immediata dei dati
  - È possibile lasciare dei link pendenti (dangling)
  - Quando si cercherà di utilizzare il link in futuro ci si accorgerà che il file riferito è scomparso

Soft-link UNIX

- Cancellazione dei dati alla cancellazione dell'ultimo link
  - Per evitare link pendenti occorre tenerne traccia, ovvero occorre gestire la presenza di oggetti e link multipli
    - Mantenere l'elenco di tutti i link è costoso (lista di lunghezza variabile)
    - Cancellare tutti i link (le entry) alla cancellazione dell'oggetto è costoso, in quanto occorrerebbe ricercarli

Hard-link UNIX

  - Conviene memorizzare solo il contatore (numero di link)
    - Nei sistemi UNIX tale contatore viene memorizzato negli i-node
    - Esso viene incrementato e decrementato in maniera opportuna

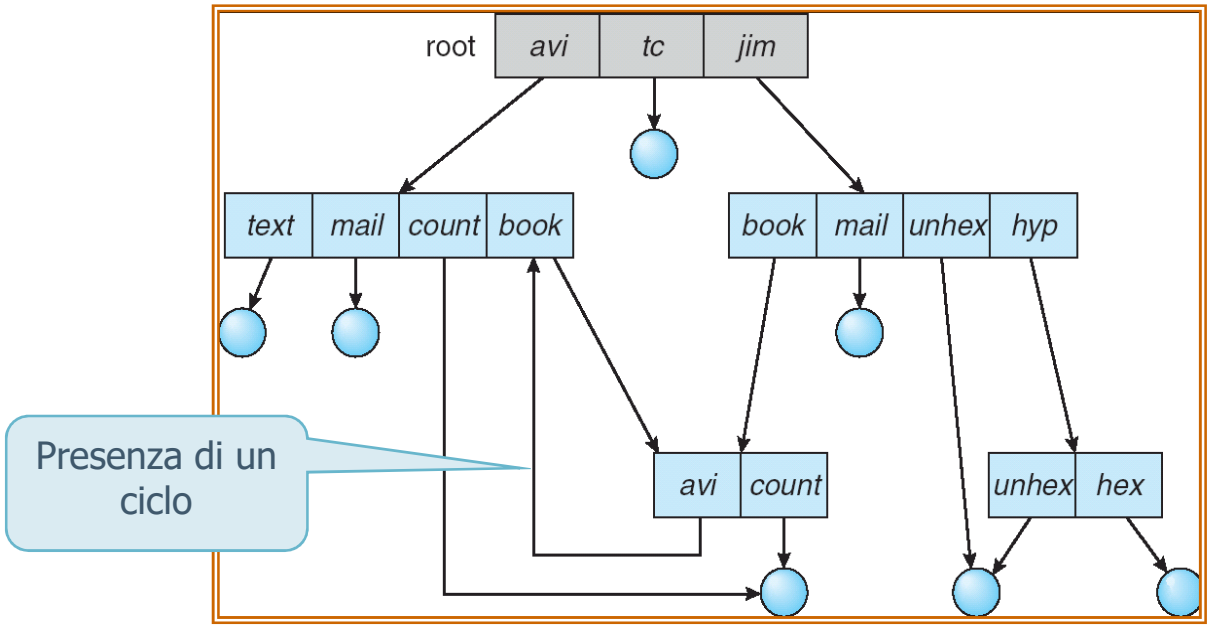
Comando "ls –i"

Direttori a grafo aciclico

- Inoltre la creazione di un link a un direttorio potrebbe causare la nascita di un ciclo nel file system
  - La gestione di un grafo ciclico richiede operazioni di maggiore complessità
    - Un comando di ricerca o di visita dovrebbe verificare la presenza di cicli per evitare ricorsioni infinite
  - Tra le possibili strategie la più semplice è non permette la creazione di link a direttori

Direttori a grafo ciclico

- L'alternativa al grafo aciclico e quella del grafo ciclico ovvero occorre
  - Permettere la creazione di cicli
  - Gestire opportunamente i cicli esistenti in tutte le fasi



Direttori a grafo ciclico

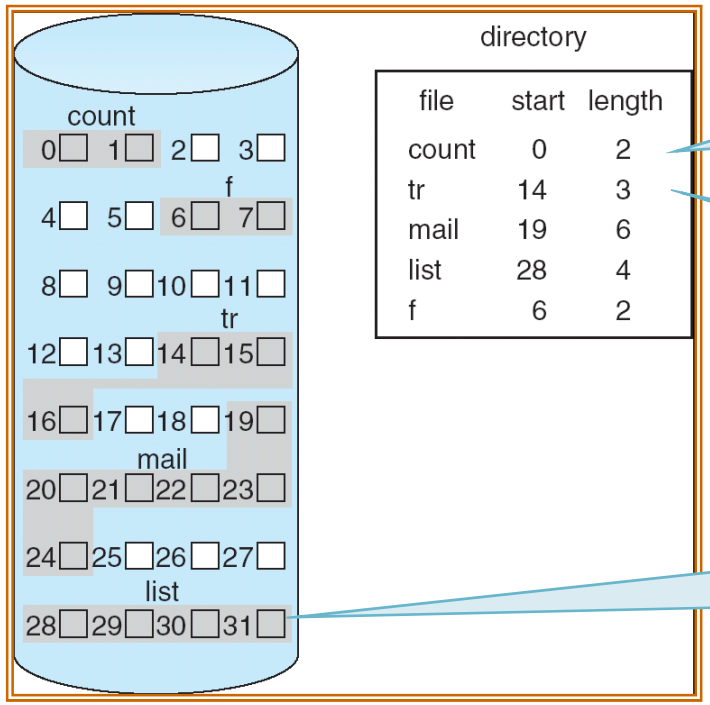
- La gestione può essere effettuata con diversi approcci che dovrebbero tenere conto di diverse problematiche
  - Un elemento potrebbe auto-referenziarsi e non essere mai cancellato e/o rilevato
  - La gestione più semplice consiste nel non visitare **mai** i link (oppure sotto-categorie dei link)

Allocazione

- Per **allocazione** si intendono le tecniche di utilizzo dei blocchi dei dischi per la memorizzazione di file
  - Non ci occuperemo della struttura delle unità di memorizzazione
  - In ogni caso tali unità possono essere viste come un insieme indicizzabile lineare (vettore) di blocchi
- Metodologie di allocazione principali
  - Contigua (contiguous)
  - Concatenata (linked)
  - Indicizzata (indexed)

Allocazione contigua

- Ogni file occupa un insieme contiguo di blocchi



Per ciascun file il direttorio specifica l'indirizzo del primo blocco (b) e la lunghezza del file (n)

Il file occupa i blocchi b, b+1, b+2, … , b+n-1

Ogni file presenta frammentazione interna (ultimo blocco parzialmente occupato)

Allocazione contigua

- Vantaggi
  - Strategia di allocazione molto semplice
    - Per ogni file si memorizzano poche informazioni
  - Permette accessi sequenziali immediati
    - Ogni blocco si trova dopo il precedente e prima del successivo
  - Permette accessi diretti semplici
    - L'i-esimo blocco a partire dal blocco b si trova all'indirizzo b+i-1

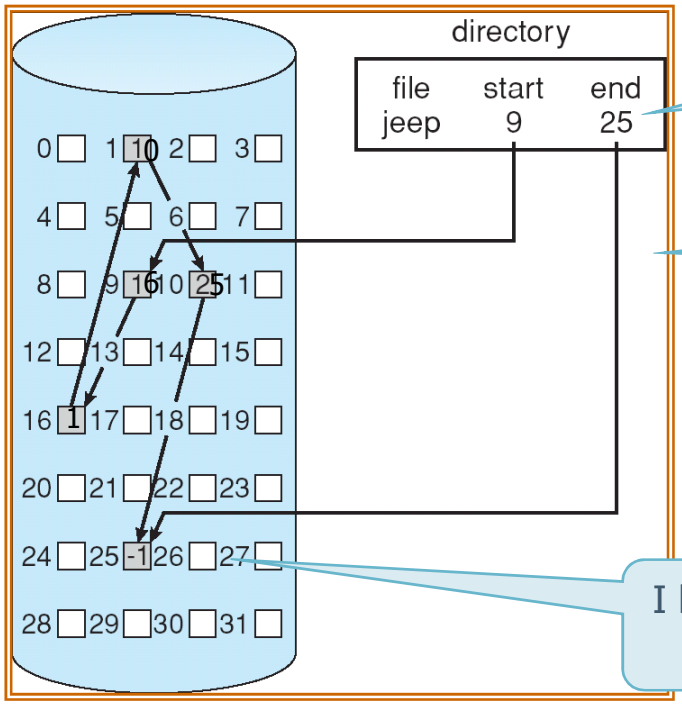
Allocazione contigua

- Svantaggi
  - Occorre decidere la politica di allocazione
    - Dove vengono allocati i file nuovi?
      - Algoritmi di first-fit, best-fit, worst-fit, etc.
      - Come è possibile determinare lo spazio necessario?
  - Nessun algoritmo di allocazione risulta privo di difetti quindi la tecnica spreca spazio
    - Tale spreco è noto come **frammentazione esterna** (insieme dei blocchi non utilizzati)
    - Possibile la ri-compattazione (on-line e off-line)
  - Problemi di allocazione dinamica
    - I file non possono crescere liberamente in quanto lo spazio disponibile è limitato dal file successivo

Occorre ricercare uno spazio libero di dimensione sufficiente

Allocazione concatenata

- Ogni file può essere allocato gestendo una lista concatenata di blocchi



Il direttorio contiene un puntatore al primo e uno all'ultimo blocco del file

Ogni blocco contiene un puntatore al blocco successivo

I blocchi di ciascun file sono sparsi per l'intero disco

Allocazione concatenata

- Vantaggi
  - Risolve i problemi dell'allocazione contigua
    - Permette l'allocazione dinamica dei file
    - Elimina la frammentazione esterna
    - Evita l'utilizzo di algoritmi di allocazione complessi

Allocazione concatenata

- Svantaggi
  - Ogni lettura implica un accesso sequenziale ai blocchi
  - Risulta efficiente solo per accessi sequenziali
    - Un accesso diretto richiede la lettura di una catena di puntatori sino a raggiungere l'indirizzo desiderato
    - Ogni accesso a un puntatore implica una operazione di lettura dell'intero blocco
  - La memorizzazione dei puntatori
    - Richiede spazio
    - È critica dal punto di vista dell'affidabilità
    - Rende lo spazio utile minore
      - Lo spazio disponibile non è più una potenza di 2

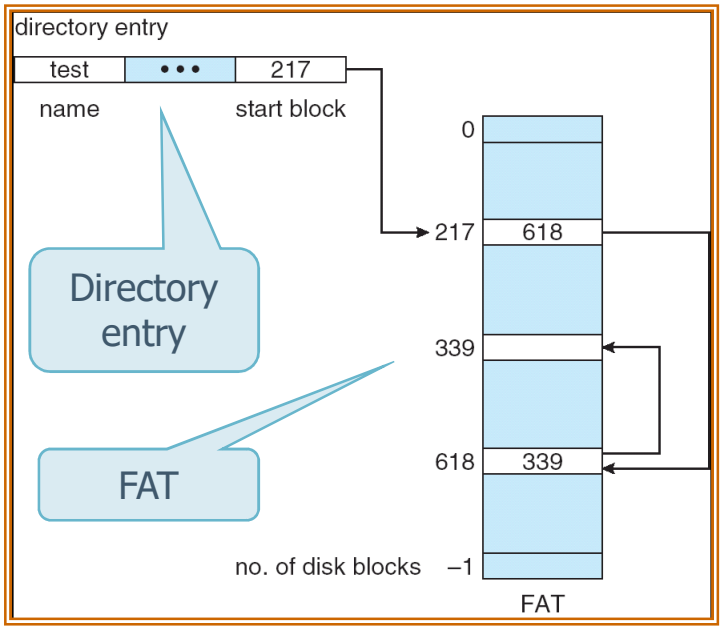
Allocazione concatenata: FAT

- L'allocazione utilizzata da MS-DOS
  - È basata su FAT (File Allocation Table)
  - È una variante del metodo di allocazione concatenato
- Il blocco degli indici contiene la FAT
  - Tabella con un elemento per ciascun blocco presente sul disco
  - La sequenza dei blocchi appartenenti a un file è individuata a partire dalla directory mediante
    - Elemento di partenza del file nella FAT
    - Sequenza di puntatori presenti (direttamente) nella FAT (non più nei blocchi)

Sposta i puntatori dai vari blocchi a un blocco specifico

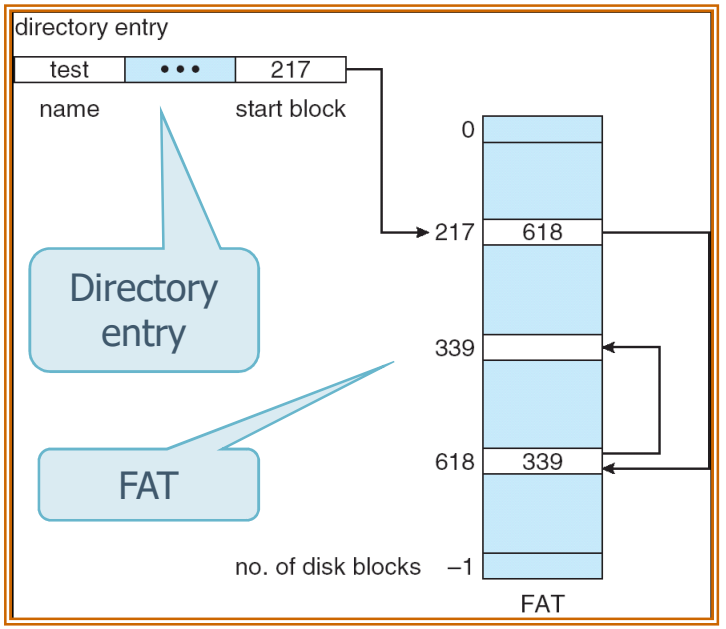
Allocazione concatenata: FAT

- I riferimenti non sono memorizzati nei blocchi su disco ma direttamente negli elementi della FAT
- La lettura di ogni blocco richiede due accessi a disco
  - Il primo accesso viene effettuato alla FAT
  - Il secondo, al blocco dati



Allocazione concatenata: FAT

- Limiti
  - Accesso lento
  - L'affidabiità è critica (persa la FAT si perde tutto)
  - La dimensione della FAT è critica

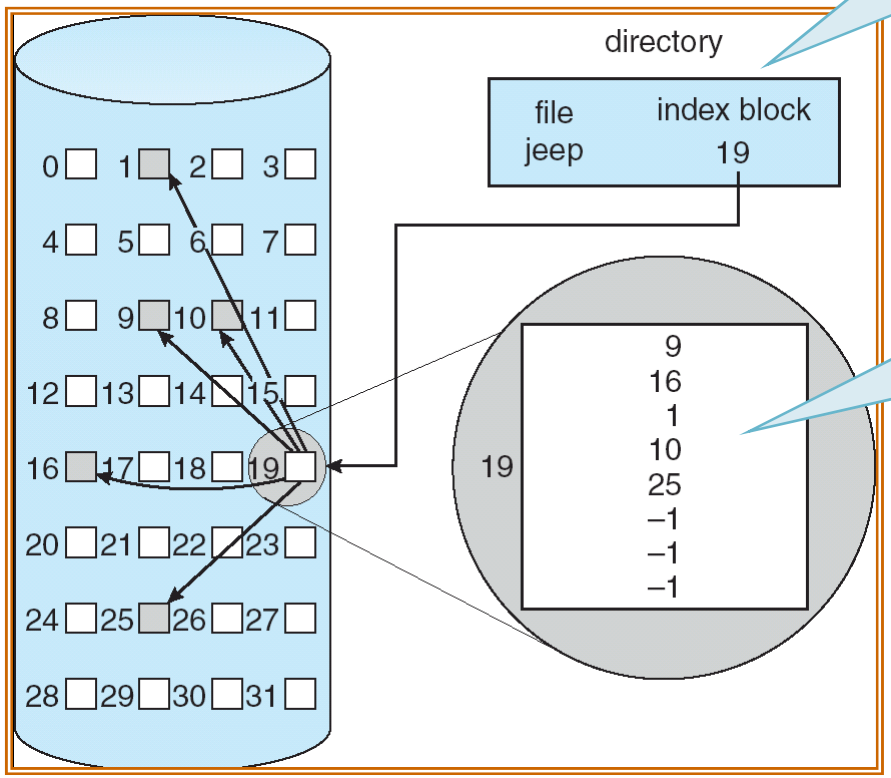


Allocazione indicizzata

- Per permettere un accesso diretto efficiente è possibile inglobare tutti i puntatori in una tabella di puntatori
  - Tale tabella di puntatori è solitamente denominata **blocco indice** o index-node (**i-node**)
- Ogni file ha la sua tabella, ovvero un vettore di indirizzi dei blocchi in cui il file è contenuto
  - L'i-esimo elemento del vettore individua l'i-esimo blocco del file

Allocazione indicizzata

Il direttorio contiene il solo puntatore al blocco indice



Non è una FAT perchè i puntatori sono tutti in sequenza (**non** si ha una **lista** di puntatori)

Allocazione indicizzata

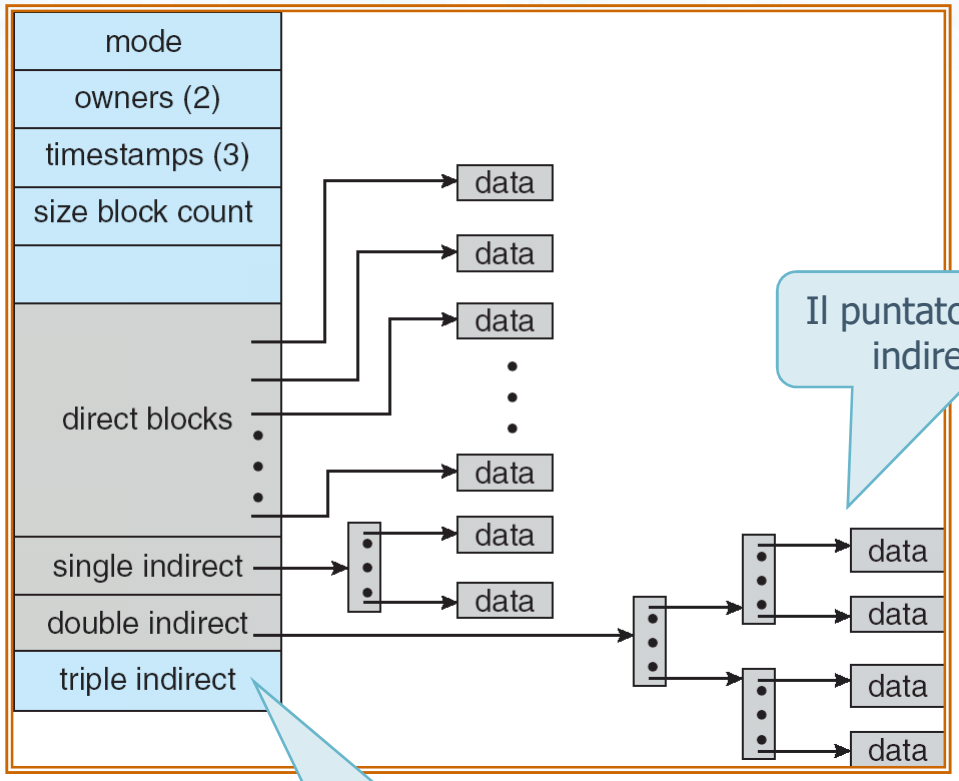
- Rispetto all'allocazione concatenata occorre sempre allocare un blocco indice
  - Blocchi indice di dimensione ridotta permettono di non sprecare troppo spazio
  - Blocchi indice di dimensione elevata aumentano in numero di riferimenti inseribili nel blocco indice
    - In ogni caso occorre gestire la situazioni il cui il blocco indice **non** è sufficiente a contenere tutti i puntatori ai blocchi del file
    - Esistono diversi schemi
      - A blocchi indice concatenati
      - A blocchi indice a più livelli
      - **Combinato**

Schema UNIX/Linux

Allocazione indicizzata: schema combinato

- Lo schema combinato è utilizzato nei sistemi UNIX/Linux
- A ogni file è associato un blocco detto **i-node**
- Ogni **i-node** contiene diverse informazioni tra cui 15 puntatori ai blocchi dati del file
  - I primi 12 puntatori sono puntatori diretti, ovvero puntano a blocchi dei file
  - I puntatori 13, 14 e 15 sono puntatori indiretti, con livello di indirizzamento crescente
    - Il blocco individuato non contiene i dati ma i puntatori (i puntatori ai puntatori) [i puntatori ai puntatori ai puntatori] a blocchi di dati del file

Allocazione indicizzata: schema combinato



Ricordare comandi "ls –la" e "ls –i"

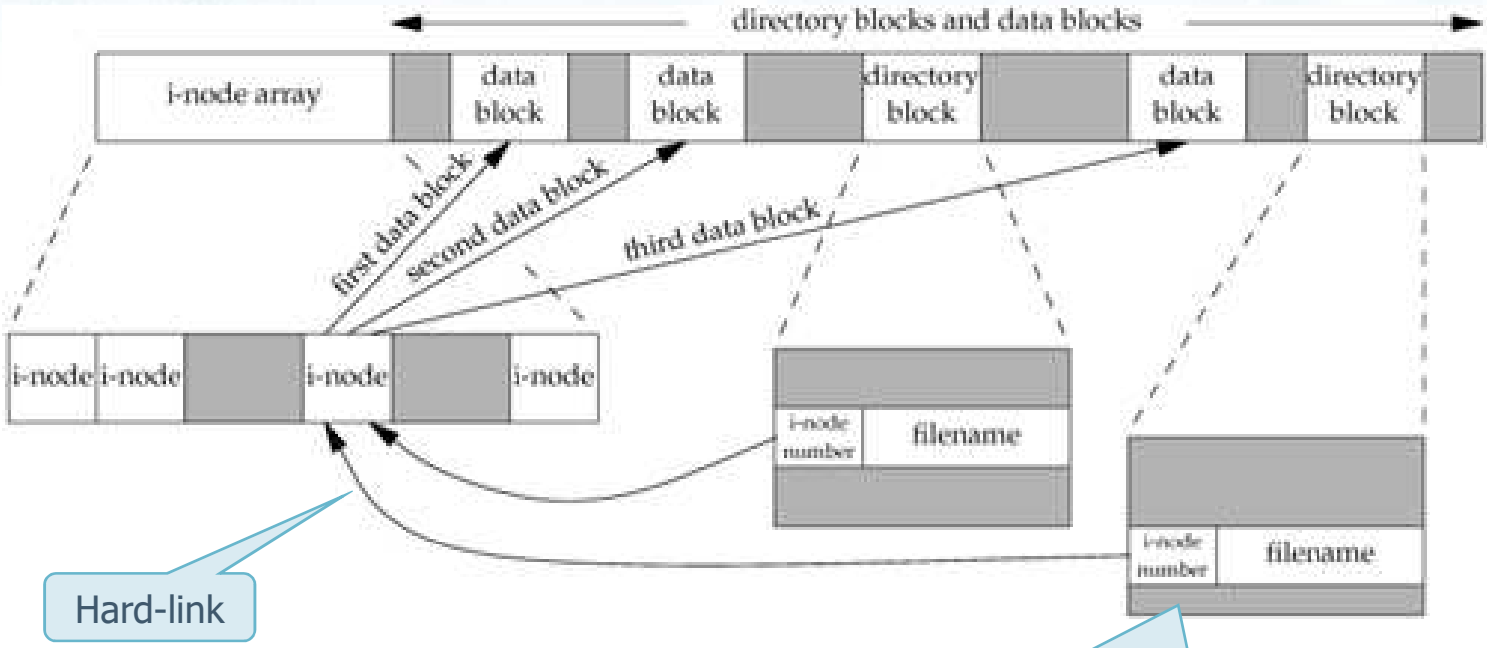
Il puntatore 14 è di tipo indiretto doppio

Il puntatore 13 è di tipo indiretto singolo

Il puntatore 15 è di tipo indiretto triplo

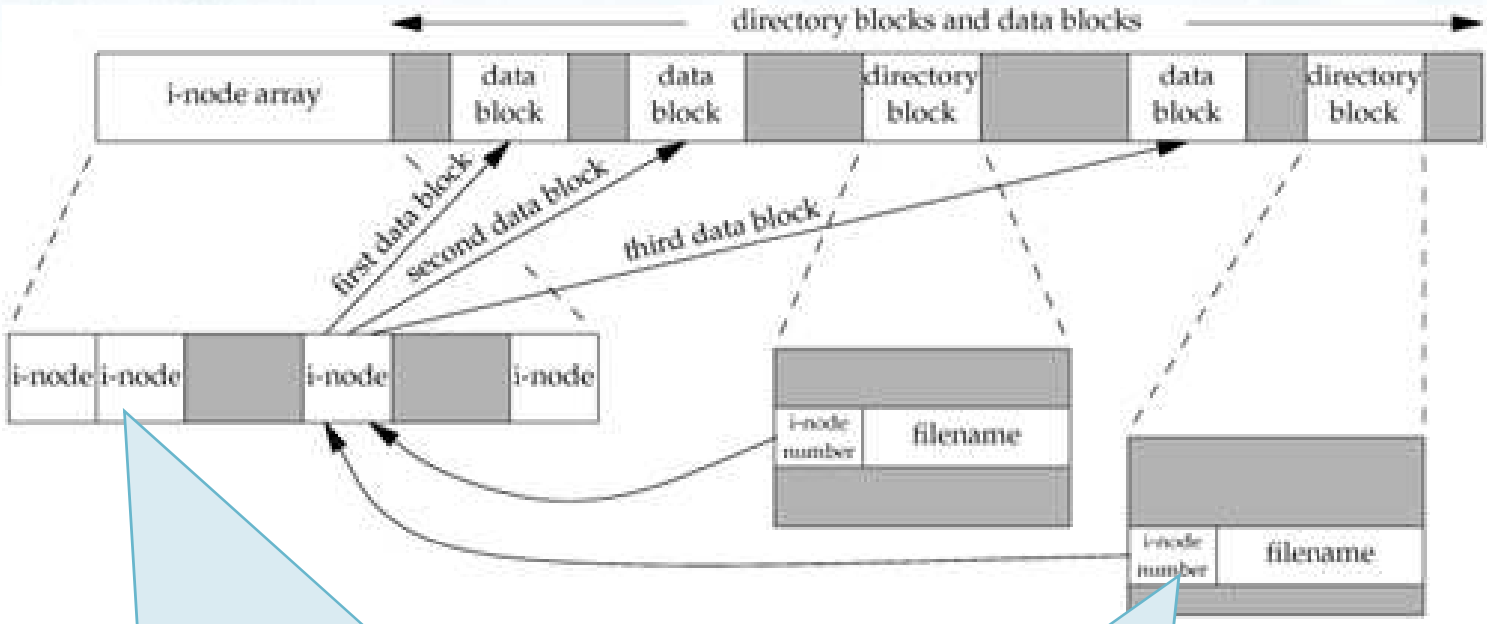
Con puntatori a 64 bit si memorizzano file sino a 2^{60} byte (exabyte)

Allocazione indicizzata: schema combinato



Un direttorio è una tabella che associa a ogni nome di file un **i-node number**
Il link da un direttorio al rispettivo i-node è detto **hard-link**
Lo stesso i-node number può essere individuato da più link

Allocazione indicizzata: schema combinato



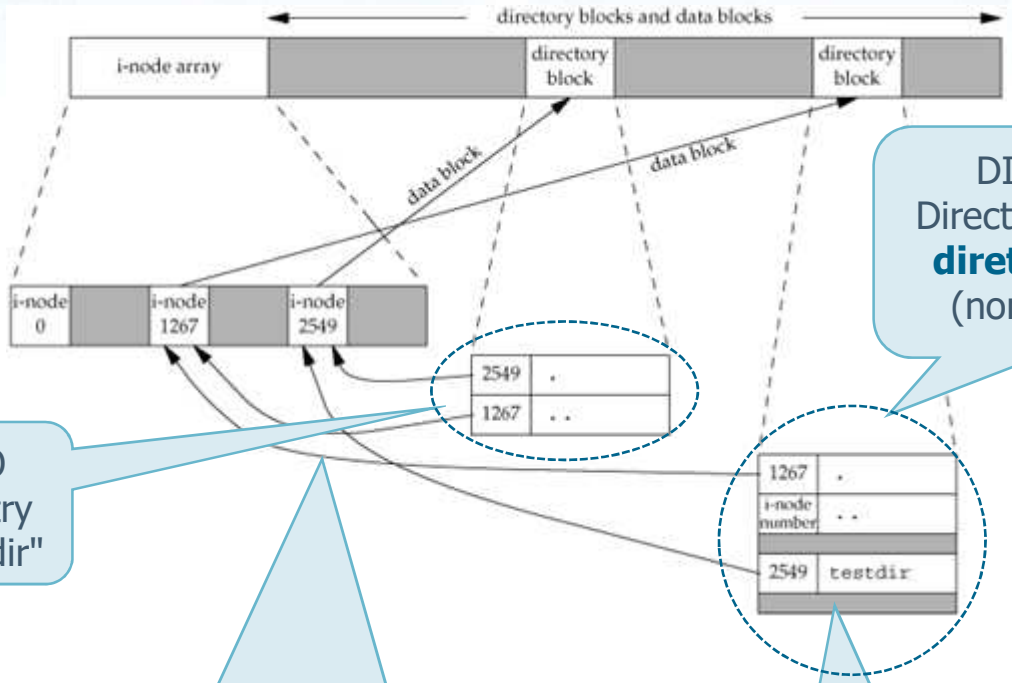
Record di lunghezza fissa che contiene la maggior parte di informazioni relative ai file (i.e., ne identifica i blocchi che lo compongono)
Contiene un contatore che individua il numero di puntatori (link)
Numerati a partire da 1; alcuni sono riservati al SO

L'i-node number corrisponde all'indice di una tabella in cui ogni elemento è un i-node e contiene le informazioni di un file

Allocazione indicizzata: schema combinato

- Hard link (link effettivo o fisico)
  - Directory entry che punta a un i-node
  - Non esistono hard link
    - Verso direttori (evita filesystem a grafo ciclico)
    - Verso file su altri file system
  - Un file è fisicamente rimosso solo quando tutti i suoi hard link sono stati rimossi
- Soft link (link simbolico)
  - Il blocco dati individuato dall'i-node punta a un blocco che contiene il path name del file
  - Sostanzialmente è un file che come unico blocco dati ha il nome di un altro file

Filesystem UNIX: Un esempio



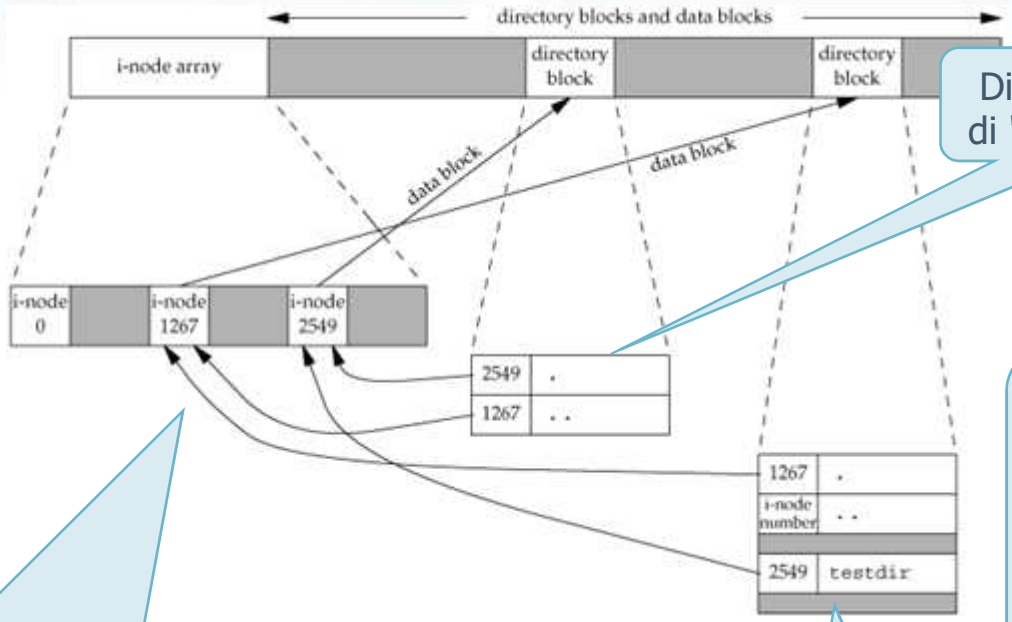
DIR PADRE
Directory entry del **direttorio** 1267 (nome ignoto)

DIR FIGLIO
Directory entry di "2549 testdir"

L'i-node 2549 è un sotto-direttorio foglia
Il suo hard link count è **pari** a **2**
Uno deriva dal direttorio padre ("testdir")
Uno deriva da se stesso ("testdir/.")

L'i-node "2549 testdir" è un sotto-direttorio **foglia** di 1267

Filesystem UNIX: Un esempio



Directory entry di "2549 testdir"

Ogni sotto-direttorio incrementa il numero di link del padre di 1 !

L'i-node 1267 è un direttorio con un sotto-direttorio
Il suo link count è pari **almeno** a 3
Uno deriva dal direttorio padre (non indicato)
Uno deriva da se stesso (".")
Uno deriva dal direttorio figlio ("./testdir/..")

L'i-node "2549 testdir" è un sotto-direttorio **foglia** di 1267

Manipolazione del file system

- Lo standard POSIX mette a disposizione un insieme di funzioni per effettuare la manipolazione dei direttori
  - La funzione **stat**
    - Permette di capire di che tipo di "entry" si tratta (file, direttorio, link, etc.)
    - Tale operazione è permessa da una struttura C di tipo **struct stat**
  - Alcune altre funzioni di manipolazione
    - getcwd, chdir
    - mkdir, rmdir
    - opendir, readddir, closedir

Struttura dati ritornata

Posizionamento

Creazione cancellazione

Visita

stat ()

```
#include <sys/types.h>
#include <sys/stat.h>

int stat (const char *path, struct stat *sb);
int lstat (const char *path, struct stat *sb);
int fstat (int fd, struct stat *sb);
```

Path di cui ritornare le informazioni

Struttura dati ritornata

- La funzione **stat** restituisce il riferimento dalla struttura **sb** (**struct stat**) per il file (o descrittore) passato come parametro
- Valori di ritorno
  - Il valore 0 se l'operazione ha avuto successo
  - Il valore -1 se c'è errore

stat ()

- La funzione
  - **lstat** restituisce informazioni sul link simbolico, non sul file puntato dal link (quando il path individua un link)
  - **fstat** restituisce informazioni su un file già aperto (riceve il descrittore del file invece del path)

```
int stat (const char *path, struct stat *sb);
int lstat (const char *path, struct stat *sb);
int fstat (int fd, struct stat *sb);
```

stat ()

```
struct stat {
  mode_t st_mode;      /* file type & mode */
  ino_t st_ino;        /* i-node number */
  dev_t st_dev;        /* device number */
  dev_t st_rdev;       /* device number */
  ...
};
```

Tipi primitivi

- Il secondo argomento di **stat** è il puntatore alla struttura **stat**
- Il campo **st_mode** codifica il tipo di file

stat ()

```
struct stat {
  mode_t st_mode;      /* file type & mode */
  ino_t st_ino;        /* i-node number */
  dev_t st_dev;        /* device number */
  dev_t st_rdev;       /* device number */
  ...
};
```

- Alcune macro permettono di capire il tipo del file
  - **S_ISREG** regular file, **S_ISDIR** directory, **S_ISBLK** block special file, **S_ISCHR** character special file, **S_ISFIFO** FIFO, **S_ISSOCK** socket, **S_ISLNK** symbolic link

Esempio

Verifica del tipo di directory entry

Permette di capire se si tratta di un direttorio !

```
struct stat buf;
...
if (stat(argv[i], &buf) < 0) {
  fprintf (stdout, "lstat error.\n");
  exit (1);
}
if      (S_ISREG(buf.st_mode))  ptr = "regular";
else if (S_ISDIR(buf.st_mode))  ptr = "directory";
else if (S_ISCHR(buf.st_mode))  ptr = "char special";
else if (S_ISBLK(buf.st_mode))  ptr = "block special";
else if (S_ISFIFO(buf.st_mode)) ptr = "fifo";
else if (S_ISLNK(buf.st_mode))  ptr = "symbolic link";
else if (S_ISSOCK(buf.st_mode)) ptr = "socket";
     printf("%s\n", ptr);
}
```

getcwd ()

```
#include <unistd.h>

char *getcwd (char *buf, int size);
```

Dimensione di buf

Get Current Working Directory

- Ottiene il path del direttorio di lavoro
- Valore di ritorno
  - Il buffer buf se è OK; NULL se c'è errore

chdir ()

```
#include <unistd.h>

int chdir (char *path);
```

Change Directory

- Modifica il path del direttorio di lavoro
- Valori di ritorno
  - Il valore 0 se è OK; il valore -1 se c'è errore

Esempio

Uso di getcwd e chdir in un programma

```
#define N 100

char name[N];

if (getcwd (name, N) == NULL)
  fprintf (stderr, "getcwd failed.\n");
else
  fprintf (stdout, "dir %s\n", name);

if (chdir(argv[1]) < 0)
  fprintf (stderr, "chdir failed.\n");
else
  fprintf (stdout, "dir changed to %s\n", argv[1]);
```

mkdir ()

```
#include <unistd.h>
#include <sys/stat.h>

int mkdir (const char *path, mode_t mode);
```

Vedere system call open

- Creano un nuovo direttorio (vuoto)
- Valore di ritorno
  - Il valore 0 se è OK
  - Il valore -1 se c'è errore

rmdir ()

```
#include <unistd.h>
#include <sys/stat.h>

int rmdir (const char *path);
```

- Cancellano un direttorio (se vuoto)
- Valori di ritorno
  - Il valore 0 se è OK
  - Il valore -1 se c'è errore

opendir (), dirent () e closedir ()

```
#include <dirent.h>


DIR *opendir (
  const char *filename
);


struct dirent *readdir (
  DIR *dp
);


int closedir (
  DIR *dp
);
```

Apre un direttorio in lettura
Valori di ritorno:
Il puntatore al direttorio se corretta
Il puntatore NULL in caso di errore

Continua la lettura del direttorio
Valori di ritorno:
Il puntatore al direttorio se corretta
Il puntatore NULL in caso di errore o al termine della lettura

Termina la lettura
Valori di ritorno:
Il valore 0 se corretta
Il valore -1 in caso di errore

Struttura dirent

```
struct dirent {
  inot_t d_no;
  char d_name[NAM_MAX+1];
  ...
}
```

- La struttura **struct dirent** ritornata da **readdir**
  - Ha formato che dipende dall'implementazione
  - Contiene almeno i campi indicati
    - Il numero di i-node
    - Il nome del file (null-terminated)

Esempio

- Si scriva un programma in grado di
  - Ricevere da linea di comando un directory path
  - Visualizzare il contenuto del direttorio differenziando le entry "file" da quelle "dir"

Esempio

```
#define N 100
...
struct stat buf;
DIR *dp;
char fullName[N];
struct dirent *dirp;
int i;
...
if (stat(argv[1], &buf) < 0 ) {
  fprintf (stderr, "Error.\n"); exit (1);
}
if (S_ISDIR(buf.st_mode) == 0) {
  fprintf (stderr, "Error.\n"); exit (1);
}
if ( (dp = opendir(argv[1])) == NULL) {
  fprintf (stderr, "Error.\n"); exit (1);
}
```

Struttura per lstat

Directory "handle"

Struttura per readdir

Richiesta informazioni sul path in argv[1]

Se non è un direttorio termina

In caso contrario si apre il direttorio

Esempio

```
i = 0;
while ( (dirp = readdir(dp)) != NULL) {
  sprintf (fullName, "%s/%s", argv[1], dirp->d_name);
  if (lstat(fullName, &buf) < 0 ) {
    fprintf (stderr, "Error.\n"); exit (1);
  }
  if (S_ISDIR(buf.st_mode) == 0) {
    fprintf (stdout, "File %d: %s\n", i, fullName);
  } else {
    fprintf (stdout, "Dir  %d: %s\n", i, fullName);
  }
  i++;
}
if (closedir(dp) < 0) {
  fprintf (stderr, "Error.\n"); exit (1);
}
```

Lettura direttorio (iterando su tutte le entry)

Visualizzazione dati

Richiesta informazioni sulla entry fullName

Chiusura e terminazione