

# **The Second Extended File System**

Struttura e implementazione

Salvatore Davide Rapisarda

# Indice

---

1. Definizione di file system e termini legati ad esso.....	3
2. Dischi rigidi.....	5
2.1 Hardware dei dischi rigidi.....	5
2.2 Indirizzamento dei settori fisici.....	5
2.3 Mbr e partizioni.....	6
3. Breve storia del file system di Linux.....	8
4. The Second Extended File System.....	9
4.1 Struttura del file system.....	9
4.2 Super Block.....	10
4.3 Group Descriptor.....	12
4.4 I-Node.....	13
4.4.1 Attributi e permessi di un file.....	15
4.4.2 Blocchi di dati .....	15
4.4.3 Collegamenti ( Links ).....	16
4.5 Entrate delle directory.....	17
5. Virtual File System.....	18
6. Mantenimento del file system.....	20

## Capitolo 1.

# Definizione di file system e termini legati ad esso.

---

La maggior parte delle applicazioni che girano su di un calcolatore hanno bisogno di memorizzare in modo permanente e di rintracciare informazioni. Queste informazioni sono registrate all'interno dei **file**. Ad ogni file vengono associati un insieme di attributi come: il nome, l'accesso, i permessi, la dimensione, data di creazione o data ultima modifica. La parte di un sistema operativo che si occupa complessivamente dei file è chiamata **file system**. All'interno di un file system sono trattati diversi tipi di file: esistono infatti file regolari, file speciali a caratteri, file speciali a blocchi e le directory.

I file regolari sono quei file che contengono i dati memorizzati dall'utente proprietario del file. I file speciali a caratteri sono dei file in cui è possibile leggere un carattere alla volta e vengono usati per gestire unità seriali, stampanti e reti. Mentre, i file speciali a blocchi, sono dei file in cui è possibile leggere dei blocchi di informazione, cioè un certo numero di caratteri alla volta, e vengono usati per gestire unità disco come dischi rigidi, unità floppy e unità CD-ROM.

Per tener traccia dei file, il file system normalmente fornisce le **directory**, che in molti sistemi sono a loro volta dei file. Una directory, praticamente, è un file che può contenere file e altre directory. Il file system fornisce anche una directory speciale chiamata **root directory**, che ha la funzione di contenere tutti i file e le directory del file system.

E' possibile vedere le directory di un file system come una struttura ad albero, la cui radice è la root directory e i nodi dell'albero delle directory, che a loro volta contengono i files.

Un'altra distinzione da fare per i file è quella riguardo al tipo di accesso. I sistemi operativi offrono due tipi di accesso : **l'accesso sequenziale** e **l'accesso casuale**. Nel primo caso un applicazione può leggere tutti i byte di un file in ordine partendo dall'inizio, senza saltarne qualcuno o leggerli fuori dall'ordine. Nel secondo caso, i file ad accesso casuale, invece, possono essere letti in qualunque ordine.

Le chiamate di sistema che normalmente un sistema operativo offre per la gestione del file system sono:

1. CREATE: Viene creato un file senza dati.
2. DELETE: Viene cancellato il file dal file system.
3. OPEN: Apre un file e carica un descrittore del file in memoria.
4. CLOSE: Chiude un file.
5. READ: Legge un numero di byte da un file.
6. WRITE: Scrive un numero di byte su un file.

7. APPEND: Aggiunge un numero di byte alla fine di un file
8. SEEK: Muove il puntatore dei dati in un file ad accesso casuale.
9. SET ATTRIBUTES: Modifica gli attributi di un file. Questa più che una chiamata di sistema sono più chiamate di sistema che modificano i vari attributi di un file.
10. RENAME: Cambia il nome di un file.

Il sistema operativo offre anche delle chiamate di sistema per la gestione delle directory.

1. OPENDIR: Apre una directory ( scende di un livello nell'albero )
2. CLOSEDIR: Chiude una directory ( risale di un livello nell'albero )
3. READDIR: Legge il contenuto di una directory, e quindi il nome dei file contenuti in essa.
4. RENAME: Cambia il nome di una directory.
5. LINK: Crea un collegamento della directory in un'altra zona del filesystem.
6. UNLINK: Rimuove un collegamento.

# Dischi Rigidi

---

## 2.1 Hardware dei dischi rigidi

Un disco rigido è formato da più dischi, tutti imperniati attorno allo stesso asse di rotazione, con le testine di lettura/scrittura che si affacciano su entrambe le superfici di ogni piatto, non costrette a seguire una traccia a spirale, ma libere di essere portate avanti ed indietro su tutta la superficie. Essendo però unico il blocco dei bracci che le trasporta, sono comunque costrette a stare tutte alla stessa distanza dall'asse, una sotto l'altra. Per mappare ogni zona di un disco contenente dati, i cosiddetti settori fisici, si sono usati tre numeri: il numero di cilindro ( Cylinder, **C** ), il numero di testina ( Head, **H** ) ed il numero di settore ( Sector, **S** ). Il numero di testina individua la testina che si occupa della lettura/scrittura dei dati e la superficie su cui si affaccia.

Il numero di cilindro individua la distanza del settore fisico, cioè la posizione del braccio che porta le testine, ed è il numero d'ordine della traccia su cui esso si trova, quando tutte le tracce ( concentriche ) siano state numerate consecutivamente a partire dalle più esterne. Infine, lungo la circonferenza di ciascuna traccia si trovano i **settori fisici**, che sono blocchi di dati aventi dimensione di 512 byte, che costituiscono l'unità minima di allocazione.

## 2.2 Indirizzamento dei settori fisici

I costruttori dei primi PC non hanno tenuto conto in alcun caso del principio ingegneristico di modularità, che permette di assemblare più parti distinte in modo che il costruttore di una sia costretto a sapere solo il minimo indispensabile riguardo alle altre. I programmi che vogliono memorizzare dati sul disco sono stati costretti a conoscere molto su di esso, e cioè non solo di che tipo di memoria si tratta, ma anche dei dettagli implementativi. Da qualche parte qualcuno dovrà tradurre il numero di indirizzamento logico con cui i programmi pensano, in una posizione sul disco, per esempio nei numeri CHS, ma la cosa più sensata è che se ne occupi l'elettronica del disco. L'INT13h è colui a cui tutte le applicazioni si rivolgevano per effettuare letture e scritture sul disco, ed è la causa principale di tutti i problemi relativi alla dimensione e all'indirizzamento effettivo di tutti i settori fisici di un disco. I moderni sistemi operativi non usano più questa tecnica e usano i controller dei dischi rigidi per effettuare letture/scritture su un disco. I numeri massimi di combinazioni possibili, dato dal prodotto dei rispettivi valori di C, H e S, producono i valori del numero di settori "indirizzabili", i quali, moltiplicati per i 512 byte di un settore, producono i valori

di soglia che hanno creato problemi quando la dimensione tipica di un disco rigido li superava. Il sistema operativo che vuole indirizzare un settore fisico utilizza un indirizzo logico dei settori ( Logical Sector Number, LSN ) che in teoria potrebbe passare direttamente al controllore del disco in quella che si chiama Logical Block Addressing (LBA). Poiché l'LSN non viene passato direttamente al disco ( LBA ), ma viene convertito prima di essere passato al BIOS, in un valore CHS, con al massimo 16 testine, 1.024 cilindri e 63 settori, si possono gestire dischi fino a 528.482.304 byte. La cosiddetta barriera dei 528MB.

Per adeguarsi, due bit per i settori sono stati passati ai cilindri arrivando a  $256/4 = 64$  settori e  $256 * 4 = 1.024$  cilindri. Per rappezzare la situazione è stato inventato l'E-CHS ( Extended-CHS ), che consiste nel presentare alle applicazione la “geometria tradotta” del disco. Visto che per le applicazioni le testine possono essere fino a 256, si moltiplicano queste per 2, 4, 8, o 16, dividendo per lo stesso fattore il numero di cilindri.

Le applicazioni moderne usano passare direttamente l'indirizzo logico dei settori al disco usando la modalità LBA. Ma sfortunatamente, il numero LBA ha solo 28 bit e non 32 bit come l'LSN, perché lo standard E-IDE usa 16 bit per i cilindri, 4 bit per le testine ed 8bit per i settori, che in totale sono appunto 28 bit. L'E-CHS non aiuta assolutamente in questo caso. Infatti se il fattore di moltiplicazione e divisione è N, si hanno al massimo  $1.024 * N$  cilindri,  $256 / N$  testine, 63 settori, per un massimo di  $( 1024 * N ) * ( 256 / N ) * 63 * 512$  byte = 8.455.716.864 byte. Il limite degli 8,4 Gb. Per superare quest'ultimo limite, l'unica soluzione è quella di aggiornare il BIOS della macchina che porta il BIOS ad una modifica delle routine di lettura/scrittura dell'INT13.

### 2.3 MBR e Partizioni

Il primo settore di ogni disco rigido viene chiamato MBR ( Master Boot Record ) ed ha una dimensione di 512 byte. Esso contiene il codice binario per avviare la macchina , chiamato **boot loader**, e la tabella delle partizioni. Una **partizione** è un numero di settori fisici riservati ad un sistema operativo per la gestione del proprio file system. Quando il computer viene avviato il BIOS si occupa di leggere l'MBR e di metterlo in memoria. Dopodiché, lascia il controllo della macchina al boot loader che si preoccupa di leggere la tabella delle partizioni e di decidere quale dei sistemi operativi avviare. Gli ultimi 2 byte dell'MBR formano i “byte flag” e valgono sempre 0x55AA.

Nell'offset 0x1BE dell'MBR, si trova la tabella delle partizioni formata da 4 elementi. Ogni elemento di questa tabella ha la seguente struttura :

```
struct MBR_Entry {
    unsigned char boot;
    unsigned char s_head;
    unsigned s_sector : 6;
    unsigned s_cylinder : 10;
    unsigned char type;
    unsigned char e_head;
    unsigned e_sector : 6;
    unsigned e_cylinder : 10;
    unsigned long start;
    unsigned long size;
};
```

Il primo campo **boot**, indica se la partizione è bootable, cioè se la partizione contiene un sistema operativo che deve essere avviato. I campi **s\_head**, **s\_sector** e **s\_cylinder** si riferiscono al settore fisico di partenza in formato CHS. Nel campo **type** vi è memorizzato un valore che identifica il tipo di file system assegnato alla partizione. I campi **e\_head**, **e\_sector** e **e\_cylinder** si riferiscono al settore fisico di fine partizione, sempre nel formato CHS. Il campo **start** punta al primo settore logico della partizione in formato LBA, mentre il campo **size** indica la dimensione, in settori fisici, della partizione.

Se il tipo di partizione vale 0, l'elemento di quella tabella non è una partizione valida. Esiste un particolare tipo di partizione chiamata **Extended Partition**, che non è proprio una partizione, ma che tramite il campo **start** di un **MBR\_Entry**, punta ad un settore fisico che contiene un'altra tabella delle partizioni dello stesso tipo. Questo tecnica viene adottata per partire un disco rigido in più di 4 possibili partizioni.



Figura 1

## Capitolo 3

# Breve storia del File System di Unix

---

Unix nacque intorno a gli anni '70 grazie a Thompson. Il primo file system montato da questo era il MinixFS, implementato da Linux Torvalds. Esso poteva memorizzare solo file aventi dimensione massima di 64 Mb e con un nome di file lungo al massimo 14 caratteri. Dopo molti anni i programmatori che lavoravano su Unix si accorsero che il MinixFS cominciava a essere obsoleto, e che allora bisognava implementare un nuovo file system.

Così fu assegnato il compito a Rémy Card, che insieme a Stephen Tweedie, realizzarono nell'Aprile del 1993 il nuovo file system di unix chiamato "The Extended File System". Il nuovo file system si basava sul vecchio file system MinixFS, con opportune modifiche molto importanti per la gestione dei file e delle directory. Infatti Card, riuscì a portare la dimensione massima di un file fino a 2 GB ( in teoria 4 TB ) e di estendere il nome di un file fino a 255 caratteri. Ma ancora qualcosa non andava nell'allocazione dei blocchi liberi ( i blocchi liberi, infatti, erano mantenuti in una lista ) e la politica di allocazione adottata portava a una troppa deframmentazione del file system. Pochi mesi dopo l'uscita del secondo file system per unix, Card decise di implementare un terzo file system : venne chiamato "The Extended Two File System". In sostanza la nuova implementazione risolveva i problemi nati nella seconda implementazione. Nel progetto venne introdotto anche un nuovo programmatore, Theodore Ts'O, che ebbe il compito di implementare tutte le utility di gestione del nuovo file system. Infine, questo nuovo file system venne adottato dal discendente di Unix, cioè Linux. Oggi è in fase di sviluppo una nuova versione del file system di Linux, chiamata "The Extended Three File System", in cui saranno aggiunte delle funzioni che permetteranno di realizzare una specie di diario di bordo del file system.

# The Second Extended File System

---

### 4.1 Struttura del file system

La partizione in cui risiede il file system ext2 viene divisa in blocchi logici di una dimensione ben definita. Le dimensioni di ogni blocco possono essere comprese fra 1024 e 4096 byte. Il file system, a sua volta, è diviso in alcune partizioni logiche chiamate **gruppi di blocchi** ( block groups ). In ogni gruppo vengono duplicate le informazioni critiche del file system, in modo da poterle recuperare in una situazione critica del file system.

Il blocco 0 non è usato da Linux e spesso contiene il codice per inizializzare il computer. All'offset 1024 del blocco zero ( e quindi se un blocco logico ha dimensione 1024, si parla del blocco 1 ) si trova il **superblocco** ( superblock ). Esso contiene informazioni critiche sulla struttura del file system, compreso il numero di i-node, il numero di blocchi del disco e l'inizio della lista dei blocchi liberi. La distruzione del primo superblocco rende il file system non leggibile.

Dopo il superblocco si trovano i **descrittori dei gruppi** ( group descriptor ). Praticamente è una tabella di descrittori di gruppo, dove ognuno di essi contiene informazioni relativi al blocco contenente la **block bitmap**, che è la mappa dei blocchi liberi e usati nel gruppo, all' **i-node bitmap**, che è la mappa degli i-node liberi e usati nel gruppo, ed infine alla **tabella degli i-node** ( i-node table), che è una struttura dati del kernel contenente tutti gli **i-node** dei file e delle directory aperte. Ogni i-node è una struttura che descrive esattamente ogni file, riportando informazione come la dimensione, i permessi, gli attributi e tutte le informazioni per localizzare tutti i blocchi del disco su cui si trovano i dati del file. Le directory vengono gestite mediante le **entrate di directory** ( directory entry ), che sono delle strutture dati che contengono i nomi dei file contenuti nella directory e il puntatore all'i-node del file stesso. Il file system prevede di riservare anche dei blocchi del file system, che vengono usati dal superuser, così da poter montare il file system anche se tutti i blocchi del file system sono stati allocati.

Un esempio di un blocco di gruppo si può osservare in figura 2. Passiamo adesso ad una descrizione più accurata delle strutture dati appena descritte. Le seguenti strutture sono scritte in C per favorire la lettura e la comprensione delle strutture dati.

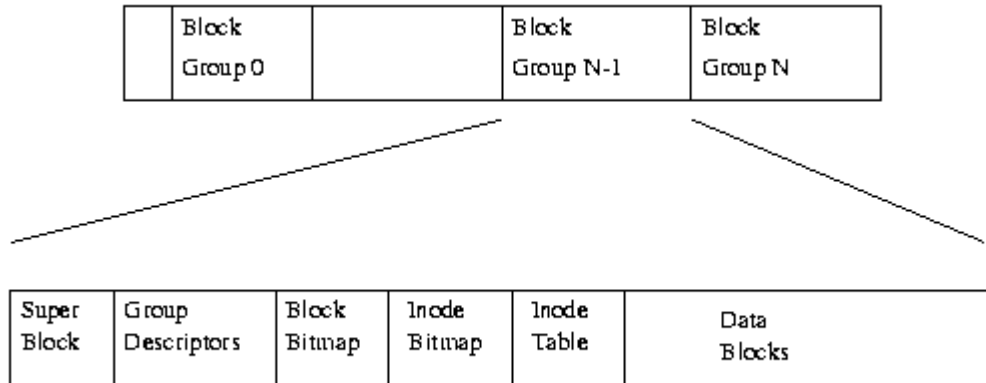


Figura 2

## 4.2 SuperBlock

Passiamo adesso ad esaminare una delle strutture più importanti di un file system ext2. Questa struttura, infatti, mantiene tutte le informazioni con cui è possibile leggere il file system. La struttura di un super blocco è la seguente :

```
struct ext2_super_block {
    unsigned long  s_inodes_count;
    unsigned long  s_blocks_count;
    unsigned long  s_r_blocks_count;
    unsigned long  s_free_blocks_count;
    unsigned long  s_free_inodes_count;
    unsigned long  s_first_data_block;
    unsigned long  s_log_block_size;
    long          s_log_frag_size;
    unsigned long  s_blocks_per_group;
    unsigned long  s_frags_per_group;
    unsigned long  s_inodes_per_group;
    unsigned long  s_mtime;
    unsigned long  s_wtime;
    unsigned short s_mnt_count;
    short         s_max_mnt_count;
    unsigned short s_magic;
    unsigned short s_state;
    unsigned short s_errors;
    unsigned short s_pad;
    unsigned long  s_lastcheck;
    unsigned long  s_checkinterval;
    unsigned long  s_reserved[238];
};
```

Esaminiamo adesso ogni campo del super blocco.

<b>Campo</b>	<b>Descrizione</b>
<code>s_inode_count</code>	mantiene il numero totale di i-node supportati dal file system ext2.
<code>s_blocks_count</code>	mantiene il numero totale di blocchi supportati dal file system ext2.
<code>s_r_blocks_count</code>	mantiene il numero totale di blocchi riservati al superuser
<code>s_free_blocks_count</code>	mantiene il numero totale di blocchi liberi del file system ext2
<code>s_free_inodes_count</code>	mantiene il numero totale di i-node liberi nel file system ext2
<code>s_first_data_block</code>	l'indice del primo blocco di informazioni. Di solito, quest'indice vale 1 per i file system ext2 che hanno la dimensione del blocco di 1024 byte, e vale 0 per le altre dimensioni di blocchi
<code>s_log_block_size</code>	usato per calcolare la dimensione di un blocco logico di dati. Il valore si ottiene ruotando a sinistra la cifra 1024 di <code>s_log_block_size</code> bit.
<code>s_log_frag_size</code>	usato per calcolare la grandezza dei frammenti logici
<code>s_blocks_per_group</code>	mantiene il numero totale di blocchi presenti in un gruppo
<code>s_frags_per_group</code>	mantiene il numero totale di frammenti presenti in un gruppo
<code>s_inode_per_group</code>	mantiene il numero totale di i-node presenti in un gruppo
<code>s_mtime</code>	mantiene la data dell'ultimo mount del file system
<code>s_wtime</code>	mantiene la data dell'ultimo
<code>s_mnt_count</code>	mantiene il numero di volte che il file system è stato montato
<code>s_max_mnt_count</code>	mantiene il numero massimo di volte che il file system può essere montato dopo che si sono presentati errori.
<code>s_magic</code>	il numero magico che permette l'identificazione del file system ext2. Esso vale sempre 0xEF53 in esadecimale.
<code>s_state</code>	mantiene lo stato del file system. Esso contiene valori come <code>EXT_VALID_FS</code> ( 0x0001 ) se non ci sono stati errori nell'ultima lettura/scrittura del file system. Se ci sono stati errori contiene il valore <code>EXT2_ERROR_FS</code> ( 0x0002 ).
<code>s_errors</code>	indica il tipo di errore
<code>s_pad</code>	non usato
<code>s_lastcheck</code>	mantiene il tempo dell'ultima analisi del file system
<code>s_checkinterval</code>	mantiene il tempo massimo possibile tra le analisi del file system
<code>s_reserved</code>	non usato

Tabella 1

### 4.3 Group Descriptor

Subito dopo il super blocco si trova la tabella dei descrittori dei gruppi. Il numero di descrittori di gruppo viene dato dal calcolo della formula  $(\text{sizeof}(\text{ext2\_group\_desc}) * \text{number of groups}) / \text{block size}$ . Ogni elemento di questa tabella ha il seguente formato :

```
struct ext2_group_desc
{
    unsigned long    bg_block_bitmap;
    unsigned long    bg_inode_bitmap;
    unsigned long    bg_inode_table;
    unsigned short   bg_free_blocks_count;
    unsigned short   bg_free_inodes_count;
    unsigned short   bg_used_dirs_count;
    unsigned short   bg_pad;
    unsigned long    bg_reserved[3];
};
```

Scopriamo adesso il significato di ogni campo.

Campo	Descrizione
<a href="#">bg_block_bitmap</a>	mantiene il puntatore ai blocchi logici contenenti la bitmap dei blocchi per quel gruppo
<a href="#">bg_inode_bitmap</a>	mantiene il puntatore ai blocchi logici contenenti la bitmap degli inode per quel gruppo
<a href="#">bg_inode_table</a>	mantiene il puntatore alla tabella degli i-node del gruppo
<a href="#">bg_free_blocks_count</a>	mantiene il numero dei blocchi liberi nel gruppo
<a href="#">bg_free_inode_count</a>	mantiene il numero di i-node liberi nel gruppo
<a href="#">bg_used_dirs_count</a>	mantiene il numero di inode allocati per le directory nel gruppo
<a href="#">bg_pad</a>	padding

Tabella 2

## 4.4 I-node

Passiamo adesso a descrivere una delle più importanti strutture del file system ext2. Un i-node è un descrittore di file, e contiene tutte le informazioni necessarie alla memorizzazione e alla lettura di un file. Tutti gli i-node sono memorizzati all'interno di un gruppo nella tabella degli i-node. Ogni gruppo può mantenere un numero massimo di i-node descritto nel super blocco. Per individuare il gruppo in cui un i-node si trova basta applicare la seguente formula :  $((\text{Numero inode} - 1) / \text{i-node per gruppo}) + 1$ . Esistono nel file system ext2 dei i-node che hanno una funzione speciale e sono riportati di seguito :

<b>Identificatore</b>	<b>Indice</b>	<b>Descrizione</b>
EXT2_BAD_INO	1	Blocchi danneggiati del disco
EXT2_ROOT_INO	2	Root Directory
EXT2_ACL_IDX_INO	3	ACL
EXT2_ACL_DATA_INO	4	ACL
EXT2_BOOT_LOADER_INO	5	Boot Loader
EXT2_UNDEL_DIR_INO	6	Directory eliminate
EXT2_FIRST_INO	11	Primo I-node libero

Tabella 3

La struttura di un i-node è la seguente :

```
struct ext2_inode {
    unsigned short i_mode;
    unsigned short i_uid;
    unsigned long i_size;
    unsigned long i_atime;
    unsigned long i_ctime;
    unsigned long i_mtime;
    unsigned long i_dtime;
    unsigned short i_gid;
    unsigned short i_links_count;
    unsigned long i_blocks;
    unsigned long i_flags;
    unsigned long i_reserved1;
    unsigned long i_block[15];
    unsigned long i_version;
    unsigned long i_file_acl;
    unsigned long i_dir_acl;
    unsigned long i_faddr;
    unsigned char i_frag;
    unsigned char i_fsize;
    unsigned short i_pad1;
    unsigned long i_reserved2[2];
};
```

Passiamo adesso ad analizzare campo per campo la struttura precedente

<b>Campo</b>	<b>Descrizione</b>
i_mode	mantiene gli attributi e permessi d'accesso del file ( vedi Tabella 6 )
i_uid	mantiene l'id del proprietario del file
i_size	mantiene la dimensione del file in byte
i_atime	mantiene la data dell'ultimo accesso al file
i_ctime	mantiene la data dell'ultima modifica della stessa struttura
i_mtime	mantiene la data dell'ultima modifica del file
i_dtime	mantiene la data di eliminazione del file
i_gid	mantiene l'id del gruppo
I_links_count	mantiene il numero di link che puntano a questo file
i_blocks	mantiene il numero di blocchi da 512 byte allocati
i_flags	mantiene informazione sul file ( vedi tabella 5 )
i_reserved1	riservato
i_block	mantiene i puntatori ai blocchi di dati ( vedi paragrafo 4.4.2)
i_version	mantiene la versione del file
i_file_acl	mantiene la control access list del file ( non usata )
i_dir_acl	mantiene la control access list della directory ( non usata )
i_faddr	mantiene il puntatore al blocco contenente il frammento del file
i_frag	mantiene numero di frammenti del file nel blocco
i_fsize	mantiene la dimensione la dimensione del frammento
i_pad	padding
i_reserved2	reserved.

Tabella 4

<b>Identificatore</b>	<b>Valore Esadecimale</b>	<b>Descrizione</b>
EXT2_SECRM_FL	0x0001	Eliminazione sicura. Quando è attivo, alla cancellazione del file vengono sovrascritti i blocchi del file con dati a caso.
EXT2_UNRM_FL	0x0002	Undelete. Quando è attivo, alla cancellazione vengono salvate le informazioni indispensabili per il recupero del file.
EXT2_COMPR_FL	0x0004	Compresso. I blocchi del file sono compressi e in lettura/scrittura bisogna utilizzare l'algoritmo di compressione.
EXT2_SYNC_FL	0x0008	Sincronismo. L'IO sul file avviene in modo sincrono invece che asincrono.

Tabella 5

#### 4.4.1 Attributi e permessi di un file

Il primo campo “i\_mode” di un i-node descrive tutti gli attributi e i permessi del file tramite un insieme di bit, precisamente 16, che seguono le regole riportate nella seguente tabella. Esso contiene attributi come il tipo di file ( link, file speciale a blocchi, un file speciale a caratteri,...), directory ed i permessi al file ( lettura/scrittura/ eseguibile) da parte dell’utente, del gruppo a cui appartiene il file e da parte di tutti gli utenti. Di seguito verranno mostrate le “maschere” che permettono di conoscere lo stato delle bandiere relative ad ogni attributo o permesso.

Identificatore	Valore Esadecimale	Descrizione
S_IFMT	F000	Flag di formato
S_IFSOCK	A000	socket
S_IFLNK	C000	Flag di link simbolico
S_IFREG	8000	Flag di file regolare
S_IFBLK	6000	Flag di file speciale a blocchi
S_IFDIR	4000	Directory
S_IFCHR	2000	Flag di file Speciale a caratteri
S_IFIFO	1000	FIFO
S_ISUID	0800	Flag di Super User ID
S_ISGID	0400	Flag di Super Group ID
S_ISVTX	0200	Sticky bit
S_IRUSR	0100	Flag di Lettura per l’utente proprietario
S_IWUSR	0080	Flag di Scrittura per l’utente proprietario
S_IXUSR	0040	Flag di Esecuzione per l’utente proprietario
S_IRGRP	0020	Flag di Lettura per il gruppo
S_IWGRP	0010	Flag di Scrittura per il gruppo
S_IXGRP	0008	Flag di Esecuzione per il gruppo
S_IROTH	0004	Flag di Lettura per altri utenti
S_IWOTH	0002	Flag di Scrittura per altri utenti
S_IXOTH	0001	Flag di Esecuzione per altri utenti

Tabella 6

#### 4.4.2 Blocchi di dati

Il campo “block” dell’i-node è un array di 16 elementi che contiene gli indirizzi dei blocchi logici del file. I primi 13 elementi di questo array contengono gli indirizzi effettivi dei blocchi logici del file, mentre i restanti puntano a loro volta a tre diversi blocchi : a **singola indirezione**, **doppia indirezione** e **trippla indirezione**. Nel primo caso, a singola indirezione, si punta ad una tabella ( il numero degli elementi della tabella corrisponde al rapporto tra la dimensione di un blocco logico e la dimensione di un puntatore a un blocco ) che contiene a sua volta degli indirizzi ai blocchi logici che contengono i dati del file. Nel secondo caso, a doppia indirezione, si punta ad una tabella che contiene indirizzi che puntano a blocchi che contengono altre tabelle di indirizzi che puntano alla fine ai blocchi logici del file. Nel terzo caso, a tripla indirezione, si punta ad una tabella che contiene

indirizzi di blocchi che puntano ad altre tabelle che contengono indirizzi di blocchi che, alla fine, puntano ad altre tabelle che contengono gli indirizzi dei blocchi logici che contengono i dati del file. In questo modo teoricamente, si riescono a creare file di dimensione fino a 4 Tb. La figura seguente aiuta a comprendere il meccanismo delle tabelle a singola, doppia e tripla indirezione.

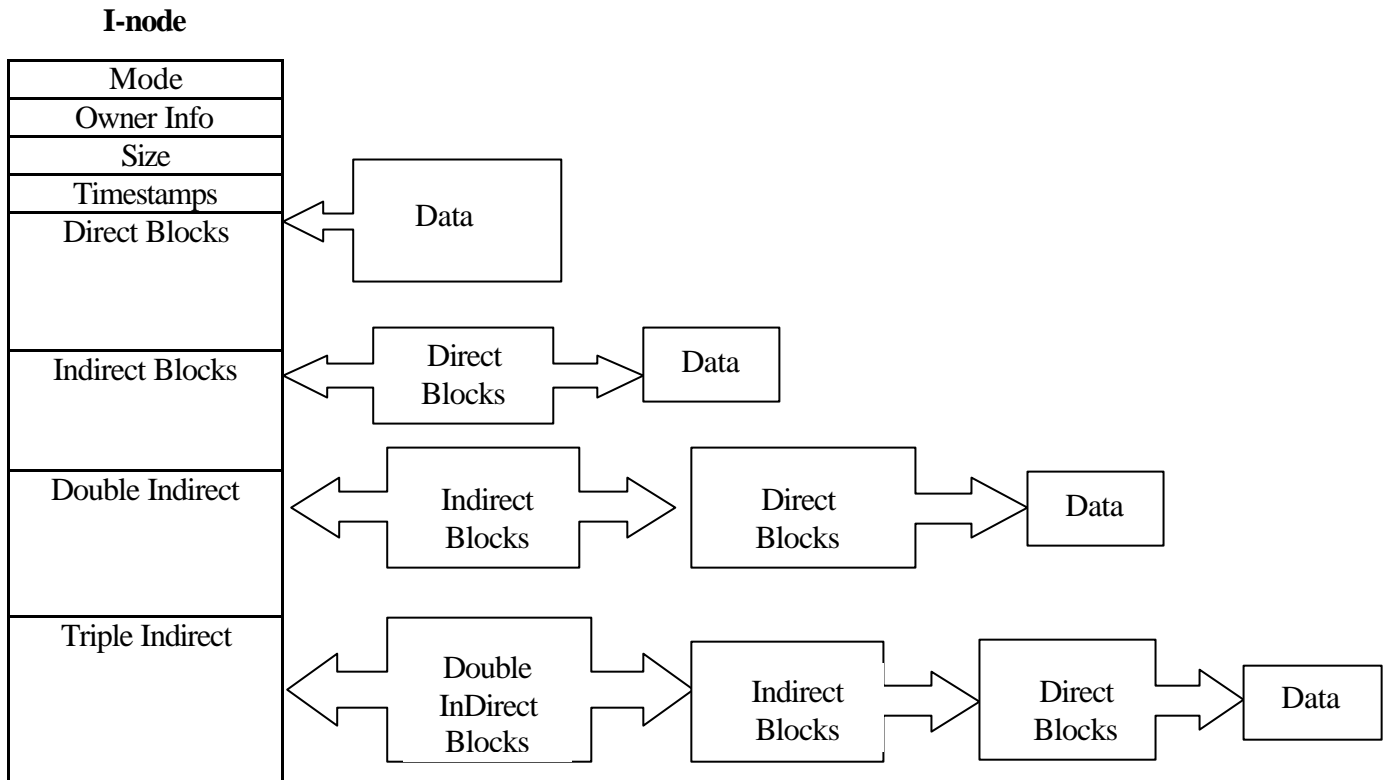


Figura 3

#### 4.4.3 Collegamenti ( Links )

Quando il percorso di un file all'interno di una directory è troppo lungo si suole utilizzare dei file speciali chiamati **link**. Un link, per il file system ext2, non è altro che un i-node che punta direttamente al file, ecco perché viene chiamato anche collegamento di tipo hardware. Infatti, se viene apportata una modifica al link viene immediatamente apportata al file a cui esso è collegato. Quando si cancella un link, però, non viene cancellato direttamente il file a cui è collegato, ma viene eliminato il file link.

Il campo "i\_links\_count" tiene traccia di quanti collegamenti si sono creati verso il file linkato, e viene decrementato quando un link su di esso viene cancellato.

## 4.5 Entrate delle directory

Le `dir_entry` (entrate di directory) non sono altro che una struttura in cui viene memorizzato i nomi dei file del contenuto di una directory. Le `dir_entry` si trovano all'interno dei blocchi di dati di un `i-node` che è una directory. La struttura generale delle `dir_entry` è la seguente :

```
struct ext2_dir_entry {
    unsigned long  inode;
    unsigned short rec_len;
    unsigned short name_len;
    char           name[255];
};
```

Questa struttura è molto importante poiché tramite essa si può tener traccia di tutti i nomi dei file e delle directory del file system ext2. Analizziamo adesso i campi di questa struttura.

<b>Campo</b>	<b>Descrizione</b>
<code>inode</code>	mantiene il puntatore all' <code>i-node</code> del file
<code>rec_len</code>	mantiene la dimensione della struttura <code>dir_entry</code>
<code>name_len</code>	mantiene la lunghezza del nome del file
<code>name</code>	contiene il nome del file

Tabella 7

Il campo "`rec_len`" serve per poter leggere tutti le `dir_entry` all'interno di un blocco. Quando la somma di tutti i campi `rec_len` di tutte le `dir_entry` raggiungono la dimensione massima del blocco logico, la `dir_entry` termina.

# Virtual File System

---

Nei sistemi Unix e Linux, il “file” è trattato come un oggetto, e sono definite per esso e per tutti gli oggetti come lui le stesse chiamate di sistema e gli stessi comandi. Questi funzionano indipendentemente dal supporto fisico che contiene l’informazione e dal modo in cui l’informazione è organizzata sul supporto.

Il VFS ( Virtual File System ) si occupa di realizzare, in modo indipendente dalla device, la memorizzazione e la gestione dei file.

Sia Unix che Linux gestiscono il file system in memoria allo stesso modo, attraverso delle strutture dati che somigliano a quelle dell’ext2 : il super-blocco, inode, directory e file. L’albero dei file che viene visto in un determinato momento dipende da come le differenti parti vengono assemblate; ogni parte in questo caso è rappresentata da una partizione di disco rigido o un altro dispositivo che viene “montato” nel sistema. Le strutture utilizzate dal VFS, somigliano molto a quelle dell’ext2 :

- Il “super blocco” è la struttura dati che contiene le informazioni riguardo al filesystem di cui fa parte. Linux è in grado di montare fino a 64 dispositivi tramite un vettore statico di 64 di tali strutture.
- Un “i-node” ( index-node ) è associato ad ogni file. L’inode contiene tutte le informazioni riguardanti un file tranne il suo nome ed i suoi dati.
- La “directory” è un file che associa i nomi dei file agli inode.
- Il “file” è qualcosa associato ad un inode. Di solito i file sono aree di dati, ma possono anche essere directory, dispositivi, FIFO o socket.

Il problema di poter gestire differenti formati di dati in maniera trasparente è stato affrontato trasformando i super-blocchi, gli inode e i file in “oggetti”: ogni oggetto è costituito da un insieme di operazioni che possono essere usate su di lui. L’ idea del VFS, perciò, è implementata tramite insieme di operazioni che agiscono su tali oggetti. Ogni oggetto include una struttura dati che elenca le operazioni per agire su di lui, e la maggior parte di tali operazioni, scritte in linguaggio C, ricevono come argomento un puntatore all’oggetto stesso, permettendo la modifica di quest’ultimo .

In pratica, un super-blocco contiene un campo “struct super\_operations \*s\_op”, un inode contiene “struct inode\_operations \*i\_op” ed un file contiene “struct file\_operations \*f\_op”.

La gestione dei dati e la bufferizzazione viene effettuata dal kernel, ed è indipendente dal formato di dati immagazzinati: ogni comunicazione con il supporto di immagazzinamento avviene attraverso una delle strutture “operations”. Infine, il “tipo di file system” è il modulo software che si occupa di tradurre le operazioni sull’effettivo meccanismo di immagazzinamento dei dati.

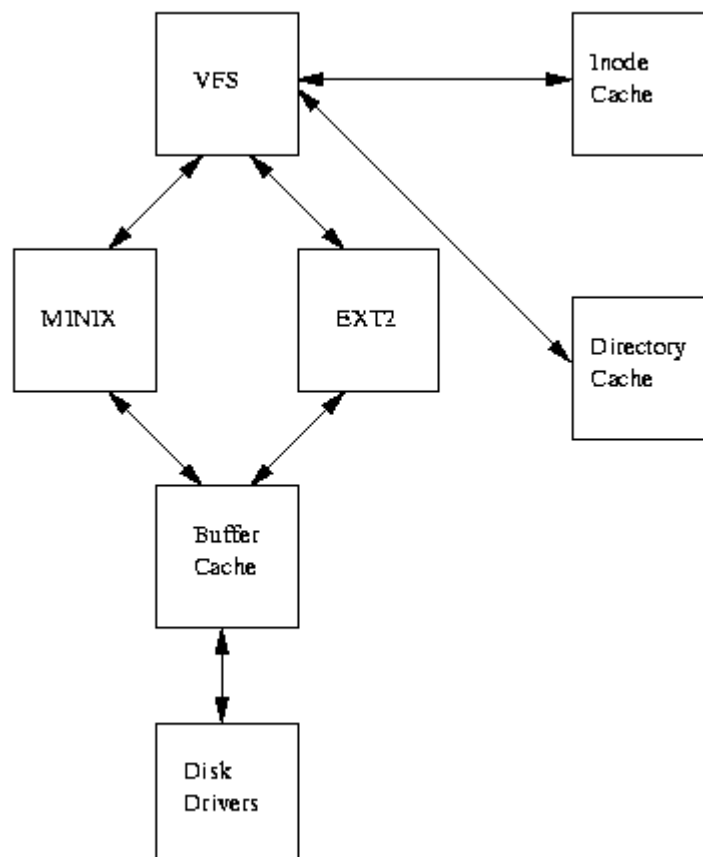


Figura 4

# Mantenimento del file system

---

Tutti i sistemi operativi che gestiscono un file system, mettono a disposizione dell'utente un package di utilità che permettono di gestire, controllare e riparare il proprio file system. Unix e Linux mettono a disposizione un insieme di utilità per la gestione del file system ext2, che verranno discusse qui di seguito :

- **mount** : monta un file system. Tutti i file(s) in un sistema Unix/Linux sono disposti in un grande albero, la cui radice è '/'. Il comando mount serve per “attaccare il file system trovato nella device specificata al grande albero già esistente.
- **umount** : smonta un file system. “Stacca” un file system già “attaccato” al grande albero.
- **mke2fs** : usato per creare un Linux Extended Second File System su una device. Esso crea e inizializza tutte le strutture fondamentali del file system ext2.
- **dumpe2fs** : stampa a video tutti i campi del super blocco e i descrittori di gruppo in formato leggibile di un file system ext2.
- **e2fsck** : controlla l'integrità di un file system ext2 e lo ripara in caso di errori.
- **tune2fs** : permette di cambiare i valori dei campi del super blocco di un file system ext2. Questa operazione può avvenire solo in caso di un file system non montato.
- **debugfs** : permette di esaminare attraverso dei comandi il file system ext2. Si usa per aggiungere o cambiare lo stato ed il contenuto di un file system ext2 senza bisogno di montarlo.

## **Bibliografia**

Andrew S. Tanenbaum, *I moderni sistemi operativi*, Jackson Libri

Alessandro Rubini, *Virtual File System*, <http://linux.nuvoli.to.it/pj/pj9704/vfs.html>

David A Rusling, *The file system*, <http://www.linuxdoc.org/LDP/tlk/fs/filesystem.html>

*Dischi rigidi, BIOS e sistema operativo*  
<http://www.melograno.net/talpanet/ToolBox/int/hw/dischi.html>

Louis-Dominique Dubeau, *Analysis of the Ext2fs structure*,  
[http://step.polymtl.ca/%7Eldd/ext2fs/ext2fs\\_2.html](http://step.polymtl.ca/%7Eldd/ext2fs/ext2fs_2.html)

John Newbigin, *John's spec of the second extended filesystem*,  
<http://uranus.it.swin.edu.au/~jn/linux/>

Card, Rémy, Ts'o, Theodore e Tweedie Stephen, *Linux File Systems*,  
<ftp://ftp.ibp.fr/pub2/linux/packages/ext2fs/ext2-1.eps.gz>

## **Realizzata da**

Salvatore Davide Rapisarda

Università degli Studi di Catania

Facoltà di Informatica --- Anno 2000 / 2001

Indirizzo di posta elettronica <[blackmicio@tiscalinet.it](mailto:blackmicio@tiscalinet.it)>