Lezione 3 - La struttura del Filesystem

26 novembre 2006

Alessio Checcucci

Riccardo Aldinucci

Q.It Universita' degli Studi di Siena

1 La struttura del filesystem

Il filesystem e' quella parte di sistema operativo che permette all'utente di interagire con i dati immagazzinati nei supporti di memoria secondaria, esso e' costuito dai metodi e dalle strutture dati che un sistema operativo usa per tenere traccia dei file in un disco o in una partizione. Gli obiettivi che si deve porre un filesystem moderno che deve interagire con dispositivi hardware di capacita' sempre crescente sono l'efficienza nel reperire le informazioni, l'integrita' dei dati presenti sui supporti di memorizzazione sia dovuti a crash che ad accessi non autorizzati ai sistemi.

Un filesystem organizza i dati dividendo il supporto di memorizzazione in una serie di blocchi di dimensione finita. All'interno dei blocchi vengono immagazzinate tre tipi di informazioni:

- dati archiviati dall'utente e dalle applicazioni che girano sul sistema.
- dati riguardanti l'organizzazione della struttura gerarchica del sistema di memorizzazione visibili agli utenti.
- dati riguardanti la struttura fisica del filesystem (metadata), in genere non visibili agli utenti.

L'albero delle directory e' organizzato in modo che ciascun blocco sia utilizzato come nodo o come foglia. I nodi sono quei blocchi che memorizzano tutte le informazioni riguardanti la struttura gerarchica del filesystem. Essi conterranno i riferimenti ad altri nodi di livello inferiore oppure riferimenti alle foglie, che sono in definitiva i blocchi utilizzati per immagazzinare i dati.

Esistono poi dei blocchi speciali destinati a contenere solo metadata (per es. il superblocco) che esulano dall'organizzazione gerarchica delle directory.

1.1 Garanzia di integrita' dei dati

L'integrita' dei dati susseguente ad un crash del sistema puo' riguardare corruzione sia dei dati che dei metadata. Il sistema deve allora prevedere un metodo che permetta di gestire in sicurezza il danneggiamento del superblocco. Questo in genere viene ottenuto conservandone piu' copie sparse nel filesystem, contenenti una quantita' di informazioni sufficienti a riportare il filesystem ad uno stato consistente.

Un'altra soluzione utilizzata per garantire l'integrita' sono le bitmap di allocazione, ovvero record di dimensione variabile dove lo schema dei bit rappresenta l'allocazione o meno dei data block.

In caso di crash del sistema, il filesystem (che sara' stato opportunamente marcato) verra' automaticamente sottoposto ad un check. Naturalmente un'operazione di questo genere su supporti di dimensioni generose comporta un notevole dispendio di tempo e un lungo downtime. Per ovviare a questi problemi e' stato introdotto nei filesystem un log delle transazioni (journal), in cui vengono registrate tutte le operazioni di I/O prima che vengano effettivamente realizzate. Al riavvio dopo un crash e' quindi sufficiente un roll forward delle transazioni complete presenti nel journal per avere di nuovo un filesystem consistente.

Per quel che riguarda la garanzia di integrita' dei dati da modifiche fraudolente, il sistema operativo deve applicare i criteri di protezione basati sull'appartenenza degli oggetti ad utenti e gruppi.

Un altro aspetto importatante che si ha nei sistemi multiutente e' la limitatezza dello spazio disponibile e la necessita' di assegnare uno spazio massimo (quota) che ogni utente puo' occupare.

1.2 I layer del kernel per i filesystem

Tutte le richieste di accesso ai filesystem vengono intercettate dal kernel, in particolare l'interfaccia verso lo User Space e' costituita dal layer livello piu' alto ovvero il Virtual Filesystem Switch. Quest'ultimo associa ad ogni richiesta il codice opportuno a gestirle a seconda del filesystem a cui si deve accedere. Tutte le richieste di accesso ai dati sono gestite per mezzo di un buffer dati virtuale detto buffer cache. Le informazioni in uscita dalla buffer cache vanno ad interagire con i driver dei vari filesystem che comandano direttamente i dispositivi fisici o quelli mappati per mezzo del Device Mapper.

La buffer cache Sebbene non faccia parte dei layer veri e propri del filesystem, il meccanismo di buffer cache e' stato introdotto per limitare le attese da parte dei processi per i device lenti a leggere e scrivere dati. In questo senso sarebbe controproducente scrivere grosse porzioni di dati in un'unica soluzione, mentre e' piu' efficiente compiere scritture limitate ad intervalli di tempo regolari, in modo da limitare l'impatto delle operazioni di I/O sulla rapidita' dei processi utente.

La buffer cache e' quindi una disk cache che memorizza buffer.[1] Il kernel mantiene molte informazioni su ogni buffer, incluso un "dirty" bit che indica quando il buffer e' stato alterato in memoria e deve essere scritto sul dispositivo oltre ad un timestamp che indica quanto il buffer deve essere tenuto in memoria prima di essere scritto. La dimensione della buffer cache e' variabile e le pagine sono allocate quando un nuovo buffer deve essere creato e nessuno e' disponibile. Quando la memoria di sistema diventa scarsa i buffer sono rilasciati e le pagine riciclate.

La buffer cache e' sostanzialmente costituita da:

- Un set di buffer heads che descrivono la buffer cache
- Una hash table che aiuta il kernel a rintracciare il buffer head che descrive il buffer associato ad un certo dispositivo e ai block number

1.3 Il VFS

Una delle chiavi del successo di Linux e' la sua abilita' a coesistere con altri sistemi. E' possibile montare in modo trasparente dischi o partizioni che contengono formati di file usati da molti altri sistemi operativi sia di ampia diffusione che dedicati a particolari contesti.

Linux gestisce il supporto ad una molteplicita' di tipi di dischi nello stesso modo utilizzato da altre varianti di Unix, per mezzo del concetto di *Virtual Filesystem*.

L'idea di fondo del Virtual Filesystem e' che gli oggetti che internamente rappresentano file e filesystem nella memoria del kernel portano con se una notevole quantita' di informazioni, c'e' un campo o una funzione che supporta

[1] Un buffer e' un area di memoria che contiene un disk block.

ogni operazione che sia possibile compiere su un filesystem supportato da Linux. Per ogni funzione che venga invocata, il kernel sostituisce la reale funzione che supporta il filesystem su cui andremo ad operare.

1.3.1 Il ruolo del VFS

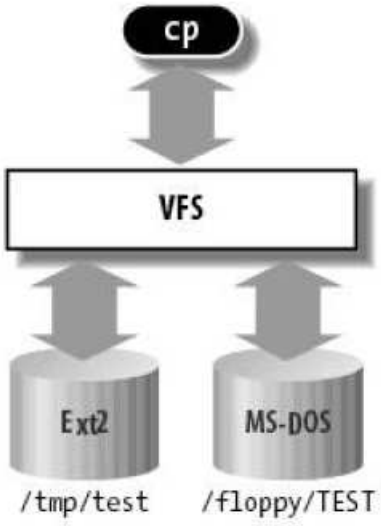
Il Virtual Filesystem (detto anche Virtual Filesystem Switch o VFS) e' un software layer del kernel che gestisce tutte le system call relative ad un filesystem Unix standard. La sua caratteristica principale e' quella di fornire un'interfaccia comune per vari tipi di filesystem.

Prendiamo per esempio il comando:

cp /floppy/TEST /tmp/test

Dove /mnt/floppy e' il mount point di un filesystem FAT (MS-DOS) e /tmp e' un comune ext2 filesystem.

Il VFS e' un layer di astrazione fra le applicazioni e l'implementazione dei filesystem. Il programma cp quindi non ha nessuna necessita' di conoscere i formati dei filesystem su cui andra' ad operare, al contrario esso interagisce con il VFS per mezzo di alcune system call generiche, eseguendo il seguente codice:




```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
       O_WRONLY|O_CREAT|O_TRUNC, 0600);
do {
    i = read(inf, buf, 4096);
    write(outf, buf, i);
} while (i);
close(outf);
close(inf);
```

I filesystem supportati dal VFS si dividono in tre grandi categorie:

- **Filesystem disk based**: sono quelli che gestiscono lo spazio di memorizzazione disponibile in un disco locale o in un dispositivo che emula un disco. Alcuni esempi sono:
  - Filesystem nativi in Linux, come ext2, ext3 e ReiserFS.
  - Filesystem di altre varianti Unix, come quelli di derivazione SysV, UFS, Minix e Veritas VxFS.
  - Filesystem Microsoft come MS-DOS, VFAT e NTFS.

  - ISO9660 per i CD e UDF per i DVD.
  - Altri filesystem proprietari come: HPFS, Apple HFS, Amiga FFS e Acorn (ADFS).
  - Filesystem journaled provenienti da altri Unix: SGI Irix XFS e IBM AIX JFS.

- **Network Filesystem**: questi permettono un semplice accesso ai file situati su altre macchine collegate in rete. Alcuni di essi sono:
  - NFS, Coda e AFS
  - CIFS (Common Internet FileSystem)
  - NCP (Novell Netware Core Protocol)

- **Filesystem speciali**: Sono filesystem che non gestiscono alcuno spazio ne' locale ne' remoto. A questa categoria appartengono per esempio /proc e /sys.

Vale la pena di ricordare che le directory Unix costituiscono un albero che si origina nella root (/) directory. La root directory e' contenuta nel root filesystem. Tutti gli altri filesystem possono essere *montati* sulle subdirectory del root filesystem. Quando un filesystem viene montato su una directory il contenuto originario della directory non sara' piu' accessibile, ma tornera' ad esserlo appena esso verra' smontato.

Un filesystem disk based in genere si trova su un device a blocchi. Un'utile caratteristica del VFS e' quella di gestire anche device a blocchi virtuali (per es. i loop device) che permettono di montare i filesystem contenuti nei file.

1.3.2 Il Common File Model

L'idea chiave del VFS consiste nell'introduzione di un *common file model* capace di rappresentare tutti i filesystem supportati. Il modello copia esattamente il file model tradizionale di Unix.

Ogni specifico filesystem dovra' quindi tradurre la propria organizzazione fisica nel VFS common file model.

Per esempio i filesystem come FAT non trattano le directory come file, al contrario di quello che si aspetterebbe VFS. Quindi sara' necessario che l'implementazione Linux di FAT sia tale da costruire "al volo" i file corrispondenti alle directory. Naturalmente questi file esistono solo come oggetti nella memoria del kernel.

Piu' concisamente, il kernel non puo' avere codificate le funzioni generiche di accesso ai filesystem. Al contrario deve usare un puntatore per ognuna di queste operazioni. Esso referenziera' la funzione corretta per il particolare filesystem a cui stiamo accedendo.

Un modo di vedere il common file model e' con l'approccio Object-oriented, dove un oggetto definisce sia una struttura dati che i metodi che operano su di essa.

Il common file model e' costituito dai seguenti oggetti:

- *The superblock object*: immagazzina informazioni sul filesystem montato. Per i disk based filesystem, in genere coincide con un filesystem control block immagazzinato su disco.

- *The inode object*: immagazzina informazioni generali su un file. Per i disk based filesystem, in genere coincide con un filesystem control block immagazzinato su disco. Ad ogni inode viene associato un *inode number* che identifica univocamente il file nel filesystem.
- *The file object*: immagazzina informazioni circa l'interazione fra un file aperto e un processo. Questa informazione esiste solo nella memoria del kernel durante il periodo in cui il processo ha accesso al file.
- *The dentry object*: immagazzina informazioni a riguardo del collegamento di una directory (che e' un particolare nome di file) con il corrispondente file. Ogni disk based filesystem immagazzina questa informazione in modo particolare.

Oltre a fornire un'interfaccia comune a tutte le implementazioni di filesystem, il VFS ha un altro ruolo importante realtivo alle performance del sistema. I dentry objects piu frequentemente utilizzati sono contenuti nella *dentry cache* che accelera la traduzione del pathname di un file nell'inode dell'ultima voce del pathname.

Filesystem Type Registration Spesso l'utente configura il kernel Linux per riconoscere i filesystem necessari ad operare con il sistema al boot, ma il codice per un filesystem non deve essere necessariamente incluso nell'immagine statica del kernel, ma puo' essere caricato come modulo. Il VFS deve tenere traccia di tutti i tipi di filesystem il cui codice e' incluso nel kernel in un certo momento. Questo viene fatto per mezzo del *Filesystem Type Registration*.

1.3.3 Gestione del filesystem

Come ogni Unix tradizionale, Linux utilizza il *system's root filesystem*, cioe' il filesystem direttamente montato dal kernel durante la fase di boot e che contiene gli script per l'inizializzazione del sistema e i programmi essenziali.

Gli altri filesystem possono essere montati successivamente sulle directory di preesistenti filesystem. Ogni filesystem avra' la propria root directory. La directory su cui il filesystem viene montato si chiama *mount point*. Un filesystem montato e' il figlio del filesystem a cui appartiene la directory che costituisce il mount point.

Namespaces Negli Unix tradizionali c'e' un solo albero di filesystem montati che si originano dal system's root filesystem. Ogni processo puo' potenzialmente accedere a ogni file a patto di conoscerne il path.

Linux 2.6 introduce un meccanismo piu' fine, ogni processo puo' avere il proprio albero di filesystem montati, detto *namespace* del processo. Questo viene assegnato ai processi creati con clone(). Il processo a questo punto opera sul proprio namespace, tutte le operazioni di mount e umount saranno relative solo al namespace e visibili solo ai processi che ne condividono il namespace.

Mounting Negli Unix tradizionali un filesystem puo' essere montato su un solo mount point. In Linux la cosa e' diversa, e' possibile montare lo stesso

filesystem piu' volte e quindi e' possibile accedere alla sua root directory attraverso piu' mount point, anche se il filesystem resta unico (e' unico il superblocco).

Il mounting di un generico filesystem viene effettuato per mezzo della system call mount(), la sua routine di servizio sys_mount() viene invocata con i seguenti parametri:

- Il pathname del device file che contiene il filesystem o NULL se non e' richiesto
- Il pathname del mount point
- Il tipo di filesystem (che deve essere registrato)
- I mount flags
- Un puntatore a una struttura che dipende dal filesystem (puo' essere NULL)

Mounting del root filesystem Montare il root filesystem e' una parte cruciale della procedura di inizializzazione del sistema. E' una procedura abbastanza complessa, perche' il kernel permette di avere il root filesystem in una quantita' disparata di device diversi.

Nel caso classico in cui essa sia in una partizione dell'hard disk, durante il boot il kernel recupera il major number del device che contiene il filesystem (il paramentro viene salvato a compile time o passato dal boot loader). Oltre a questo sono considerate le mount flags che erano state salvate nel kernel col comando rdev, oppure passate sempre dal bootloader (*rootflags*).

A questo punto l'operazione avviene in due passi:

- Il kernel monta lo special *rootfs* filesystem. Che si limita a fornire la directory vuota che serve da mount point iniziale.
- Il kernel monta il vero root filesystem sulla directory vuota.

Il motivo per cui il kernel monta il rootfs prima del filesystem definitivo e' dovuto al fatto che spesso il kernel monta e smonta vari root filesystem prima di arrivare a quello definitivo e questo e' molto semplificato dall'uso di rootfs.

1.4 I filesystem

1.4.1 Il filesystem ext2

Il filesystem ext2 e' il filesystem nativo di Linux ed e' utilizzato, oppure e' stato usato, da qualsiasi utente Linux. Come tutti i filesystem Unix, anche ext2 aderisce agli standard POSIX compliant, sebbene abbia una sua peculiare implementazione.

La prima versione del file system Linux era basato su Minix. Con l'evoluzione del sistema operativo venne introdotto un filesystem nativo detto *Extended Filesystem (ext)*, che aveva molte caratteristiche interessanti, ma prestazioni deludenti. Il *Second Extended Filesystem (ext2)* e' stato introdotto nel 1994, oltre ad alcune nuove caratteristiche, si e' dimostrato performante e robusto.

Fra le caratteristiche salienti possiamo annoverare:

- La possibilita' di scegliere la dimensione dei blocchi (1KB, 2KB o 4KB) in base alla dimensione stimata dei file che dovranno essere memorizzati. Con piccoli file e' meglio scegliere blocchi piccoli per limitare la frammentazione interna, mentre blocchi piu' grandi sono piu' efficienti in caso di file di grandi dimensioni.
- La possibilita' di scegliere il numero di Inode e quindi proporzionalmente il numero dei file memorizzabili, massimizzando lo spazio effettivamente utilizzabile.
- Il filesystem organizza i disk block in gruppi. Ogni gruppo include data blocks e inode in tracce adiacenti. In questo modo si puo' accedere ai file memorizzati in un singolo block group minimizzando il disk seek time.
- Il filesystem prealloca dei data blocks a normali file prima che essi li utilizzino effetivamente. Questo permette di ridurre la frammentazione.
- Supporto per i fast symbolic link. Se il pathname del link e' lungo meno di 60 caratteri, viene memorizzato nell'Inode e quindi tradotto senza accedere a nessun data block.
- Una strategia di file-updating particolarmente curata, che minimizza l'impatto dei crash di sistema.
- Supporto per i check automatici di consistenza a boot time. Questi check possono essere attivati da una serie di condizioni, come per esempio il numero di mount o il tempo trascorso dall'ultimo check.
- Supporto per file immutabili e file append only che nessuno (nemmeno il superuser) puo' prevalicare.
- Compatibilita' sia con le semantiche SysV che con BSD per il GroupID di un nuovo file. In SysV il nuovo file assume il GID del processo che lo ha creato, mentre in BSD quello della directory in cui viene creato.

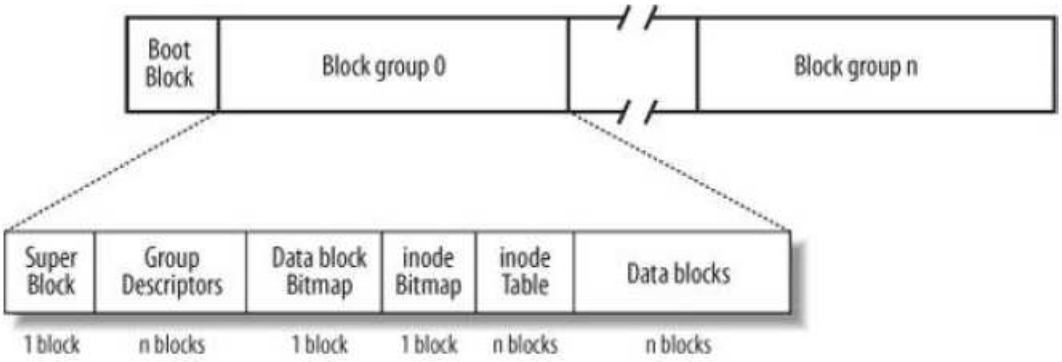
1.4.2 Le disk data stuctures di ext2

Il primo blocco di ogni partizione ext2 non e' mai gestito da ext2, ma e' riservato per il partition boot sector. Il resto della partizione ext2 e' diviso in *block groups*, ognuno dei quali ha una particolare struttura. Alcune delle strutture dati presenti in un block group devono entrare esattamente in un blocco, mentre altre possono richiedere piu' blocchi. Tutti i block group nel filesystem hanno stessa dimensione e sono memorizzati sequenzialmente, in questo modo il filesystem puo' derivare la locazione del block group semplicemente dal suo indice.

I Block group riducono la frammentazione dei file, visto che il kernel cerca di mantenere i data block di un file nello stesso block group finche' e' possibile. Un block group e' costituito da:

- La copia del filesystem superblock (1 blocco)
- La copia del gruppo dei descrittori dei block group (n blocchi)
- La data block bitmap (1 blocco)

- La inode bitmap (1 blocco)
- Un gruppo di inode (inode table)(n blocchi)
- I data block (n blocchi)



Se un blocco non contiene alcuna informazione utile esso e' vuoto.

Sia il superblock che il gruppo dei descrittori dei block group sono duplicati in ogni block group. In realta' solo le copie presenti nel block group 0 sono utilizzate dal kernel, mentre le altre sono lasciate immutate. Quando viene eseguito un check di coerenza del filesystem esso fara' riferimento alle copie presenti nel blocco 0. Se esse risultassero corrotte, l'amministratore di sistema puo' specificare una copia alternativa, sebbene vecchia. In genere comunque le copie ridondanti contentengono sufficienti informazioni da pemettere di riportare la partizione ad uno stato consistente.

Il numero di block groups dipende dalla dimensione della partizione e dalla dimensione del blocco del filesystem. La limitazione e' insita nel fatto che la block bitmap e' contenuta in un solo blocco. Quindi per un blockgroup ci potranno essere al massimo $8 \times b$ blocchi, dove b e' la dimensione del blocco in byte. Quindi il numero totale di block group e' circa $s/(8 \times b)$ dove s e' la dimensione della partizione in blocchi.

La Inode Table La tabella degli Inode consiste di una serie di blocchi consecutivi, ognuno dei quali contiene un numero predefinito di Inode. Il numero del primo blocco della Inode table viene memorizzato nel group descriptor.

Tutti gli Inode hanno la stessa dimensione (128bytes). Quindi un blocco da 4096 byte ne contiene 32. Per sapere quanti blocchi sono occupati dalla inode table, basta dividere il numero totale degli inode del gruppo per il numero di inode per blocco.

Ogni Inode presenta tutta una serie di campi POSIX, che sono quelli richiesti dal VFS, oltre ad altri campi specifici che hanno a che fare con l'allocazione dei blocchi.

All'interno dell'Inode esiste il campo *i_ blocks*, che e' un array di (15) puntatori ai blocchi che identificano i data blocks allocati al file.

Il numero dell'inode non viene memorizzato nella struttura, in quanto puo' essere ricavato dal blockgroup number e dalla posizione relativa nella inode table.

Allocazione dei data block per vari tipi di file

- Regular file: i blocchi di un regular file vengono allocati solo quando esso inizia ad avere dati, quindi esso viene inizialmente creato senza blocchi allocati. Un file puo' essere svuotato per mezzo della system call *truncate()*. Che poi e' quello che viene fatto dalla shell quando si impartisce un comando *>filename*.

- Directory: sono implementate in ext2 come un particolare tipo di file che memorizza il nome del file insieme al proprio numero di inode. Le strutture contenute in questo particolare file hanno lunghezza variabile, in quanto il nome del file puo' contenere fino a 255 caratteri, viene comunque fatto un pad per avere una struttura multipla di 4.

- Symbolic link: se il pathname di un link simbolico ha fino a 60 caratteri, esso viene memorizzato all'interno dell'Inode nella struttura i_blocks che contiene 15 interi da 4 byte. Se il pathname e' piu' lungo, allora un singolo data block deve essere allocato.

- Device file, pipe e socket: nessun data block e' necessario per questi tipi di file, tutte le informazioni relative sono memorizzate nell'inode.

1.4.3 Amministrazione dello spazio disco

Ext2 amministra lo spazio su disco allocando e deallocando inode e data blocks. I principali problemi che deve affrontare sono:

- Deve essere compiuto ogni sforzo per evitare le frammentazione del file, cioe' pezzi di informazione allocati in data block non adiacenti. Questa comporta una maggiore attivita' del disco e un deperimento delle prestazioni.

- L'amministrazione dello spazio disco deve essere efficiente dal punto di vista del tempo. Il kernel dovrebbe essere capace di reperire da un file offset il corrispondente numero di blocco nella partizione ext2. Ovvero limitare al massimo l'accesso alle tabelle di indirizzamento presenti su disco.

L'indirizzamento dei data block Ogni file non vuoto e' costituito da un gruppo di data blocks che possono essere referenziati sia dalla loro posizione all'interno del file (*file block number*) che dalla loro posizione nella partizione (*logical block number*).

Come abbiamo visto nell'Inode e' presente il campo i_block, costituito da un array di 15 strutture che puntano ai data blocks. Queste strutture possono avere varia natura:

- I primi 12 elementi corrispondono ai primi 12 data blocks del file (file block number: 0-11)

- Il componente di indice 12 contiene il logical block number di un blocco che che rappresenta un array di secondo ordine di logical block numbers. Essi corrispondono ai file block che vanno da 12 a *b/4+11*, dove b e' la dimensione del blocco in byte.

- Il componente di indice 13 contiene il logical block number di un blocco che rappresenta un array di secondo ordine, a sua volta le entry di questo blocco puntano ad un array di terzo livello, che immagazzina i logical block number corrispondenti ai file block number da *b/4+12* a $(b/4)^2+b/4+11$.
- Infine il componente di numero 14 rappresenta un terzo livello di indirezione, che permette di mappare i logical block numbers nei file block di ordine $(b/4)^3+(b/4)^2+b/4+11$.

Per avere un'idea delle dimensioni dei file che un filesystem ext2 tradizionale (nelle versioni piu' recenti molti limiti sono stati rimossi) basta guardare la seguente tabella:

Block size	**Direct**	**1-Indirect**	**2-Indirect**	**3-Indirect**
1024	12KB	268KB	63.55MB	2GB
2048	24KB	1.02MB	513.02MB	2GB
4096	48KB	4.04MB	2GB	-

1.4.4 I limiti di ext2

I limiti e le nuove caratteristiche che dovrebbero essere presenti in ext2 sono principalmente:

- Sensibilita' alla block fragmentation. In genere sui moderni dispositivi, viene sempre specificata la dimensione del blocco piu' grande. Questo comporta che i piccoli file sprechino molto spazio. Il problema puo' essere risolto permettendo a diversi file di essere memorizzati in frammenrti diversi dello stesso blocco.
- Gestione trasparente dei file compressi e crittati.
- Logical deletion: ovvero la possibilita' di rendere l'utente capace di recuperare i dati cancellati per sbaglio.
- Journaling: che e' stato gia' implementato nel filesystem ext3.

1.5 Il filesystem Ext3

Il filesystem ext3 e' un nuovo filesystem Linux progettato per essere totalmente compatibile con ext2, ma presentare come ulteriore caratteristica quella del journaling. Vale la pena di dire che la maggior parte delle distribuzioni odierne prevedono ext3 come default filesystem.

Un filesystem ext3 puo' essere smontato e rimontato come ext2 (naturalmente rinunciano al journaling) senza nessun problema. Anche un filesystem ext2 puo' essere convertito al journaling molto semplicemente (*tune2fs*) e successivamente rimontato come ext3.

1.5.1 Il journaling

Come abbiamo visto in caso di crash del sistema, i dirty buffer che sono i memoria e ancora (o solo parzialmente) non sono stati salvati su disco lascerebbero il filesystem in uno stato inconsistente. Per prevenire questo comportamento tutti gli Unix filesystem vengono controllati prima di essere montati. Se non sono stati correttamente smontati (campo *s_ mount_ state* del superblocco), un programma di lunga durata viene avviato per eseguire il controllo.

Lo scopo del journaling e' quello di evitare questa attivita' di controllo, andando a guardare una particolare area disco che contiene le ultime operazioni di scrittura, detto *journal*. In questo modo rimontare un filesystem dopo un crash diventa una questione di pochi secondi.

1.5.2 L'implementazione di ext3

L'idea che sta dietro a ext3 e' quella di eseguire ogni operazione di cambiamento di alto livello del filesystem in due passi. Per prima cosa una copia dei blocchi che devono essere scritti viene salvata nel journal, poi quando il data transfer nel journal e' completo (*data commited to the journal*), i data block sono scritti sul filesystem. Quando l'I/O sul filesystem termina (*data commited to the filesystem*) le copie presenti nel journal possono essere sovrascritte.

Durante il recover del filesystem dopo il crash, l'utilita' *e2fsck* distingue due casi:

- *Il crash e' avvenuto durante una commit sul journal*: i blocchi relativi al cambiamento di alto livello o mancano dal journal o sono incompleti. In ogni caso vengono ignorati. Quindi le modifiche sono perse, ma il filesystem e' consistente.

- *Il crash e' avvenuto dopo la commit sul journal*: le copie dei blocchi sono valide e e2fsck le scrivera' sul filesystem. In questo caso tutte le modifiche vengono applicate e il filesystem sara' consistente e aggiornato.

In genere i filesystem journaled non copiano tutti i blocchi nel journal. Come sappiamo ogni filesystem e' costituito da due tipi di blocchi, quelli che contengono i *metadata* e quelli quelli che contengono i *dati*. Nel caso dei filesystem ext2/3 ci sono 6 tipi di metadata: superblock, group block decriptor, inodes, data bitmap, inode bitmap e blocchi utilizzati per l'indirect addressing.

Alcuni filesystem (XFS, JFS, ReiserFS) si limitano a loggare le operazioni che coinvolgono i metadata. Infatti il log dei metadata e' sufficiente per riportare il filesystem in uno stato consistente. Pero' visto che le operazioni sui data block non vengono loggate, nulla vieta che un crash possa corrompere il contenuto dei file.

Il filesystem ext3 puo' essere configurato per loggare le operazioni che coinvolgono sia i dati che i metadata, naturalmente maggiore e' il livello di log, maggiori le operazioni di scrittura e minori le prestazioni del filesystem.

Il filesystem ext3 ha tre modalita' di logging:

- **journal**: Tutte le modifiche a data e metadata sono salvate nel journal. E' il metodo piu' sicuro, ma piu' lento.

- **ordered**: Solo le modifiche ai metadato sono salvate nel journal. In ogni caso ext3 raggruppa metadata e relativi data block in modo che i data blocks siano scritti prima dei metadata. In questo modo la possibilita' di avere corruzioni di dati nel file e' ridotta. Questa e' la modalita' di *default* in ext3.

- **writeback**: Solo le modifiche ai metadata sono salvate nel journal. E' la modalita' di default di altri filesystem e naturalmente la piu' veloce.

La modalita' di journaling puo' essere specificata al momento del mount (*mount -t ext3 -o date=<option> ...*).

1.5.3 Il Journal Block Device Layer

In genere il journal viene creato in un file nascosto *.journal* nella root directory del filesystem.

Ext3 non amministra il journal in proprio, ma utilizza un layer generale del kernel detto Journal Block Device (JBD). Naturalmente JBD usa in genere lo stesso disco per loggare le modifiche apportate al filesystem ext3 e quindi e' vulnerabile agli stessi crash di ext3. Ovvero JDB deve anche proteggere se stesso dai crash che possano corrompere il journal.

L'interazione fra ext3 e JBD si basa su tre elementi:

- Log record: Descrive una singola modifica di un disk block nel journaling filesystem.

- Atomic operation handle: In genere ogni system call che modifica il filesystem mette in moto l'Atomic operation handle.

- Transactions: Include varie atomic operations handle i cui record sono marcati come validi da e2fsck allo stesso tempo.

1.5.4 L'HTree

Il filesystem ext3 eredita da ext2 il meccanismo di gestione delle directory basato sulle linked list. Con un sistema del genere non e' possibile utilizzare gli algoritmi di bilanciamento e ottimizzazione della ricerca derivanti dal mondo database. Inoltre un modello del genere espone al rischio di grosse perdite di dati in caso di corruzione di nodi vicini alla radice.

Per valicare questi problemi e' stato reso disponibile a partire dal kernel 2.4 un sistema di organizzazione ad albero bilanciato detto *HTree*. In questo sistema ogni blocco viene identificato da una hash key (*di default tea*) a 32bit e ogni chiave fa rifermento ad oggetti immagazzinati in una foglia dell'albero. I due livelli di nodi di cui e' costituito l'albero sono sufficienti per memorizzare oltre 16 milioni di file.

Questo sistema e' stato studiato per essere compatibile con il precedente basato sulle liste a cui si sovrappone. L'HTree viene infatti utilizzato solo per le operazioni di ordinamento e ricerca, mentre le operazioni sui file seguono il classico approccio di inode e data block.

1.6 ReiserFS

ReiserFS e' stato un filesystem scritto da zero da un'idea di Hans Reiser della Namesys. La sua introduzione nel kernel stabile risale al kernel 2.6.1, ma esso e' diventato effettivamente utilizzabile solo dalla verisione 2.6.6. Vale la pena di notare che fra gli sponsor del progetto possiamo annoverare Novell (attraverso Suse) e Darpa.

Naturalmente quello di cui stiamo parlando e' ReiserFS versione 3. Al momento la versione piu' recente disponibile e' la 4, che per ora Linus Torvalds non ha ancora voluto includere nel kernel ufficiale, inizialmente per motivi riguardanti il non totale adattamento al VFS. Esso e' pero' disponibile nel tree -mm mantenuto da Andrew Morton.

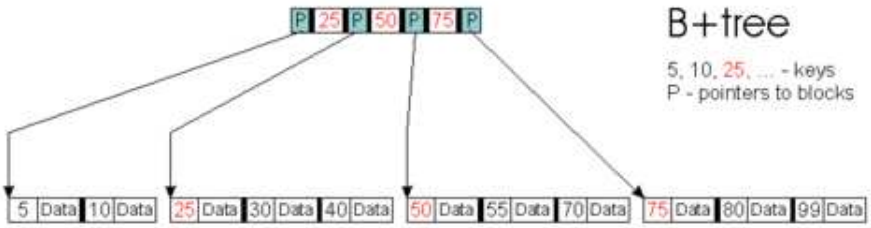
Per molti aspetti ReiserFS3 e 4 condividono il disegno.

La struttura base di ReiserFS versione 3 e' un albero bilanciato derivato da B+Tree utilizzato da XFS (e in parte da NTFS) e chiamato *B*Tree*. ReiserFS4 utilizza invece un albero B+Tree quasi originale detto *dancing tree*.

La versione 3 del filesystem e' ancora un block filesystem con blocchi di dimensione fissa 4096 byte. La versione 4 mantiene i blocchi da 4kB, ma introduce l'allocazione per extent, ovvero unita' di blocchi contigui referenziabili con un unico puntatore.

Il B*Tree Nella scienza dei calcolatori gli alberi bilanciati sono strutture dati utilizzate per database e filesystem, perche' permettono di mantenere i dati ordinati e accelerano le operazioni di inserimento e ricerca. Possiamo definire un albero come:

- Un albero e' un gruppo di nodi organizzati in un root node e zero o piu' gruppi di nodi addizionali detti sotto-alberi (subtree)
- Ogni subtree e' un albero
- Nessun nodo dell'albero punta al root node e esattamente un puntatore da un nodo dell'albero punta a ogni non-root node nell'albero
- Il root node ha un puntatore ad ogni subtree, cioe' un puntatore al root node del subtree



L'idea dietro ad un albero bilanciato (o B-Tree) e' che i nodi interni possano avere un numero variabile di child nodes in un intervallo predefinito. Quando un dato viene inserito o rimosso dalla struttura dati il numero di child nodes varia. Per mantenere il range predefinito, i nodi possono essere divisi o uniti. L'albero stesso viene bilanciato con il solo requisito che tutti i leaf node stiano

allo stesso livello. Ogni elemento di un internal node agisce come valore di separazione per i suoi child node o per i suoi subtree, gli internal node sono di solito rappresentati da un set ordinato di elementi e puntatori ai child node. Ogni nodo interno (tranne il root node) potra' contenere un massimo di U elementi e un minimo di L elementi, in realta' il numero il numero di puntatori ai child node meno uno. Quindi i limiti sono U-1 e L-1, con U che puo' essere 2L o 2L-1. Ovvero ogni internal node e' almeno per meta' pieno. Questo vuol dire che due nodi per meta' pieni possono essere uniti, mentre un nodo pieno puo' essere diviso in due. In questo modo si mantengono tutte le proprieta' dell'albero.

I leaf node hanno tutte le proprieta' dei nodi, ma non hanno child pointers.

Il root node ha un vincolo sul massimo numero di child, ma nessun limite inferiore.

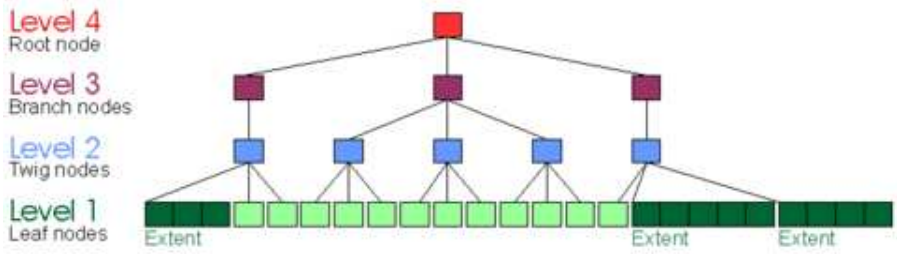
Un albero di profondita' n+1 potra' contenere U volte piu dati di uno di profondita' n.

Esistono alberi bilanciati in cui solo i leaf nodi contengono i dati, quindi questi nodi avranno una struttura diversa dagli altri. In generale pero' tutti nodi possono contenere valori.

Il B*Tree e' una variante del balanced tree che ha il vincolo che i non-root node debbano essere pieni almeno 2/3 invece di 1/2. Per poter ottenere questo invece di dividere un nodo non appena questo diventa pieno, le sue chiavi vengono condivise con il nodo adiacente, quando entrambi sono pieni, vengono suddivisi in tre nodi.

ReiserFS4 utilizza un'implementazione dell'albero detta *dancing tree* molto simile al B+Tree. In esso i data record sono memorizzati nei leaf node, ogni leaf node contiene i record di un particolare range di chiavi (che muta dinamicamente). Ogni leaf node viene referenziato dai sui parent, attraverso un puntatore al child node (in genere il block number nei filesystem) e la chiave di valore piu' piccolo nel range coperto dal leaf node. Ogni parent puo' avere un parent di livello superiore. Il numero massimo di puntatori in un record e' detto ordine del B+Tree.

Quello che si fa nel campo del filesystem e' far corrispondere un nodo ad un blocco su disco.



Paramentri di ReiserFS A differenza della maggior parte dei filesystem journaled, ReiserFS utilizza il B*Tree (o il dancing tree) per memorizzare sia i data che i metadata. Tutti le componenti del filesystem vivono all'interno dell'albero.

Le strutture usate per mantenere le informazioni relative ad un file sono chiamate nel ReiserFS *object*, mentre quelle relative ad un inode si chiamano

stat_ data items, quelle delle directory *directory items* e quelle realtive ai data file *direct* o *indirect items*.

I direct items sono semplicamente i data file, mentre gli indirect items sono puntatori a nodi che non hanno un formato predefinito. I direct items hanno un formato variabile e sono contenuti all'interno dei leaf node, direttamente nell'albero. Un file che sia abbastanza piccolo viene allora inserito direttamente nel balanced tree. Gli indirect items puntano a dati che non sono inseriti direttamente nell'albero. ReiserFS4 non ha questa struttura in quanto non ci sono indirect pointer, ma i BLOB vengono memorizzati negli extent.

Questo meccanismo minimizza la frammentazione interna in quanto i direct items possono essere composti da frazioni di piu' file. Per organizzare le componenti all'interno dell'albero viene utilizzata una tecnica di hashing, con tre algoritmi disponibili:

- rupasov
- tea
- r5 (default)

Una caratteristica peculiare di ReiserFS e' che esso non prefissa il numero di inodes, ma gli stat_data vengono creati durante il funzionamento stesso del filesystem. Quindi non esiste una limitazione sul numero dei file imposta alla creazione del filesystem stesso.

Il mount del ReiserFS Le opzioni del mount di un filesystem ReiserFS sono molteplici, ma le piu' importanti sono:

- nolog: disabilita le funzioni di journaling a favore della velocita'.
- notail: disabilita' la scrittura dei piccoli file direttamente nell'albero.
- replayonly: effettua il replay delle transazioni non completate all'atto di un crash, ma non monta il filesystem.
- resize=<block number> ridimensiona il filesystem al volo per raggiungere il numero di block group indicato.

1.7 I Network Filesystem

I network filesystem pemettono ad un calcolatore di accedere ai dati presenti su un'altra macchina remota connessa via rete e interagirvi come se fossero locali. Sara' quindi possibile montare il filesystem remoto e avere tutti i meccanismi di sicurezza basati su UID e GID tradizionali di Unix.

I principali filesystem di rete sono NFS, CIFS, Coda, NCP. Oltre a questo degno di menzione e' AFS, un filesystem progettato per operare su WAN.

1.7.1 NFS

Il Network filesystem sviluppato da Sun fornisce agli utenti un accesso trasparente ai filesystem remoti. Dal punto di vista dell'utente una risorsa accessibile via NFS viene trattata esattamente come una risorsa locale. Un filesystem apparira' quindi come parte del filesystem locale.

NFS permette ai processi utente un accesso trasparente in lettura e scrittura ai filesystem montati remotamente. Trasparenza significa che il processo lavorera' con i file presenti sul filesystem montato via NFS senza richiedere modifiche al suo codice. Questo perche' NFS e' progettato in maniera da presentare le risorse remote come estensioni di quelle locali.

NFS segue il modello client-server. Un server NFS e' un sistema in cui un filesystem e' configurato in maniera da poter condividere alcune parti con altri sistemi. Un NFS client e' una macchina che accede alle risorse condivise. Tradizionalmente NFS ha sempre lavorato con UDP, la le versioni piu' recenti possono operare anche con TCP.

In NFS il filesystem condiviso viene detto *exported filesystem*. Il client monta l'exported filesystem, facendono apparire come fosse locale.

La gestione di NFS dipende dal portmapper daemon, che abbiamo visto in occasione del NIS. Esso dovra' essere attivo come primo requisito, poi una serie di altri demoni provvederanno alla gestione di NFS e degli exported filesystem. Essi sono cinque:

- *nfsd*: e' il demone responsabile di gestire le richieste dei client.
- *biod*: e' il block I/O daemon, esso agisce sul client e gestisce i processi di lettura e scittura verso il server NFS per conto dei processi utente. (questo demone non esiste nell'implementazione presente in Linux)
- *mountd*: questo demone e' il reponsabile della gestione delle richieste di mount fatte dai client.
- *lockd*: e' un demone attivo sul server e sui client e gestisce i file lock.
- *statd*: e' un demone attivo sul server e sui client e mantiene lo stato dei lock attualemente attivi.
- *quotad*: gestisce le quota sui filesystem esportati.

1.7.2 Configurazione di server e client

La configurazione del server avviene piuttosto semplicemente per mezzo del file /etc/exports (5). Preparato questo file occorre attivare i demoni (secondo l'ordine seguente: rpc.mountd, rpc.nfsd, rpc.statd, rpc.lockd ed eventualmente rpc.rquotad), oppure se essi fossero gia' attivi notificare le modifiche con il comando:

exportfs -a

Un client abilitato all'accesso potra' allora montare uno dei filesystem per cui ha i privilegi con un comune comando mount con filesystem type impostato a nfs. Naturalmente esistono una serie di opzioni per il filesystem, alcune delle quali possono essere analizzate:

- *soft*: se la richiesta di un file fallisce il client riporta l'errore al processo che richiedeva l'accesso sulla macchina client. La maggior parte dei processi non e' in grado di gestire quest'errore.
- *hard*: il programma che accede ad un file su un NFS mounted filesystem si blocca se il server va in crash. Il processo non puo' essere interrotto o killato. Quando l'NFS server torna disponibile il programma riprendera' da dove era terminato.

- *intr*: permette l'interruzione del processo se un NFS server non risponde.
- *rsize, wsize*: specificano il blocco dati che client e server si scambiano, questo influenza molto le prestazioni.

Un problema comune con NFS e' quello di montare un filesystem e poi non essere in grado di accedere ai file, perche' non si hanno i permessi. Questo e' dovuto al fatto che NFS e' stato progettato per lavorare con NIS e quindi con un meccanismo di autenticazione (e quindi di assegnazione di UID e GID) centralizzato.

Quello che accade invece e' che gli UID dell'utente nel filesystem esportato e in quello locale non coincidano, rendendo impossibile l'accesso.

Per la root il problema puo' essere diverso, vediamo il paramentro di export (lato server):

no_root_squash: per default ogni richiesta di file fatta da root viene trattata come se fosse fatta dall'utente nobody sul server. Se questa opzione e' selezionata l'utente root della macchina locale avra' lo stesso livello di accesso dell'utente root del server. Questa rappresenta una notevole falla di sicurezza.

1.7.3 AFS (cenno)

Andrew File System fa parte del progetto Andrew di Carnergie Mellon Univerisity. E il suo uso principale e' nel distributed computing.

La sua struttura e' divisa in *celle* che possono contenere un numero molto elevato di client. La cella e' un insieme di macchine client e server che in genere appartengono a qualche dominio organizzativo. Ogni cella e' mantenuta indipendente dalle altre dagli amministratori. Le subdirectory di primo livello di AFS costituiscono i genere i punti di accesso dei filespace delle celle.

AFS utilizza kerberos per i meccanismi di autenticazione e implementa le POSIX ACL.

Ogni client fa una cache dei file per avere migliori prestazioni e immunita' alla caduta delle connessioni. Le modifiche ai file sono fatte sulle copie locali, quando un file viene chiuso le porzioni cambiate sono copiate sul fileserver. la consistenza delle cache e' mantenuta con un meccanismo detto *callback*.

Il disegno di AFS implica il locking di interi file.

Una caratteristica di AFS e' il *volume*: un albero di file, sub-directory e AFS mountpoints (link ad altri AFS volume). I volumi sono creati dagli amministratori e linkati in specifici path nella cella AFS. A questo punto i client possono interagire col volume senza preoccuparsi della locazione fisica dello stesso.

I volumi AFS possono essere replicati cosi' da poter sostituire i volumi originali in caso di indisponibilita' o manutenzione.

Il name space di una workstation AFS e' suddiviso in local e shared. Lo shared name space in genere viene montato sotto /afs ed e' lo stesso per tutte le workstation.

2 La Linux Filesystem Hierarchy

Un filesystem e' costituito dai metodi e dalle strutture dati che un sistema operativo usa per tenere traccia dei file su un disco o su una partizione.

Linux, come tutti gli Unix, ha una sola struttura gerarchica di directory. Tutto ha origine nella root directory, rappresentata da /, e si espande in un albero di subdirectory. Al contrario i sistemi Microsoft Windows hanno il concetto di "drives". Questo tipo di disegno e' noto come united filesystem.

Durante i primi anni '90, ogni distribuzione aveva il proprio schema per porre i file nella gerarchia di directory. Questo, naturalmente, comportava un sacco di problemi. Il documento *Linux File System Structure* e' stato creato per dare delle linee di indirizzo che prevenissero questa anarchia. Il gruppo che ha creato il documento viene spesso identificato con l'acronimo FSSTND (FileSystem STaNarD). Questo documento e' servito per standardizzare il layout del filesystem di Linux su tutte le distro, sebbene non tutte lo abbiano adottato completamente.

Il FSSTND non ha cercato di emulare nessuno Unix commerciale, esso si limita a dettare le regole su dove posizionare un file qualora esso sia presente.

2.1 Mount e Unmount dei filesystem

2.1.1 Mount e umount

I filesystem in Unix sono tradizionalmente montati smontati per mezzo dei comandi *mount* e *umount*. Naturalmente sara' possibile montare filesystem i cui formati sono registrati nel VFS, ovvero che il kernel e' in grado di riconoscere. E' sempre possibile linkare a run time il codice oggetto di un modulo per aggiungere un filesystem a quelli riconosciuti dal kernel.

Per vedere quali filesystem sono riconosciuti in un dato momento si puo' accedere alle informazioni esportate dal kernel per mezzo del filesystem virtuale /proc e in particolare nel file:

/proc/filesystems

Se il filesystem e' riconosciuto e' possibile procedere al mount del filesystem (in genere eseguito con privilegi di superutente, salvo casi particolari) con il comando:

mount [-fnrsvw] [-t vfstype] [-o options] device dir

dove *vfstype* e' il tipo di filesystem, come si evince da /proc/filesystems, *device* e' il block device su cui risiede il filesystem e *dir* e il mount point. Per le altre opzioni vale la pena di guardare mount(8).

Per smontare un device, e quindi renderlo non piu' accessibile per mezzo di un mountpoint, si utlizza il comando:

umount [-nrv] device | dir [...]

Un paio di opzioni possono essere degne di nota, soprattutto per smontare volumi NFS che si siano bloccati. utilizzando il comando:

umount -flt nfs <device | dir>

si va a forzare l'unmount e ottenere di nuovo il controllo del sistema. Per le altre opzioni si puo' vedere umount(8).

I filesystem da montare usualmente sono i genere elencati nel file /etc/fstab (/etc/filesystems in altre varianti Unix). Una tabella utilizzata soprattutto nella fase di boot, quando uno degli script di inizializzazione impartisce il comando mount -a, montando tutti i filesystem per cui non sia stata specificata l'opzione noauto.

Per i filesystem specificati in /etc/fstab e' possibile il mount semplicemente specificando il device oppure il mount point.

In ogni istante e' possibile accedere alle informazioni relative ai filesystem montati dal file virtuale /proc/mounts, oppure dal file /etc/mtab, che fornisce ulteriori informazioni. La differenza e' che il primo fornisce informazioni provenienti dal kernel, mentre il secondo viene scritto dai comandi mount e umount, che possono anche essere configurati per non aggiornarlo. Naturalmente mtab contiene ulteriori e utili informazioni provenienti dallo userspace, come l'utente che ha eseguito il comando di mounting.

2.2 L'albero delle directory Unix

Come abbiamo piu' volte ripetuto in Unix tutto e' un file. Chiaramente l'organizzazione dell'albero delle directory gioca un ruolo molto importante nel sistema e ha una configurazione tradizionalmente accettata da tutte le varianti.

Per avere un idea dell'organizzazione del filesystem Unix e' possibile impartire il comando:

tree -L 1

e vedere il primo livello dell'albero delle directory.

Chiaramente ogni directory puo' essere una semplice directory dell'albero originatosi in / oppure il mount point e quindi la root di un filesystem a se stante.

Le directory notevoli sono le seguenti:

2.2.1 La directory /

Per aderire con lo standard FSSTND le seguenti directory o link simbolici sono necessari in /.[2]

```
/bin       Essential command binaries
/boot      Static files of the boot loader
/dev       Device files
/etc       Host-specific system configuration
/lib       Essential shared libraries and kernel modules
/media     Mount point for removeable media
/mnt       Mount point for mounting a filesystem temporarily
/opt       Add-on application software packages
/sbin      Essential system binaries
/srv       Data for services provided by this system
/tmp       Temporary files
/usr       Secondary hierarchy
/var       Variable data
```

```
/ -- the root directory
/home User home directories (optional)
/lib<qual> Alternate format essential shared libraries
                (optional)
/root Home directory for the root user (optional)
```

[2] Da non confondere con la root del sistema e' la directory /root che costituisce la home directory del superuser. Essa non viene posta sotto la directory /home come per tutti gli altri utenti. Questo perche' generalmente su /home viene montato un altro filesystem che non sarebbe disponibile immadiatamente al boot in sistuazioni di emergenza oppure quando si opera in modalita' single user.

Com'e' noto tutti i file system iniziano con /, la root directory. Tutte le altre directory sono figlie della root directory. La partizione su cui e' situato il root filesystem viene montata durante il boot e il sistema non puo' fare il boot de non e' in grado di trovarla.

2.2.2 La directory /bin

La directory /bin contiene, diversamente dalla /sbin, molti utili comandi che possono essere usati sia dagli utenti comuni (non-privilegiati) che dall'amministratore. Gli eseguibili presenti sono considerati (al contrario di quelli presenti in /usr/bin) essenziali, nel senso che essi devono essere disponibili anche nel caso in cui solo la partizione che contiene / sia montata.

2.2.3 La directory /boot

Questa directory contiene tutto cio' che e' necessario per il processo di boot, eccetto che che i file di configurazione che non servono a boot time e il map installer. La directory /boot contiene dati che sono utilizzati prima che il kernel inizi l'esecuzione dei programmi user-mode.

2.2.4 La directory /dev

La directory /dev e' la locazione la locazione degli special file. La maggior parte dei device sono "block devices", che fanno riferimento a dispositivi che immagazzinano dati, oppure "character device" che fanno riferimento a dispositivi che trasmettono o trasferiscono dati. Alcuni esempi sono:

- /dev/ttyS0 - First serial port
- /dev/psaux - PS/2 connection
- /dev/lp0 - First parallel port
- /dev/dsp - The name DSP comes from the term digital signal processor, a specialized processor chip optimized for digital signal analysis. Sound cards may use a dedicated DSP chip, or may implement the functions with a number of discrete devices. Other terms that may be used for this device are digitized voice and PCM.
- /dev/usb (USB Devices) - This subdirectory contains most of the USB device nodes. Device name allocations are fairly simplistic so no elaboration is be necessary.
- /dev/sda First SCSI device
- /dev/scd First SCSI CD-ROM device.

I dispositivi sono definiti per tipo, come block o character, e dai *major* e *minor* number. Il major number viene usato per selezionare la categoria del device e il minor number viene usato per identificare uno specifico dispositivo.

I device vengono normalmente creati con il comando mknod; esiste uno script di nome MAKEDEV che crea tutti i device nel sistema.

Attulmente (kernel 2.6) esiste un sistema in userspace (udev) che crea i device dinamicamente e che ha sostituito la precendente implementazione in kernel space nota come devfs.

2.2.5 La directory /etc

Questa directory e' il centro nevralgico del sistema. Essa contiene tutti i file di configurazione di sistema nel suo albero di sottodirectory. Un "configuration file" e' definito come un file locale utilizzato per controllare le operazioni di un programma, deve essere statico e non deve essere un eseguibile binario. Normalmente in questa directory non dovrebbero esserci binari.

Nell'albero delle sottodirectory vale la pena di citare:

- /etc/X11 - Directory che contiene i file di configurazione per X Windows System. Molto spesso questi file sono link simbolici alla directory /usr/X11R6.
- /etc/cron.d, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly - Directory che contengono gli script eseguiti da cron
- /etc/crontab - File di configurazione di cron
- /etc/exports - File di configurazione delle directory esportate via NFS
- /etc/fstab - File System Table. E' il file di configurazione di mount. I filesystem elemcati sono montati automaticamente al boot.
- /etc/group - Group file
- /etc/hostname - Contiene l'hostname della macchina
- /etc/host.conf - Contiene l'ordine di ricerca per il resolver dei nomi di host
- /etc/hosts.allow, /etc/hosts.deny - Configurazione dei tcpwrappers
- /etc/xinetd.conf - Configurazione del superserver dei servizi TCP/TP
- /etc/xinetd.d/ - Directory con la configurazione dei servizi che girano sotto il controllo di xinetd
- /etc.init.d - Script eseguiti all'inizializzazione del sistema e al cambio di runlevel
- /etc/inittab - File di configurazione del programma init
- /etc/issue - Output del comando getty prima del login prompt
- /etc/kde/ - Script di inizializzazione per kde e per la configurazione di KDM
- /etc/ls.so.conf, /etc/ld.so.cache - Contengono rispettivamente il search path utilizzato dal linker e la cache delle librerie trovate nelle directory configurate in /etc/ls.so.conf.
- /etc/login.defs - File di controllo per il login package
- /etc/magic - configurazione e magic data per il comando file(1)
- /etc/modules.conf (o /etc/modprobe.conf) - Configurazione del comando modprobe

- /etc/modules - Moduli che devono essere caricati all'avvio del sistema
- /etc/motd - Message Of the Day
- /etc/mtab - Lista dei filesystem attualmente montati
- /etc/nsswitch.conf - File di configurazione del System Database/Name Service Switch
- /etc/pam.d/ - Directory base della configurazione dei moduli PAM
- /etc/passwd - File delle password
- /etc/profile - File e comandi che vengono eseguiti da molte shell al login
- /etc/profile.d/ - Shell scripts eseguiti al login
- /etc/protocols - Elenco del protocolli di rete
- /etc/rcN.d/ - Dove N e' un numero. Script eseguiti in quel runlevel. Molto spesso sono link a /etc/init.d
- /etc/resolv.conf - Elenco dei nameserver e dei suffissi utilizzati dal revolver
- /etc/samba/ - Directory di configurazione di Samba
- /etc/securetty - Lista dei terminali su cui root puo' fare login
- /etc/sudoers - Configurazione del comando sudo
- /etc/shadow - Shadow password file
- /etc/sysctl.conf - File di configurazione che permette di alterare dei parametri del kernel
- /etc/services - File dove sono definite le IANA ports e i realtivi nomi dei servizi
- /etc/shells - Lista delle shell che possono essere utilizzate nel sistema
- /etc/skel/ - Skeleton files che devono essere utilizzati alla creazione degli utenti
- /etc/sysconfig/ - La directory contiene file di configurazione e subdirectory per il setup delle specifiche della configurazione di sistema e per il processo di boot.
- /etc/ssh/ - FIle di configurazione di ssh
- /etc/syslog.conf - Specifica la locazione dei file di log
- /etc/timezone - Specifica la timezone del sistema

2.2.6 La directory /home

Linux e' un ambiente multiutente e ad ogni user viene assegnata una specifica directory accessibile solo da lui/lei e dall'amministratore di sistema. Le user home directory si trovano in genere in /home/$USER. La home directory dell'utente contiene anche i file di configurazione personali che generalemente iniziano con il carattere ".".

2.2.7 La directory /lib

La directory /lib contiene i moduli del kernel necessari al boot e le immagini delle shared library necessarie a far girare i programmi nel root filesystem.

2.2.8 La directory /lost+found

Questa directory ha la sua importanza dopo il crash di un sistema. Al seguente boot un check di tutti i filesystem verra' eseguito. L'utilita' fsck cerchera' di recuperare qualunque file corrotto. Il risultato del suo lavoro lo troveremo in questa directory.

2.2.9 La directory /media

Il suo scopo e' quello di fornire i mount point per tutti i device rimovibili.

2.2.10 La directory /mnt

Questa directory e' un generico mount point sotto cui montare i propri filesystem e i device. In genere non ci sono limitazioni al creare mount point ovunque nel sistema, ma /mnt e' dedicata a questo scopo.

2.2.11 La directory /opt

Questa directory e' riservata per tutto il software e i pacchetti aggiuntivi che non siano parte dell'installazione di default.

2.2.12 La directory /proc

/proc ha la particolarita' di essere un filesystem virtuale. Viene spesso riferito come process information pseudo filesystem. Esso non contiene veri file, ma informazioni di sistema durante il runtime. In questo senso puo' essere considerato un centro di informazione e controllo per il kernel.

Il proc filesystem permette all'utente di interagire con le strutture dati del kernel e modificare a runtime una serie di comportamenti semplicemente scrivendo gli opportuni valori in una serie di file virtuali, presenti nella subdirectory /proc/sys. Questo e' possibile direttamente o per mezzo si una utility *sysctl* a cui puo' essere passato il paramentro da cambiare, oppure puo' essere opportunamente modificato il file /etc/sysctl.conf. La lista dei parametri disponibili puo' essere ottenuta con l'opzione -a, mentre con l'opzione -p si fa esaminare di nuovo il contenuto del file di configurazione.

2.2.13 La directory /root

Questa non e' altro che la home directory del superutente.

2.2.14 La directory /sbin

Linux discrimina i comuni eseguibili da quelli che vengono utilizzati per la manutenzione del sistema e i task amministrativi. Questi ultimi risiedono in questa directory, oppure quando meno importanti in /usr/sbin. I programmi di amministrazione installati localmente dovrebbero essere messi in /usr/local/sbin.

2.2.15 La directory /usr

La directory /usr contiene la maggior parte dei dati su un sistema. Il suo nome significa User System Resources, ma viene anche identificata come "user" visto che storicamente conteneva le home directory degli utenti.

Al suo interno troviamo fra le altre:

- /usr/X11R6 - La struttura delle directory dell'X Windows System
- /usr/bin - Questa directory contiene la maggior parte degli eseguibile nel sistema
- /usr/include - Header files necessari per la compilazione del codice sorgente
- /usr/lib - Directory contenente librerie
- /usr/local - L'idea di fondo e' che questa fosse una usr directory locale, mentre la vera /usr doveva essere montata read-only remotamente sulla rete. In realta' viene utilizzata dagli amminstratori quando installano localmente del software a partire dai sorgenti.
- /usr/man - Repository delle man pages
- /usr/sbin - Contiene programmi adibiti all'amminstrazione di sistema. In genere non fa parte del $PATH degli utenti
- /usr/share - La directory contiene file sharable architecture-indipendend
- /usr/src - Contiene i sorgenti del kernel

2.2.16 La directory /var

Questa directory contiene dati variabili, come file di log e spool di stampa, posta e del cron. Per queste caratteristiche potrebbe essere posta sotto /usr, ma questo impedirebbe di poter montare /usr read-only.

- /var/lock - Contirne lock file di programmi
- /var/log - Directory deputata a contere i log di sistema e degli applicativi, soprattutto i demoni
- /var/run - Contiene i PID file del processi che ne creano

2.2.17 La directory /tmp

Questa directory contiene per la maggior parte file che sono richiesti temporaneamente. In molti sistemi questa directory viene ripulita a boot time.

2.3 I device file

I device file si dividono in *character* oriented device file e *block* oriented device file.

Character device Si trovano tipicamente nella /dev directory. Essi forniscono l'interfaccia del VFS per comunicare con i device drivers attraverso il filesystem un carattere alla volta. I permessi di questi file iniziano con la lettera *c* che li identifica, ogni device file contiene un *major* e un *minor* number che permettono di identificare il device driver relativo al dispositivo.

Block device I block device file si trovano tipicamente della direcrory /dev. Essi sono del tutto equivalenti ai character device file, ma essi sono identificati da una lettera *b* nella stringa dei permessi e sono utilizzati per trasferire blocchi di dati invece che caratteri.

Abbiamo visto che i device file sono l'interfaccia di accesso dello user space ai device driver. Questi driver possono essere implementati direttamente nell'immagine del kernel del sistema operativo che viene caricata al boot, oppure possono essere, e questo accade molto piu' frequentemente, disponibili come moduli separati che sara' possibile linkare e unlinkare dal kernel stesso a runtime.

La gestione dei moduli viene fatta per mezzo di una serie di utilita' in userspace, che eseguono una serie di controlli e vanno poi ad invocare le opportune system call per il caricamento degli stessi:*insmod, modprobe, rmmod, lsmod.*

Il caricamento manuale dei moduli puo' essere scomodo, specialmente se una macchina ha molti utenti connessi senza privilegi di amministrazione. Per ovviare a questo problema e' possibile configurare il kernel per il caricamento automatico dei moduli.

Ricordiamo che il comando modprobe e' in grado di caricare un modulo e tutti quelli da cui dipende, inoltre e' in grado di passare a ogni modulo i corretti parametri, in base a quanto specificato nella sua configurazione: in genere in /etc/modules.conf oppure /etc/modprobe.conf. Questo agisce in userspace.

Nel kernel space e' attivo un elemento dal nome kmod che permette il caricamento automatico dei moduli quando essi siano necessari al funzionamento del sistema.

Vediamo il funzionamento di modprobe. Un device file puo' essere creato nella directory /dev con il comando:

mknod -m <permissions> /dev/<name> type major minor

La coppia di valori major e minor number contribuiscono ad identificare un dispositivo all'interno dello stesso tipo. Il major nuber identifica la classe e il minor il particolare dispositivo all'interno della classe o una diversa modalita' di funzionamento del dispositivo.

Un modulo che identifica una classe di dispositivi viene univocamente identificato dal suo major-number, ovvero ad ogni major number corrisponde un modulo che ne implementa il driver. Il modulo avra' un diverso comportamento in base al minor number.

Cosa accade dunque quando il kernel deve caricare un modulo. Esso conosce il major number associato ad esso, perche' legato al device da gestire. Kmod quindi invoca modprobe con un certo *char.major-<number>.* Il modulo dovra' essere presente nel file delle dipendenze (modules.dep) creato al boot, o a richiesta, da comando depmod.

I moduli dipendono poi uno dall'altro e kmod dovra invocare modprobe per caricare tutti i moduli su cui si hanno dipendenze. Il kernel conosce queste

dipendenze perche' sono espresse esplicitamente all'interno del modulo dalla funzione *request_module()*.

3 Il filesystem avanzato

3.1 Estensioni dei filesystem

Abbiamo visto l'implementazione classica del filesystem ext2. In particolare la struttura dell'inode ha tradizionalmente avuto un numero di campi piuttosto limitato. Con la crescita di Linux a livello enterprise si sono pero' sentite le necessita' di aggiungere ai filesystem (e quindi al VFS) alcune caratteristiche che erano presenti da tempo negli Unix commerciali. Queste erano focalizzate a rendere il meccanismo di controllo sull'accesso ai file piu' granulare e a limitare il potere del superuser.

Per realizzare queste *extended capabilities* sono stati aggiunti dei campi (*extended attributes o EA)* alla struttura dell'inode, che nelle ultime implementazioni appare decisamente piu' complessa.

In particolare esiste un campo dell' Inode *i_file_acl*, se il suo valore e' diverso da zero, allora contiene il numero del blocco in cui sono contenuti gli extended attributes relativi a quell'inode. Piu' inode con gli stessi extended attributes possono puntare allo stesso EA block.

3.1.1 ACL

Le ACL o Access Control List, sono un meccanismo per rendere piu' granulare e flessibile il classico meccanismo di acceso ai file basate sulla tripletta dei permessi per owner user, owner group, other.

Il modello convenzionale di permessi POSIX usa tre *classi* di utenti a cui assegnare i permessi: owner, owner group e other users. E tre permission bit per ogni classe per assegnare i permessi di read (r), write(w) e execute(x).

Le ACL possono essere divise in due classi. La *minimum* ACL che semplicemente comprende le entry per i tipi owner, owner group e other users che corrispondono ai permission bit tradizionali. Un'*extended* ACL oltrepassa questo concetto. Essa conterra' una *mask* e varie entry per *named users* e *named groups*.

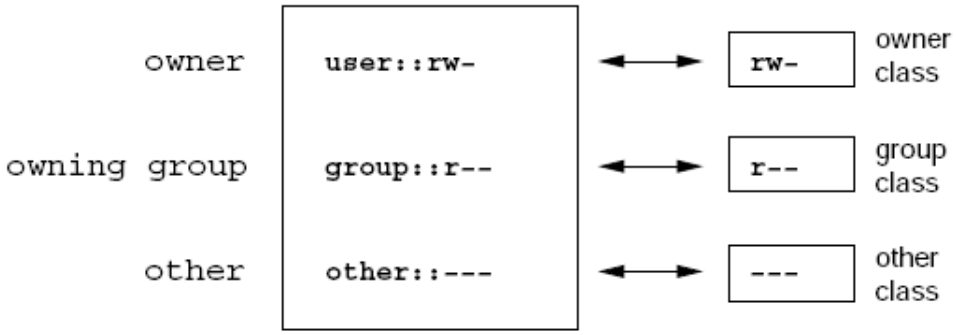
Type	**Text Form**
owner	user::rwx
named user	user:name:rwx
owning group	group::rwx
named group	group:name:rwx
mask	mask::rwx
other	other::rwx

I permessi definiti per *owner* e *other* sono sempre attivi, gli altri (*named user, named group* e *owning group*) possono essere macherati dalla *mask*. Per esempio se un named user ha i permessi rw- su un file e la maschera e' r-x, allora i permessi effettivi saranno r–.

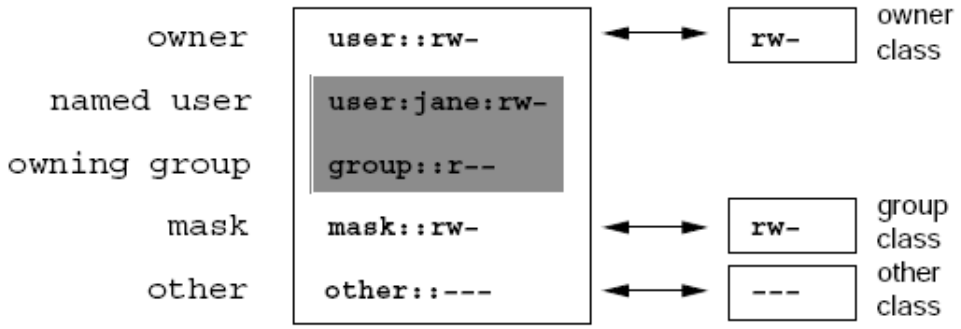
Si puo' dire che saranno attivi i permessi presenti sia nella entry sia nella mask.

I permessi dell'owner sono sempre mappati nella parte owner della ACL

- Nel caso di minimum ACL (senza mask) la group class viene mappata nella parte owning group della ACL.
- Nel caso di extended ACL (con mask) i permessi della group class sono mappati in quelli della mask entry.



Minimum ACL: ACL Entries Compared to Permission Bits



Extended ACL: ACL Entries Compared to Permission Bits

I comandi userspace per interagire con le acl sono: *setfacl*, *getfacl* e *chacl*. Quest'ultimo viene fornito solo per compatibilita' con altri tipi di Unix ed e' un frontend di setfacl.

L'utility getfacl serve a interrogare la ACL relativa agli oggetti passati come argomenti.

Piu' interessante e' setfacl che permette di modicicare (o creare se se necessario) le ACL. Le opzioni -m e -x permettono, rispettivamente, di modificare e eliminare elementi delle ACL

Per le directory e' possibile definire una default ACL. Essa stabilisce i permessi di accesso che tutti gli oggetti sotto questa directory erediteranno quando vengono creati. Naturalmente essa influisce anche su tutte le subdirectory. Una default ACL non ha un effetto diretto sui permessi di accesso, ma entra in gioco solo quando gli oggeti vengono creati.

Le entry delle ACL sono analizzate nel seguente ordine: *owner, named user, owning group,named group, other*. L'accesso e' gestito in base alla entry che meglio si adatta al processo. Se un processo appartiene a piu' gruppi, la cosa si complica e un gruppo casuale fra quelli con i permessi richiesti viene scelto.

La conservazione delle ACL da parte dei programmi che manipolano i file potrebbe non essere garantita. in particolare un programma che modifica direttamente un file mantiene le ACL. Un file che crea un file copia lo modifica e poi lo rinomina potrebbe perdere le ACL.

Per poter avere le ACL attive su un filesystem, e' necessario montarlo con l'opzione *acl*.

3.1.2 Le capabilities

Le capabilities sono extended attributes introdotti nei filesystem nell'ottica della sicurezza. I privilegi degli utenti hanno tradizionalmente due varianti in Linux: user e root. Gli utenti comuni hanno poteri molto limitati e possono interagire solo con file e processi di loro proprieta'. Root, al contrario ha un potere illimitato e puo' interagire con qualunque file, processo e con l'hardware senza restrizioni. Quello che si e' pensato e' stato di creare un livello intermedio, ovvero concedere ai programmi e all'utente root solo i privilegi che gli sono necessari. In questo modo se l'account di root fosse violato, il sistema non potrebbe essere danneggiato completamente.

Il livello intermedio si chiama POSIX capabilities. Queste dividono il sistema in gruppi logici a cui concedere o revocare le capabilities. Le capabilities permettono all'amministratore di stabilire granularmente casa un processo puo' fare o meno.

Le capabilities sono elencate nel file /usr/include/linux/capability.h. Ogni capability non e' altro che un bit di una bitmap a 32 bit. Al momento 28 di esse sono definite.

Il file /proc/sys/kernel/cap-bound contiene un valore a 32bit che identifica le capabilities attive. Se una capability viene rimossa dal sistema, sara' impossibile per qualunque processo, anche quelli della root, acquisirla di nuovo. In realta' ci sono due modi per ottenere di nuovo una capability:

- init puo' riaggiungere le capabilities (altrimenti potrebbe non essere in grado di cambiare runlevel), ma nessuna implementazione per ora e' in grado.
- Un processo con CAP_SYS_RAWIO puo' modificare la kernel memory per mezzo di /dev/mem e riacquisire la capability

Esiste un comando *lcap* che permette di rimuovere le capability. Quando si modificano le capabilities, queste influenzano solo i nuovi processi creati via exec(2).

Al momento non esiste nessun metodo integrato nelle utilita' di sistema (come chattr) per modificare le capabilities di un file. Esiste la possibilita'

di usare i servizi della libcap per mezzo di alcuni programmi (setpcap) che permettono di modificare le capabilities di alcuni processi.

L'uso delle capabilities e' sempre stato limitato, sia perche' costituiscono un sistema non troppo flessibile, sia perche' un loro uso poco accorto potrebbe rendere il sistema inaccessibile anche dal superutente.

Si preferisce oggi utilizzare al loro posto un'infrastruttura di Mandatory Access Control integrata nel kernel di nome NSA SELinux. Purtroppo l'estrema flessibilita' si paga in termini di difficolta' di configurazione.

3.2 Quota

L'uso delle disk quota puo' prevenire l'occupazione eccessiva dello spazio disco da parte degli utenti. Quota permette di definire il numero di inode che un gruppo o uno user puo possedere e il numero di disk block che possono essere allocati ad un utente o a un gruppo.

Le quota sono gestite in base all'utente e al filesystem, quindi per ogni filesystem in cui un utente possa creare un file occorre assegnare una quota.

Esistono una serie di tool per creare e amministrare le quota. Le quota possono essere attivate indipendentemente per utenti e gruppi con le opzioni *usrquota e grpquota* da passare al mount del filesystem.

I quota database vengono approntati dal comando *quotacheck* (specificando il formato quota v.2). Qualora questi file (per user e group) non esistano essi vengono creati nella root del filesystem, con i come aquota.user e aquota.group. Da questo punto quotacheck permette di controllare le quota su tutti i filesystem.

Per assegnare le quota si utilizza il comando *edquota* che utilizza vi (oppure cio' che e' specificato nella variabile di shell EDITOR) per modificare la quota, il numero di blocchi da assegnare e' sempre specificato in KB.

Vediamo il significato dei soft limit e degli hard limit:

- Soft limit: indica la massima quantita' di spazio che un quota user ha su una partizione. Se combinato con un grace period, esso agisce come una soglia superata la quale uno warning viene inviato all'utente.
- Hard limit: agisce quando il grace period e' attivo. Specifica il limite assoluto all'occupazione disco che il quota user non puo' comunque superare.
- Grace period: rappresenta un limite di tempo prima del quale il soft limit e' attivo per un filesystem con quota abilitato.

Le principali utility per la gestione della quota sono: quotacheck, edquota, repquota che produce un report dello stato attuale del filesystem, quotaon e quotaoff usati per abilitare e disabilitare il quota accounting a runtime.

References

[1] O'Reilly - Daniel P. Bovet, Marco Cesati "Understanding the Linux Kernel"

[2] Access Control List -http://www.suse.de/~agruen/acl/chapter/fs_acl-en.pdf

[3] Linux Filesystem Hierarchy - http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/index.html

[4] Linux Kernel 2.4 Internals -http://www.tldp.org/LDP/lki/index.html

[5] Linux device file list -http://www.lanana.org/docs/device-list/devices-2.6+.txt

Contents