Architettura a livelli di dischi, blocchi e filesystem

Un sistema Linux utilizza una struttura a più livelli che va dal livello fisico dei dispositivi fino alle chiamate di sistema tipo `open()` e `read()`.

(**7**) **Applicazioni** (usano open, read, write via libc e system call).

(**6**) **VFS - Virtual File System** (offrono API unificate, open, read, write, seek, close).

(**5**) **Filesystem** (ext4, xfs, btrfs, zfs, …) implementano le API di accesso specifiche e le eventuali gestioni di volumi ridondati, replicati e distribuiti.

(**4**) **Strutture del filesystem**: inode, blocchi FS.

(**3**) **Dispositivo a blocchi** (**LBA logici**). Interfaccia esposta sia da dispositivi hw locali, sia da dispositivi virtuali locali, sia da storage remoti per tramite di client locali. I dispositivi /dev/sda ne sono un esempio.

(**2**) **Driver del dispositivo**. Sono moduli del kernel che mappano le richieste di accesso a blocchi logici LBA e le traducono in comandi per la specifica tecnologia. Nel caso di storage a blocchi in rete inoltrano le richieste ai server remoti. Questi moduli si occupano anche di gestire la tabella delle partizioni nel disco.

(**1**) **Tecnologia fisica** (HDD: settori — SSD: pagine/erase blocks)

A che serve questa separazione?

- permette a Linux di usare decine di filesystem diversi
- permette di sostituire l'hardware senza cambiare software
- permette l'uso di dischi virtuali (LVM, iSCSI, loopfile, immagini QCOW2, RAID…)
- riduce la complessità isolando le responsabilità nei vari livelli

1) Dischi - Livello fisico: settori, pagine e blocchi fisici

HDD: settori fisici reali
Gli hard disk meccanici hanno una geometria fisica composta da tracce e cilindri che suddivide il disco in settori realmente incisi sulla superfice magnetica:

- Settori fisici da **512 byte** (vecchio standard)
- Oppure **4K** (standard moderno *Advanced Format*)

SSD: non esistono veri "settori fisici"
Gli SSD **simulano** i settori per il sistema operativo, ma internamente usano:

- **Pagine** (unità minima di lettura/scrittura → 4K–16K)
- **Blocchi di cancellazione (erase blocks)** (dimensione 256K–2M)

Sono necessarie alcune funzioni cruciali del livello fisico di SSD:

- Wear leveling - distribuisce le scritture per evitare che alcune celle si usurino prima.
- Garbage collection - sposta dati validi da un blocco a un altro prima di cancellare un blocco intero.

2) Device Driver – Driver dei dispositivi- traduce richieste di accesso a blocchi logici in richieste ai dispositivi fisici

Ogni dispositivo di storage (HDD, SSD, NVMe, USB, iSCSI) al suo interno funziona diversamente ma deve esporre **blocchi logici numerati** tipicamente formati da 512 bytes o, più spesso, **4096 bytes** (4KB): `LBA 0, LBA 1, LBA 2, ...`

Occorre un livello che traduca le richieste per blocchi logici in richieste per la tecnologia sottostante. Questo livello è fornito:

- **Dai driver per l'hardware** (tipicamente moduli del kernel: sata, ahci, nvme, dm).
- **Da un software di virtualizzazione locale**→ loop devices, volumi LVM, file immagine.
- **Da un target iSCSI** o Fibre Channel→ il server esporta LBA virtuali, cioè l'iSCSI espone blocchi **logici**, non fisici.

Quando arriva una richiesta di lettura o scrittura su un blocco logico, questa viene tradotta in una o più operazioni sull'hardware o trasportata via rete fino al server che la soddisfa.

- Il livello dei Device Driver distingue volumi logici (partizioni) all'interno del disco fisico.
- La formattazione è una operazione preliminare che organizza i blocchi logici dei dischi.
- La **formattazione del disco** crea le partizioni nel disco ed è realizzata dal device driver.

- La creazione della partizione (usare **sudo fdisk**)
  - stabilisce quale intervallo di blocchi logici verrà usato da una certa partizione.
  - indica quale tipo di filesystem ci sarà nella partizione.
  - Queste informazioni sono scritte nella **Tabella delle partizioni su disco**, collocata in specifici **settori fisici** del disco.

- La **tabella delle partizioni** si trova
  - nel **settore** 0 del disco, se viene usato il tradizionale MBR Master Boot Record;
  - oppure nei **settori fisici a partire dal 2**, se del tipo moderno GPT (GUID PartitionTable) compatibile con UEFI, il BIOS moderno.

3) Dispositivi a Blocchi - LBA (Logical Block Addressing)

Il livello dei blocchi logici (LBA Logical Block Addresing) è un livello di astrazione che offre una visione unificata dei dispositivi di storage locali, esponendo **blocchi logici numerati** tipicamente formati da 512 bytes o, più spesso, **4096 bytes** (4KB):

```
LBA 0, LBA 1, LBA 2, ...
```

Ad esempio i device /dev/sda /dev/sda1 /dev/sdb /dev/nvme0n1, /dev/dm-0, /dev/mapper/vg-lv sono device a blocchi.

Questo livello individua quale è il device driver a cui passare la richiesta, che potrebbe essere:

- **Un driver per l'hardware** (tipicamente moduli del kernel).
- **Un software di virtualizzazione locale**→ loop devices, volumi LVM, file immagine.
- **Un target iSCSI** o Fibre Channel→ il server esporta LBA virtuali, cioè l'iSCSI espone blocchi **logici**, non fisici.

Quando arriva una richiesta di lettura o scrittura su un blocco logico, questa viene inoltrata allo specifico device driver.

4) Filesystem – strutture dati

Il filesystem organizza i blocchi logici forniti dai dischi in strutture dati per mantenere le informazioni su files e directory.

La formattazione è una operazione preliminare che organizza i blocchi logici dei dischi ma non li crea e non ne cambia la dimensione, sono stati già creati dal livello sottostante.

La formattazione viene svolta in due momenti diversi, prima a livello di device driver, poi a livello di filesystem.

- La **formattazione del disco** crea **le partizioni nel disco** ed è realizzata dal device driver.
- **La formattazione del filesystem** crea **il filesystem dentro una partizione** (sudo mkfs.ext4, mkfs.xfs, …) :
  - crea nella partizione i "Blocchi del Filesystem" e li organizza in strutture dati, inizializzando il contenuto di alcune di queste strutture.

Ad esempio, il filesystem di tipo ext4 (Extended Filesystem 4) è suddiviso in:

- **Blocchi del filesystem**, normalmente 4KB come i blocchi logici.
  Contengono:
  - **Inode** (piccoli, 256 B, molti inode stanno in un blocco del filesystem) contengono i metadati sui file, ma non il nome del file.
  - **Blocchi dati** (1 blocco dati sta in 1 blocco del filesystem) contengono i dati di file e directory.
  - **Strutture dati del filesystem** organizzate in più blocchi del fs.

I blocchi del filesystem sono organizzati in strutture, in ext4 abbiamo:

- **Gruppi di Blocchi** (**Block Group**) insiemi di blocchi del filesystem che costituiscono l'entità minima di una partizione. Normalmente una partizione ha più Block Groups. Partizioni piccole possono avere un solo Block Group. Ogni Block Group contiene:
    - **Superblocchi** (formati da più blocchi del filesystem) contengono informazioni sul filesystem di ciascuna partizione. Esiste un **superblocco principale della partizione,** spesso collocato in una posizione specifica del primo blocco logico della partizione, e alcuni superblocchi di backup contenuti negli altri Gruppi di Blocchi (Block Group).
    - **Bitmap degli Inode** (stanno in più blocchi del fs), indicano gli inode liberi e occupati.
    - **Bitmap dei blocchi** (stanno in più blocchi del fs), indicano i blocchi liberi e occupati.
    - **Tabella degli Inode** (**Inode Table)** insieme di blocchi del filesystem che contengono gli inode, più inode in ciascun blocco.
    - Altre strutture interne.

Gli Inode sono strutture che memorizzano informazioni sui file:

- proprietario
- permessi
- timestamp
- puntatori ai blocchi dei dati

NB: Gli inode non contengono i dati, ma gli indici dei blocchi in cui sono contenuti i dati.

5) Filesystem vero e proprio – Implementa operazioni standard del filesystem (open, read, write, seek…)

Ogni filesystem implementa i modi per accedere ai propri file e directory.
Se il filesystem usa una qualche **forma di ridondanza software**, ad esempio salvando le informazioni in più locazioni/dischi come **zfs** (Zettabyte filesystem), oppure se alla ridondanza **aggiunge scritture incrementali** (come il filesystem **btrfs**, B-tree filesystem, usato per memorizzare immagini di container in forma efficiente) allora la lettura e scrittura su file diventa una operazione molto complessa che .
Analogamente, se il filesystem è un filesystem di rete, questo espone dei file che risiedono in macchine remote, perciò, l'accesso ai file coinvolgerà delle comunicazioni via rete.

Per organizzare e semplificare le operazioni di accesso ai file in queste situazioni complesse, **ogni filesystem** espone delle API di accesso ai file, cioè **implementa delle operazioni base che consentono di accedere ai file.** Ogni filesystem le implementa a modo suo.
Queste operazioni vengono standardizzate dal kernel Linux tramite **VFS** (Virtual File System) e sono principalmente **open, read, write, seek, close** e molte altre.

✓ open()
Apre un file e restituisce un file descriptor.
VFS cerca l'inode, verifica i permessi, prepara la struttura `struct file`.

✓ read()
Il kernel:

1. Legge i blocchi del filesystem
2. Traducendoli in richieste LBA
3. Il driver del disco li converte in accessi fisici/pagine

✓ write()
Simile a read, ma:

- Il filesystem decide dove allocare i blocchi
- Update dell'inode
- Flush su disco (con ext4 → journaling)

✓ seek()
Modifica il file offset nella struttura `struct file` gestita da VFS,
non accede al disco.

6) Il Virtual File System (VFS)

Il VFS è un livello di astrazione che:

- uniforma tutti i filesystem
- fornisce un'unica API al kernel
- gestisce inode generici (`struct inode`)
- gestisce operazioni uniformi (`read`, `write`, `mmap`, `open`…)

VFS permette di montare filesystem completamente diversi tra loro:

- ext4
- vfat
- xfs
- btrfs
- NFS File system distribuito (più esatto in rete che distribuito)
- SMB Accesso a file remoti per windows.
- Procfs
- sysfs
- tmpfs

In Linux **tutto è un file**, e questo è possibile grazie al VFS.

7) Applicazioni, librerie, system call - Call di alto livello (open/read/write/seek)

Esempio tipico di operazione su file.

Quando un'applicazione fa una operazione di lettura su un file dopo averlo aperto ed avere così ottenuto un file descriptor:

```
read(fd, buf, 4096);
```

succede:

0. L'applicazione chiama la funzione di libreria che chiama la system call.
1. La chiamata arriva al kernel e viene passata al VFS
2. Il VFS delega al filesystem specifico (ext4 ecc.)
   2.1. Se il filesystem è un **filesystem remoto**, la richiesta viene mandata al livello VFS del server remoto via rete.
   2.2. Se in filesystem è locale la richiesta viene gestita dal modulo del kernel che implementa lo specifico filesystem.
3. Il filesystem individua l'inode e i blocchi FS
4. Questi vengono tradotti in blocchi logici LBA
5. Il driver del disco converte gli LBA in operazioni fisiche.
   5.1. Se i blocchi sono contenuti in uno **storage a blocchi** remoto allora la richiesta viene inviata via rete allo storage a blocchi remoto.
6. Il disco (HDD/SSD) legge la pagina/settore reale

Dettagli: “File System di rete” e “Storage a blocchi in rete”

Sono dei sistemi che consentono di usare dei file collocati su delle macchine remote ma che si collocano a due livelli architetturali diversi e espongono entità diverse.

STORAGE A BLOCCHI (espongono livello 3 LBA)
Gli **storage a blocchi in rete** (**Block Storage**) consentono di esportare delle entità viste come dischi remoti, cioè come device formati da blocchi logici LBA. Questi dischi (volumi) logici sono comunemente chiamati **LUN** (**Logical Unit Number**) che vengono presentati dai server ai client. Ogni LUN è visto dal sistema operativo del client come un disco locale non formattato. Occorre un protocollo che specifichi come vengono effettuate le comunicazioni e come vengono fatte le richieste di accesso ai blocchi logici. Una volta che sono messi a disposizione del client, questi dischi remoti possono essere formattati con un filesystem dal sistema operativo del client ed usati per archiviare file e directory.
Gestione della atomicità delle operazioni di lettura/scrittura: Normalmente questi protocolli di accesso ai LUN **non si occupano della atomicità delle operazioni**: se due client scrivono sullo stesso blocco possono causare problemi e addirittura la corruzione del filesystem realizzato sui blocchi. I sistemi di storage a blocchi forniti dai cloud provider risolvono brutalmente impedendo a più VM di accedere allo stesso LUN. Per condividere atomicamente storage a blocchi si può implementare, nei blocchi sopra ai client, dei filesystem speciali, detti Filesystem a cluster (Cluster File System - CFS) che realizzano un sistema di lock distribuito: i client si conoscono e comunicano tra loro, prima di modificare un file il client blocca l’inode e scrive solo quando nessun altro sta scrivendo sul file.

Tra i protocolli più usati per usare LUN remoti ricordiamo:

- **iSCSI (Internet Small Computer System Interface):** Il più diffuso su reti IP standard (Ethernet). Consente di incapsulare i comandi SCSI (Standard Command Set for storage devices) su pacchetti TCP/IP.
- **Fibre Channel (FC):** Un protocollo di rete ad alta velocità e bassa latenza, richiede una rete dedicata con switch FC e schede HBA (Host Bus Adapter) dedicate sui server. Offre le massime prestazioni e affidabilità.
- **NVMe-oF (NVMe over Fabrics):** Un protocollo più recente e all'avanguardia che estende l'efficienza del protocollo NVMe (Non-Volatile Memory Express) per SSD su vari "fabrics" di rete (Ethernet, Fibre Channel, InfiniBand). Offre latenza estremamente bassa e throughput elevato.

- **Cloud Provider Block Storage (Managed Services):**
  - **AWS Elastic Block Store (EBS):** Volumi persistenti ad alte prestazioni per istanze EC2.
  - **Google Persistent Disk:** Dischi virtuali ad alta disponibilità per Google Cloud.
  - **Azure Managed Disks:** Dischi gestiti per macchine virtuali Azure.
- **Soluzioni On-Premise / Open Source:**

- **Ceph RBD (RADOS Block Device):** Implementazione robusta e scalabile di block storage distribuito, parte dell'ecosistema Ceph. Molto popolare in ambienti cloud privati (OpenStack, Proxmox).
- **Linstor:** Soluzione open source per la gestione centralizzata del block storage su cluster Linux.
- **Longhorn:** specificamente progettato per Kubernetes, molto attivo.

Viene chiamata SAN (Storage Area Network) una rete che connette degli storage a blocchi ad alcuni server a cui fornisce l'accesso a livello di blocco.

Protocollo iSCSI per storage "a blocchi" in rete
Un server iSCSI è uno storage di rete che può esportare verso i client:
- un proprio file
- un proprio device
- un proprio LVM
- un proprio disco fisico

Tutti queste entità vengono esposte facendole vedere come se fossero costituite da **LBA logici** al client.
Il client vede un "disco" e ci può creare un filesystem che avrà:
- i suoi blocchi
- i suoi inode-

Quindi iSCSI è un protocollo di rete che stabilisce come esporre e come accedere a dei "dischi" costituiti da LBA virtuali.

FILESYSTEM DI RETE o meglio DISTRIBUTED FILESYSTEM (espongono livello 5 filesystem)
Un Distributed Filesystem estende il concetto di filesystem tradizionale su una rete, permettendo a più client di accedere e condividere gli stessi file e directory contemporaneamente, come se fossero su un disco locale. Offre un'interfaccia POSIX standard (Portable Operating System Interface).
Un DFS crea un'unica gerarchia di directory e file che è accessibile da più nodi client. A differenza del Block Storage, il DFS gestisce la struttura di file e directory, i metadati e la distribuzione dei dati tra i nodi di storage del cluster.

1. **Architettura del Cluster:** Un cluster DFS è composto da più nodi, alcuni dei quali possono fungere da server di metadati (per la gestione di nomi file, permessi, gerarchie) e altri da server di dati (per l'archiviazione dei blocchi effettivi dei file).
2. **Consistenza:** Il DFS garantisce la coerenza dei dati tra i client. Se un client modifica un file, la modifica è immediatamente visibile (o quasi) a tutti gli altri client che accedono allo stesso file. Questo avviene tramite meccanismi di locking e sincronizzazione distribuiti.
3. **Accesso Client:** I client montano il filesystem distribuito usando un protocollo di rete (NFS, SMB) o un modulo FUSE, e lo vedono come un normale volume locale.

Quali sono i protocolli più utilizzati per realizzare filesystem distribuiti
- **NFS (Network File System):** Il protocollo standard per la condivisione di file in ambienti Unix/Linux. Ampiamente supportato.
- **SMB/CIFS (Server Message Block/Common Internet File System):** Il protocollo standard per la condivisione di file in ambienti Windows.

- **FUSE (Filesystem in Userspace):** Permette di implementare filesystem nello spazio utente invece che nei moduli del kernel, utile per montare DFS specifici che non supportano nativamente NFS/SMB (es. il client CephFS può essere un modulo FUSE).
- **CephFS** (**Ceph Filesystem**): Il filesystem distribuito di Ceph. Molto robusto, scalabile e versatile, ben mantenuto. È la scelta open source di riferimento per un DFS moderno.
- **MooseFS / LizardFS**: Relativamente più semplici da implementare e gestire rispetto a CephFS, con un'architettura più leggera (con un server di metadati dedicato). Ancora attivamente mantenuti e utilizzati.

Dettagli: Filesystem virtuali: procfs, sysfs, devtmpfs, tmpfs, fuse

procfs (`/proc`)

- espone informazioni sui processi e sul kernel

sysfs (`/sys`)

- espone informazioni strutturate su dispositivi e driver

devtmpfs (`/dev`)

- contiene i device nodes dinamici

tmpfs

- filesystem in RAM
- usato per /run, /dev/shm

fuse

- filesystem implementati in spazio utente

Dettagli: Loop device

I **loop device** (o dispositivi loopback) sono dei device driver speciali nel kernel Linux che permettono di **montare un file come se fosse un dispositivo a blocchi** (come un disco rigido o una partizione).
In sostanza, un loop device prende un file regolare presente nel filesystem (spesso un file di immagine, come un file .iso .img o un disco virtuale e lo rende accessibile tramite un'interfaccia a blocchi (ad esempio, /dev/loop).

Come Funzionano
Il loop device agisce come un **traduttore** a livello del kernel tra due astrazioni:

1. **File Regolare:** Visto dal filesystem come un semplice insieme di dati (la risorsa).
2. **Dispositivo a Blocchi:** Visto dal sistema operativo come un'unità di archiviazione grezza (il target).

Processo Tipico

1. **Associazione:** Si associa un file immagine (es. disco_virtuale.img) a un loop device libero (es. /dev/loop0).
2. **Lettura/Scrittura:** Quando il sistema operativo tenta di leggere o scrivere un blocco sul dispositivo /dev/loop0 , il driver del loop device intercetta queste richieste.
3. **Mappatura:** Il driver traduce l'accesso al blocco virtuale in operazioni di lettura/scrittura di byte all'interno del file disco_virtuale.img.
4. **Montaggio:** Una volta creato il dispositivo /dev/loop0, il sistema può formattarlo o montarlo con un filesystem (come ext4) come farebbe con una vera partizione.

Usi Comuni dei Loop Device
I loop device sono fondamentali per molte operazioni standard in Linux:

- **Montaggio di Immagini Disco:** Montare file .iso CD/DVD per accedere ai file (ad esempio i file di installazione di una distribuzione) all'interno di un'immagine di disco avviabile.
- **Dischi Virtuali:** Creare e montare file di grandi dimensioni che fungono da dischi per macchine virtuali.
- **Snapcraft/Snap:** Il sistema di pacchettizzazione **Snap** di Canonical utilizza i loop device per montare i singoli pacchetti (.snap files) in modo isolato come filesystem di sola lettura.
- **Cryptsetup:** I dispositivi crittografati (LUKs) spesso utilizzano i loop device per mappare un file contenente un volume crittografato.