RIASSUNTO PER L'INTERROGAZIONE

EXT4 vs NTFS • Il Boot • User Space vs Kernel Space

---

1. EXT4 vs NTFS

Cos'è un File System?

Il file system è il sistema che organizza i dati su disco. Senza di esso il disco sarebbe una sequenza caotica di bit. Gestisce file, cartelle, permessi e spazio libero.

Le origini

EXT4 — nasce nel 2008, è il file system nativo di **Linux**. Open source, evoluzione di ext3.

NTFS — nasce nel 1993, è il file system nativo di **Windows**. Proprietario Microsoft.

Confronto diretto

Caratteristica	EXT4	NTFS
Sistema nativo	Linux	Windows
Anno	2008	1993
Dim. max file	16 TB	16 EB (praticamente illimitato)
Journaling	✔ Sì	✔ Sì
Case sensitive	✔ Sì (File.txt ≠ file.txt)	✖ No
Crittografia nativa	✖ No	✔ BitLocker
Compressione nativa	✖ No	✔ Sì
Permessi	Unix (rwx)	ACL Windows
Performance su Linux	🚀 Eccellente	🐢 Lenta (driver esterno)
Performance su Windows	🐢 Solo con software extra	🚀 Eccellente

Differenze chiave da ricordare

- **Case sensitivity:** EXT4 distingue maiuscole/minuscole nei nomi file. NTFS no.
- **Journaling:** Entrambi lo usano — è un registro delle operazioni per recuperare dati dopo un crash.
- **Permessi:** EXT4 usa il sistema Unix (rwx). NTFS usa le ACL di Windows, più complesse.
- **Compatibilità:** EXT4 non si apre nativamente su Windows. NTFS su Linux richiede driver esterni.

Quando usare quale

> 🔔 *Solo Linux → EXT4 | Solo Windows → NTFS | Dual boot → una partizione exFAT condivisa | Chiavetta USB → exFAT (compatibile con tutti)*

---

2. Cosa succede durante il Boot?

Il boot è la sequenza di operazioni che avviene dall'accensione fino a quando il sistema è pronto. È un percorso **a staffetta**: ogni fase prepara la successiva.

Le 7 fasi del Boot

Fase	Nome	Cosa fa
1	Alimentazione (Power On)	L'alimentatore dà corrente. La CPU si sveglia ma non sa ancora cosa fare.
2	POST	Power-On Self Test: controlla che RAM, CPU, disco e tastiera funzionino. Segnala errori con beep.
3	BIOS / UEFI	Cerca il dispositivo da cui avviare il sistema (disco, USB...) seguendo una lista di priorità.
4	Bootloader	Piccolo programma (GRUB su Linux, Windows Boot Manager su Windows) che carica il kernel.
5	Kernel	Si inizializza, rileva l'hardware, carica i driver, monta il file system root.
6	Systemd / Init	Il primo processo (PID 1) avvia tutti i servizi: rete, audio, login manager...
7	Login → Desktop	Appare la schermata di login. Inserite le credenziali, parte il desktop.

BIOS vs UEFI — differenze chiave

	BIOS	UEFI
Età	Anni '80	Anni 2000
Interfaccia	Solo testo	Grafica, supporta il mouse
Disco max	2 TB (MBR)	9 ZB (GPT)
Secure Boot	✕ No	✓ Sì
Velocità boot	Lenta	Molto più rapida

Schema visivo del Boot

Accensione → POST (controllo HW) → BIOS/UEFI (trova il disco) → Bootloader (GRUB/WBM) → Kernel (carica driver) → Systemd (avvia servizi) → Login → Desktop

---

3. User Space vs Kernel Space

Il sistema operativo divide la memoria in due zone **separate e protette**. È una delle scelte architetturali più importanti per sicurezza e stabilità.

Kernel Space

Zona di memoria **riservata esclusivamente al kernel**. Chi opera qui ha accesso diretto e illimitato a tutto l'hardware — CPU, RAM, dispositivi. Non ci sono restrizioni.

Cosa gira nel Kernel Space:

- Il kernel stesso
- I driver dei dispositivi
- Lo scheduler dei processi
- Il gestore della memoria

User Space

Zona dove girano **tutte le applicazioni normali**. Chi opera qui **NON può toccare l'hardware direttamente**. Deve chiedere al kernel tramite le System Call.

Cosa gira nello User Space:

- Browser, editor di testo, giochi, terminale
- Le librerie di sistema (es. glibc su Linux)
- La shell (bash, zsh)
- L'interfaccia grafica

Perché questa separazione?

Motivo	Spiegazione
🔒 Sicurezza	Un programma malevolo nello User Space non può leggere la memoria di altri processi o danneggiare l'hardware.
⚙ Stabilità	Se un'app crasha nello User Space, il kernel la termina e il sistema continua. Un crash nel Kernel Space invece causa il Blue Screen (Windows) o il Kernel Panic (Linux/macOS).

Le System Call — il ponte tra i due mondi

Quando un'app ha bisogno di qualcosa di privilegiato (aprire un file, connettersi a internet, allocare memoria), fa una **System Call** — una richiesta formale al kernel.

> *Come funziona: App (User Space) → richiede "apri questo file" → System Call (open, read, write...) → Kernel accede fisicamente al disco → restituisce i dati all'app*

Confronto diretto

	Kernel Space	User Space
Accesso HW	✓ Diretto e illimitato	✕ Solo tramite System Call
Privilegi	Massimi (ring 0)	Limitati (ring 3)
Crash conseguenze	Intero sistema (BSOD/Kernel Panic)	Solo il processo crashato
Cosa gira	Kernel, driver, scheduler	App, browser, shell, GUI
Esempio System Call	—	open(), read(), write(), fork()

Schema

[USER SPACE] Browser | App | Shell | Librerie ——————————— SYSTEM CALL API ——————————— [KERNEL SPACE] Scheduler | Memory Manager | Driver | File System | Hardware

---

RIEPILOGO VELOCE

Argomento	Cosa ricordare
EXT4 vs NTFS	EXT4 = Linux, open source, case sensitive. NTFS = Windows, proprietario, BitLocker. Fuori dal loro ambiente sono lenti.
Il Boot	7 fasi: Alimentazione → POST → BIOS/UEFI → Bootloader → Kernel → Systemd → Login. Ogni fase prepara la successiva.
User/Kernel Space	Kernel space = accesso totale all'HW. User space = app normali, accesso solo tramite System Call. Separazione = sicurezza + stabilità.
System Call	Il ponte tra User e Kernel space. Le app chiedono al kernel di fare operazioni privilegiate (open, read, write, fork...).