1 I PROCESSI E GLI ALGORITMI DI SCHEDULING

1.1 La gestione dei processi

Uno dei compiti fondamentali del sistema operativo è quello di associare un processo a ogni programma in esecuzione.

Mentre il **programma** è un insieme statico di istruzioni, cioè un'entità passiva, che descrive le azioni da compiere, il **processo**, o task, è un'entità attiva che rappresenta l'esecuzione delle azioni specificate dal programma. A un processo va assegnato il tempo di CPU e la memoria di cui necessita per eseguire le istruzioni del programma.

#prendinota

Per ottimizzare l'uso delle risorse (prima di tutto la CPU), ogni processo può essere suddiviso in sottoprocessi che prendono il nome di **thread** e che possono essere eseguiti in maniera indipendente.

Rivestono allora particolare importanza i moduli del kernel (sempre residenti in memoria centrale) che si occupano dell'assegnazione del tempo di CPU e dello spazio di memoria ai singoli processi.
Tra questi moduli, i più importanti sono:

- lo **schedulatore dei lavori**, che seleziona quali programmi caricare in memoria centrale per poter essere eseguiti (diventando dei processi), concorrendo all'uso delle risorse;
- lo **schedulatore dei processi**, che decide a quale dei processi creati assegnare la CPU;
- il **controllore del traffico**, che controlla l'avanzamento dei processi e la gestione delle transizioni da uno stato all'altro.

Ma durante la sua permanenza nel sistema, un processo può necessitare dell'uso di molti altri moduli riassunti nel modello a strati (ring) del sistema operativo (FIGURA 1).

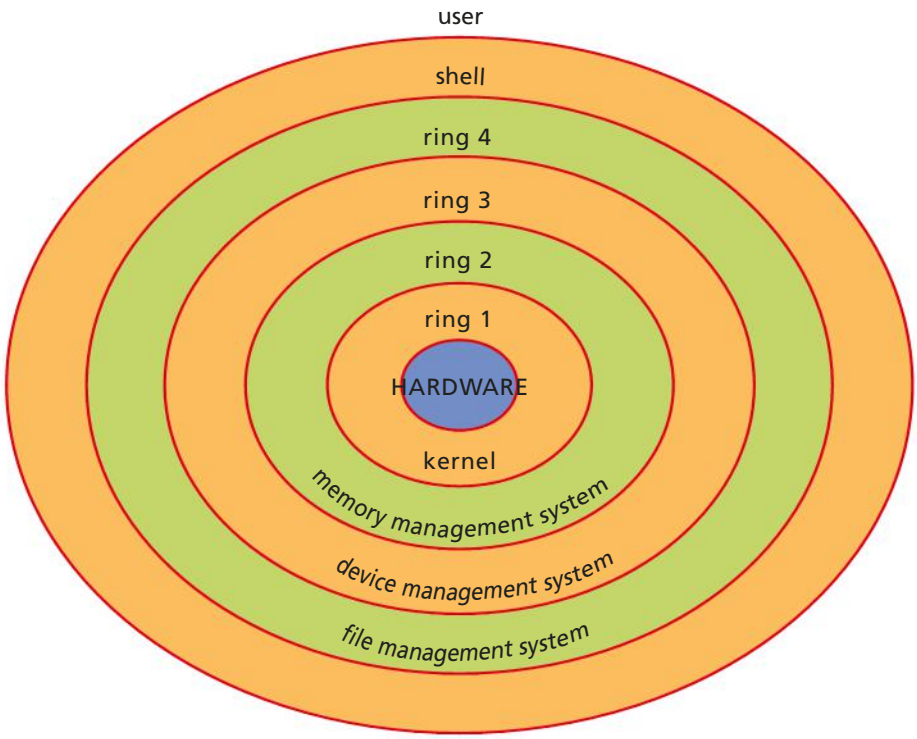


FIGURA 1 Modello a strati del sistema operativo

Se il processo prevede delle elaborazioni, il **memory management system** (ring 2) dovrà allocare, in ogni momento, la quantità di memoria centrale necessaria al processo. Se il processo richiede delle operazioni di I/O, il **device management system** (ring 3) dovrà assegnare la risorsa di I/O al processo in base alle politiche di gestione prestabilite. Se il processo necessita di archiviare i dati su memoria secondaria, il **file management system** (ring 4) ne consentirà l'organizzazione in file e cartelle.

Un processo si trova, in ogni istante, in uno **stato** ben preciso, cioè in una condizione particolare determinata da diversi fattori quali le esigenze del processo stesso, il punto in cui è arrivata la sua esecuzione e la situazione globale del sistema (cioè le altre componenti hardware e software).
I possibili stati in cui può trovarsi un processo sono:

- **HOLD**: il programma è stato proposto al sistema, è in attesa di esecuzione e si trova su memoria secondaria;
- **READY**: il programma è diventato processo e si trova in memoria centrale, pronto per essere eseguito;
- **RUN**: il processo è in esecuzione (ricordiamo che in ogni istante un solo processo si trova in questo stato, cioè gli è stata assegnata l'unica CPU);
- **WAIT**: il processo è in attesa (per esempio deve attendere la fine di un'operazione di I/O);
- **END**: il processo è terminato.

Nella FIGURA 2 è illustrato il diagramma degli stati di un processo. In esso sono rappresentati tutti gli stati e le 6 possibili transizioni.

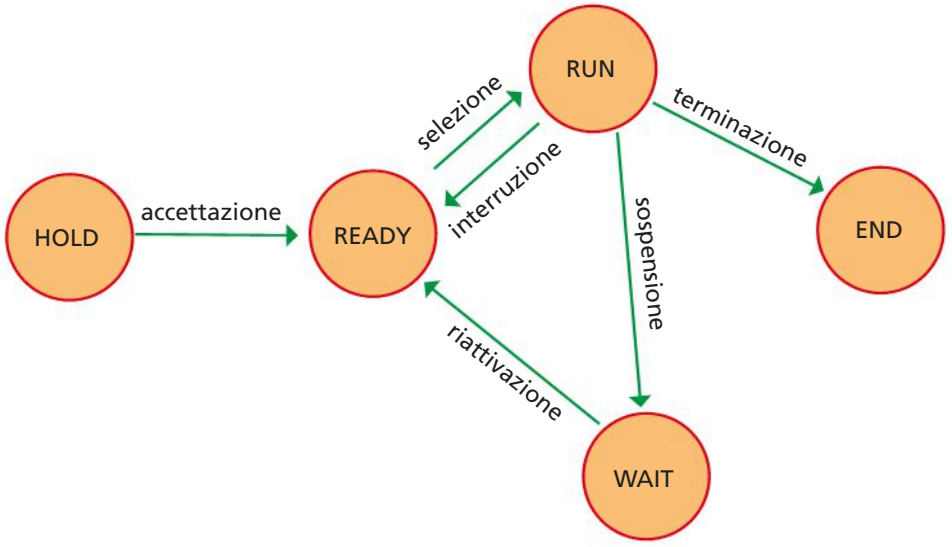


FIGURA 2 Diagramma degli stati di un processo

A ogni stato è associata una **lista** che contiene informazioni su ogni processo che si trova in quel particolare stato in quel momento.

Le due liste più importanti sono la lista di HOLD e la lista di READY che gestiscono l'entrata del processo in esecuzione.
La **lista di HOLD** contiene una serie di elementi descrittori detti **JCB** (**Job Control Block**), uno per ogni programma (job) in questo stato.

Generalmente un JCB contiene:
- l'identificativo del programma;
- l'occupazione espressa in kilobyte;
- l'indirizzo di caricamento su memoria di massa;
- gli identificatori delle periferiche;
- il puntatore al JCB successivo.

La **lista di READY**, invece, contiene dei descrittori detti **PCB** (**Process Control Block**), uno per ogni processo (process) in questo stato, che vengono generati all'atto della creazione del processo e memorizzati in memoria centrale per tutta la vita del processo.
Generalmente un PCB contiene:
- PID (Process IDentifier), l'identificatore del processo, cioè un numero progressivo;
- lo stato in cui si trova: RUN, READY, WAIT, …;
- il registro di salvataggio;
- l'indirizzo di caricamento in memoria centrale;
- gli identificatori delle periferiche;
- il puntatore alla lista dei figli;
- il puntatore al PCB successivo.

#prendinota

Una **sequenza di transizioni** inizia sempre con accettazione → selezione. Le successive transizioni (interruzione, sospensione, riattivazione e ancora selezione) dipendono dal sistema operativo, possono ripetersi più volte e si concludono con la terminazione.

Grazie a queste due strutture dati di supporto, la sequenza di transizioni può essere controllata e gestita dal sistema operativo attraverso i moduli del kernel.

Quando un programma viene proposto al sistema si trova nello stato di HOLD. Il modulo kernel **schedulatore dei lavori** sceglie, tra tutti i programmi che si trovano in questo stato, quale portare in memoria centrale passando così allo stato di READY (**accettazione: HOLD → READY**).
A questo punto, un altro modulo kernel (il **controllore del traffico**) elimina il suo JCB dalla lista di HOLD e inserisce un nuovo PCB nella lista di READY.
Infine, quando la CPU è libera, il modulo kernel **schedulatore dei processi** sceglie quale, fra i processi che si trovano nello stato di READY, mandare in esecuzione assegnandogli la CPU (**selezione: READY → RUN**).
Questa volta il controllore del traffico deve modificare lo stato contenuto nel PCB del processo portandolo in RUN.

Quando il processo è in esecuzione si possono verificare degli eventi che portano a liberare il processore. Questi eventi sono:

a. terminazione del processo in esecuzione;

b. scadenza time slice (tempo di CPU destinato al processo);

c. richiesta di operazione di I/O.

Esaminiamoli separatamente:

a. quando viene eseguita l'istruzione di fine, il processo passa nello stato di END (**terminazione: RUN → END**). Per il sistema operativo questo processo non esiste più e quindi deve provvedere a eliminare il suo PCB e a dichiarare libera la memoria centrale che occupava;

b. se, durante l'esecuzione, viene esaurito il time slice a lui destinato, il processo che era in esecuzione deve liberare il processore. Lo stato contenuto nel PCB viene portato in READY (**interruzione: RUN → READY**);

c. se durante l'esecuzione il processo richiede un'operazione di I/O, esso viene portato nello stato di WAIT, perché per proseguire deve aspettare la fine dell'operazione (**sospensione: RUN → WAIT**). Solo quando quest'operazione è terminata il processo potrebbe tornare nello stato di RUN, ma poiché la CPU potrebbe essere occupata con un altro processo, esso dovrà prima ritornare nello stato di READY (**riattivazione: WAIT → READY**).

1.2 Gli algoritmi di scheduling dei processi

Uno dei compiti principali del sistema operativo è quello di assegnare la CPU ai processi. Per fare questo abbiamo visto che il sistema operativo utilizza un modulo del kernel: lo **schedulatore** (o pianificatore) **di processi**.
Lo schedulatore di processi deve decidere a quale processo in stato di READY, quindi col suo PCB in coda nella lista di READY, assegnare la CPU, facendolo così passare allo stato di RUN.
In generale ogni algoritmo di schedulazione deve soddisfare almeno le seguenti sei regole:

1. **imparzialità**: ogni processo deve essere concesso equamente alla CPU;
2. **efficienza**: la CPU deve essere mantenuta occupata nei valori percentuali più alti possibili;
3. **tempo reale**: gli utenti interattivi devono veder minimizzati i tempi di risposta;
4. **tournaround**: gli utenti batch devono veder minimizzati i tempi di elaborazione;
5. **throughput**: si deve massimizzare il numero di processi completati nell'unità di tempo;
6. **fairness**: si deve evitare che alcuni processi non siano mai stati, o siano troppo raramente, schedulati.

Lo **schedulatore dei processi** è in grado di compiere questa funzione secondo specifiche politiche di scheduling che si suddividono in:
- **non preemptive** se la CPU, una volta assegnata a un processo, non può essere tolta fino a quando il processo non è terminato o richiede un'operazione di I/O;
- **preempitive** se la CPU può essere tolta al processo anche quando è in esecuzione.

#prendinota

Facendo una similitudine con le code alle casse di un supermercato, le politiche non preemptive vengono così applicate:
- **FCFS**: cassa standard;
- **SJF**: cassa veloce per spese con pochi elementi;
- **Scheduling con priorità**: cassa riservata alle donne incinte e ai disabili.

Le principali politiche di scheduling **non preemptive** sono:
- **First Come-First Served (FCFS)**: la CPU viene assegnata al processo che si trova alla testa della lista dei processi pronti (lista di READY), quindi il primo processo arrivato è il primo processo a essere servito. Svantaggi: il tempo medio di attesa è piuttosto lungo e sono penalizzati i processi brevi.
- **Shortest Job First (SJF)**: si assegna la CPU al processo che ha il più breve tempo di utilizzo della CPU prima di un'operazione di I/O. Questo algoritmo rende minimo il tempo medio di attesa, ma è difficile sapere l'esatta durata della successiva sequenza, si può solo farne una stima.
- **Scheduling con priorità**: si associa una priorità a ogni processo e si assegna la CPU a quello con priorità più alta. Le priorità possono essere calcolate tenendo conto sia del tempo di esecuzione che del tempo di attesa nella lista di READY.

Le principali politiche di scheduling **preemptive** sono:

- **Round-robin**: il processore viene assegnato a turno, per un intervallo di tempo stabilito (time slice), ai processi nell'ordine in cui questi ne hanno fatto richiesta. La coda dei processi nella lista di READY è gestita col metodo FIFO (First In First Out) e quindi tutti gli inserimenti provenienti dalla lista di HOLD, dalla lista di WAIT o quelli causati dall'esaurimento di un time slice avvengono al fondo.
- **Round-robin a percentuale di tempo**: con la tecnica precedente i processi che richiedono molte operazioni di I/O sono penalizzati rispetto a quelli che ne richiedono di meno, perché il rientro dallo stato di WAIT significa l'inserimento del processo al fondo della lista. La tecnica a percentuale di tempo è una variante del Round-robin: il processo non verrà inserito necessariamente al fondo, ma la posizione di rientro nella lista dipende dalla percentuale di tempo di CPU utilizzata. Questo significa che se un processo lascia il processore perché richiede un I/O e quindi non ha ancora esaurito il proprio time slice, il suo rientro nella lista non sarà più al fondo ma tanto più avanti quanto minore è la percentuale di tempo di CPU che ha utilizzato prima dell'interruzione.
- **Round-robin limitato**: è ancora una variante del Round-robin. È prefissato il numero di time slice che un processo può utilizzare: terminati questi time slice che sono gestiti con la tecnica del Round-robin tradizionale, il processo è inserito in una nuova lista (lista di READY 2) che sarà presa in considerazione solo quando la lista del Round-robin (lista di READY 1) verrà esaurita.

Confrontando lo scheduling non preemptive con quello preemptive, possiamo dire che:

- lo scheduling non preemptive utilizza algoritmi semplici, dovendo assegnare la CPU ai processi solo quando la CPU è libera da elaborazioni. In generale gli algoritmi non preemptive sono meno performanti rispetto a quelli preemptive, tranne nel caso di presenza di molti processi brevi o con molte operazioni di I/O. Le politiche non preemptive tendono a privilegiare i processi brevi e quelli che attendono da più tempo;
- lo scheduling preemptive utilizza algoritmi più complessi perché deve pianificare il passaggio della CPU da un processo all'altro interrompendo la CPU stessa, anche mentre è impegnata in altre elaborazioni.

FISSA E APPLICA LE TUE CONOSCENZE

1. **Qual è la differenza tra programma, processo e thread?**
2. **Quale modulo schedulatore del kernel decide quali programmi caricare in memoria centrale?**
3. **Come si chiamano, dall'interno verso l'esterno, i 4 ring del modello a strati del sistema operativo?**
4. **Quale transizione di stato implica il passaggio dallo stato RUN allo stato WAIT?**
5. **Quali transizioni di stato implicano il passaggio allo stato READY?**
6. **Descrivi la politica di scheduling non preemptive Shortest Job First (SJF).**
7. **Descrivi la politica di scheduling preemptive Round-robin.**
8. **Quali differenze ci sono tra lo scheduling non preemptive e quello preemptive?**

2 LA GESTIONE DELLA MEMORIA FISICA E VIRTUALE

2.1 La rilocazione e l'allocazione degli indirizzi fisici

La principale risorsa che il sistema operativo deve gestire (con la collaborazione della CPU) è la memoria centrale. Tale risorsa, generalmente identificabile con la memoria **fisica** di tipo RAM a disposizione del sistema di elaborazione, è suddivisa in gruppi di byte, ciascuno con il suo indirizzo.

> Definiamo **spazio degli indirizzi** di un programma la zona di memoria centrale formata dalla zona dedicata ai dati del programma più la zona dedicata alle sue istruzioni.

Qualsiasi istruzione o dato all'interno dello spazio degli indirizzi assegnato a un programma, prima di essere elaborato, deve quindi passare attraverso vari stadi in cui gli indirizzi vengono rappresentati in modi diversi:

- **indirizzi simbolici**: sono i nomi usati dai programmatori per definire variabili, procedure, oggetti, programmi, ecc. In essi non viene fatto alcun riferimento alle locazioni della memoria centrale. Lo sviluppatore di codice non si preoccupa mai di specificare gli indirizzi di memoria, lasciando questo compito a programmi come il compilatore e il linker. Gli indirizzi simbolici sono dunque stabiliti dall'uomo e vanno scelti in modo da rendere facilmente leggibile il codice sorgente;
- **indirizzi logici**: nella fase di assemblaggio (passaggio da codice sorgente a codice macchina) gli indirizzi simbolici del codice sorgente vengono trasformati in forma binaria; i loro contenuti sono ancora logici, cioè passano da simbolici a numerici, ma non contengono ancora gli indirizzi delle celle di memoria fisica in cui il programma verrà memorizzato durante l'esecuzione. Gli indirizzi logici sono generati e assegnati dalla CPU;
- **indirizzi virtuali**: sono gli indirizzi logici in presenza di tecniche di virtualizzazione della memoria. Poiché ormai tutti i sistemi operativi sono in grado di gestire la memoria virtuale, l'espressione "indirizzi virtuali" ha praticamente sostituito quella di "indirizzi logici". Quindi anche gli indirizzi virtuali passano da simbolici a numerici ma non contengono ancora gli indirizzi delle celle di memoria fisica in cui il programma verrà memorizzato durante l'esecuzione. Gli indirizzi virtuali sono generati e assegnati dalla CPU ma con tecniche differenti dagli indirizzi logici;
- **indirizzi fisici**: gli indirizzi logici o virtuali presenti nel file eseguibile sono tradotti in indirizzi di memoria fisica (**#rilocazione**). Ogni indirizzo fisico punta a un gruppo di byte ben preciso della RAM. È questo il momento in cui il programma è pronto a trasformarsi in processo ed essere eseguito. L'abbinamento tra gli indirizzi logici o virtuali e i corrispondenti indirizzi fisici è fatto dall'unità hardware di gestione della memoria **MMU** (Memory Management Unit) seguendo le direttive del modulo memory management system del sistema operativo (ring 2 del modello a strati).

#techwords

Rilocazione
È l'operazione che traduce gli indirizzi logici o virtuali in indirizzi fisici.

La rilocazione degli indirizzi deve sempre essere fatta prima di eseguire un'istruzione di codice macchina.
Ci sono due modalità di rilocazione:

- la **rilocazione statica** è quel meccanismo per cui il sistema operativo, **prima di porlo in esecuzione**, rialloca tutto il codice adattando gli indirizzi logici o virtuali alle posizioni in quel momento disponibili nella memoria fisica. Richiede un elevato tempo di setup, ma poi l'elaborazione procede più spedita, avendo già generato tutti gli indirizzi fisici effettivi;
- la **rilocazione dinamica** invece è l'insieme dei meccanismi con i quali un sistema operativo rialloca il codice **durante l'esecuzione del programma (runtime)**. Il codice viene caricato in memoria in formato rilocabile in una delle zone disponibili e viene usato un registro base che memorizza l'indirizzo fisico effettivo della prima locazione di memoria. Durante l'esecuzione, a ogni accesso, si calcola l'indirizzo fisico effettivo del dato o dell'istruzione partendo dall'indirizzo base.

Il calcolo degli indirizzi fisici (rilocazione statica o dinamica) può essere fatto una volta sola o può essere fatto tante volte. Questo dipende dalla natura del sistema e dai processi che lo popolano:

- si definisce **allocazione statica** quando il calcolo degli indirizzi fisici (rilocazione statica o dinamica) viene fatto **una volta sola**. Si tratta quindi di sistemi in cui un processo viene caricato in memoria, eseguito e rimosso solo al termine dell'esecuzione;
- si definisce **allocazione dinamica** quando il calcolo degli indirizzi fisici (rilocazione statica o dinamica) può essere fatto **tante volte**. Si tratta quindi di sistemi in cui un processo può essere caricato (#swap-in) e scaricato (#swap-out) dalla memoria più volte.

I sistemi operativi in grado di effettuare continuamente operazioni di swapping possono gestire la memoria virtuale.

#techwords

Swap-in e swap-out
Con **swap-in** (caricare) si intende il passaggio di un programma o una parte di esso dalla memoria secondaria alla memoria centrale. Viceversa, con **swap-out** (scaricare) si intende il passaggio da memoria centrale a memoria secondaria.

2.2 La memoria virtuale

La possibilità di fare lo swapping dei processi in allocazione dinamica, unito alla possibilità di scrivere programmi suddivisi in moduli indipendenti, ha portato alla nascita della memoria virtuale.
La memoria virtuale è un meccanismo del sistema operativo per poter mandare in esecuzione un programma anche se è più grande della RAM a disposizione o tanti programmi contemporaneamente anche se tutti insieme occupano più della RAM a disposizione.

> L'idea è di trattare la memoria centrale come una cache della memoria secondaria: la RAM diventa una cache (molto grande per essere una cache) e l'HDD o SSD diventano una RAM (molto grande per essere una RAM).

Nelle cache, essendo memorie di dimensioni assai limitate, vengono caricati solo dati e istruzioni indispensabili e di immediato utilizzo da parte della CPU. Il resto rimane nella RAM.
Applicando lo stesso principio tra RAM e memoria secondaria, le tecniche di virtualizzazione della memoria caricano nella RAM solo quelle porzioni di programma che devono essere eseguite.

Per esempio, applicativi come elaboratori di testo o fogli di calcolo, dotati di ricchissimi menu, non vengono interamente caricati in RAM. Alcune routine saranno richiamate dalla memoria secondaria (swap-in) solo quando l'utente le avrà selezionate dal menu.
Allo stesso modo, programmi o parti di programma che hanno terminato o sospeso la loro esecuzione vanno scaricati dalla RAM (swap-out) per fare spazio.

Con la tecnica della memoria virtuale il sistema operativo può quindi accettare processi di qualsiasi dimensione che vengono "parcheggiati" in memoria secondaria portando in memoria centrale solo quelle porzioni che devono essere eseguite. Il vincolo della limitatezza fisica della memoria centrale viene così superato e sarà possibile, per esempio, far girare un processo che richiede 500 kB di memoria avendone a disposizione soltanto 200 kB.
La virtualizzazione della memoria ha i seguenti vantaggi:

- non c'è più la limitazione dovuta alla dimensione della memoria fisica: diventa quindi possibile eseguire programmi di qualsiasi dimensione;
- si aumenta notevolmente il livello di multiprogrammazione del sistema;
- si possono sfruttare, come conseguenza, in modo più efficiente tutte le altre risorse dell'elaboratore.

A fronte di questi vantaggi, il sistema operativo dovrà gestire frequenti operazioni di swapping individuandone le politiche e ricalcolando gli indirizzi fisici che di volta in volta possono cambiare.

2.3 Le tecniche di allocazione della memoria

Il modo in cui i processi vengono allocati in memoria centrale dipende innanzitutto dalla tecnica utilizzata dal sistema operativo per organizzare la suddivisione di tale memoria.
Quattro sono i modi possibili di organizzare la memoria:

- in partizioni;
- in pagine;
- in segmenti;
- in segmenti paginati.

Ognuno di questi modi attua l'allocazione di un processo in memoria in base a due fattori principali:

1. **contiguità** o **non contiguità** nell'allocazione;
2. **staticità** o **dinamicità** nell'allocazione.

L'allocazione non contigua e dinamica associata alla possibilità di allocare **solo le aree di memoria di cui un processo necessita** in un dato istante (e non tutte le aree di memoria associate al processo) ha tecnicamente consentito di realizzare la memoria virtuale.

La possibilità di allocare solo le aree di memoria di cui un processo necessita non è fattibile con le tecniche di partizionamento. Occorre utilizzare la paginazione (dinamica), la segmentazione o la segmentazione paginata.

#prendinota

La contiguità implica che un processo debba essere allocato in aree consecutive di memoria fisica mentre la non contiguità permette di allocare un processo in aree separate.

#prendinota

La staticità implica che un processo, una volta allocato, non possa essere allocato altrove, mentre la dinamicità consente di scaricare (swap-out) e ricaricare (swap-in) il processo più volte in aree di memoria diverse.

TABELLA 1 Caratteristiche delle tecniche di allocazione della memoria

La **TABELLA 1** riassume le principali caratteristiche delle tecniche di allocazione della memoria. Si può notare come la virtualizzazione della memoria si ha solo con allocazione non contigua, dinamica e senza obbligo di avere l'intero processo in memoria centrale.

Tecniche di allocazione della memoria	Tabella principale usata	Altre tabelle utizzate			Allocazione	Intero processo allocato in memoria	Memoria virtuale	Particolarità da gestire
Partizioni statiche	Tabella delle partizioni				Contigua e statica	SÌ	NO	Frammentazione interna non recuperabile
Partizioni dinamiche	Tabella delle partizioni	Tabella delle aree libere			Contigua e dinamica	SÌ	NO	Frammentazione esterna recuperabile
Paginazione statica	Tabella delle pagine	Tabella dei blocchi			Non contigua e statica	SÌ	NO	Area interna alla pagina inutilizzata
Paginazione dinamica	Tabella delle pagine	Tabella dei blocchi	Tabella di mappa		Non contigua e dinamica	NO	SÌ	Page fault e thrashing
Segmentazione	Tabella dei segmenti				Non contigua e dinamica	NO	SÌ	Segment fault
Segmentazione paginata	Tabella dei segmenti	Tabella delle pagine	Tabella dei blocchi	Tabella di mappa	Non contigua e dinamica	NO	SÌ	Segment fault e page fault

L'ultima colonna della Tabella 1 mostra quali sono le problematiche che il sistema operativo si trova a dover gestire a seconda della tecnica utilizzata.
Le tecniche che non consentono la virtualizzazione non riescono a ottimizzare l'occupazione della memoria centrale, cioè creano degli sprechi dovuti alla frammentazione interna o esterna (quest'ultima recuperabile con onerose **#tecniche di compattamento**) o alla dimensione prefissata delle pagine che può non coincidere perfettamente con le dimensioni del processo.
Viceversa, le tecniche che consentono la virtualizzazione, non allocando tutto il processo in memoria centrale, sono soggette al rischio di non trovare allocata la pagina o il segmento cercato (**page fault** e **segment fault**). Inoltre, occorre fare attenzione che il sistema non sia impegnato in continui swap-in e swap-out di blocchi di memoria, rallentando fino a bloccare l'elaborazione dei processi in corso, fenomeno noto come **thrashing**. Per prevenire il thrashing si utilizzano **tecniche di rimpiazzamento** capaci di scegliere i blocchi (pagine o segmenti) da trasferire da memoria centrale a secondaria o viceversa, in base alle previsioni di utilizzo o meno di tali blocchi nelle successive elaborazioni.

#techwords

Tecniche di compattamento
Le tecniche di compattamento recuperano i piccoli frammenti di memoria rimasti liberi e li riuniscono in un'unica area più grande e quindi più facilmente utilizzabile. La compattazione si può applicare solo alle aree **esterne** alle partizioni o ai segmenti in cui è suddivisa la memoria. L'operazione è onerosa poiché impegna la CPU e la memoria, bloccando temporaneamente l'esecuzione dei processi.

FISSA E APPLICA LE TUE CONOSCENZE

1. Che cos'è lo spazio degli indirizzi?
2. Qual è la differenza tra rilocazione statica e dinamica?
3. Qual è la differenza tra allocazione statica e dinamica?
4. Quali sono i vantaggi della virtualizzazione della memoria?
5. Quali sono i quattro modi possibili di organizzare la suddivisione della memoria?
6. Quali caratteristiche sono indispensabili per poter applicare tecniche di virtualizzazione della memoria?
7. A che cosa servono le tecniche di compattamento?
8. A che cosa servono le tecniche di rimpiazzamento?

3 L'INTERAZIONE CON IL SISTEMA OPERATIVO

3.1 Le interfacce utente

Nell'Unità 6 del primo volume abbiamo introdotto la **shell** del sistema operativo: l'interfaccia tramite la quale l'utente può operare sul sistema.

> La shell è un programma che gestisce la comunicazione fra l'utente e il sistema operativo, interpretando ed eseguendo i comandi dell'utente (viene chiamata anche **interprete dei comandi**).

Le shell dei sistemi operativi possono avere caratteristiche molto diverse, anche se il loro compito rimane sempre quello di identificare la richiesta effettuata dall'utente nonché gli eventuali parametri a essa associati.
L'interazione con il sistema operativo avviene solitamente con le shell a interfaccia grafica e a linea di comando. Le shell a menu sono ancora utilizzate nei firmware come BIOS e UEFI.

Graphical User Interface (GUI)

È l'interfaccia che usiamo sullo smartphone, sul tablet e sul computer. La GUI è una metafora del desktop, cioè una scrivania su cui andiamo a svolgere tutti i lavori e che ci permette di interagire con la macchina manipolando una serie di **oggetti grafici** convenzionali, come le icone per rappresentare i file e le finestre per rappresentare le applicazioni. Dietro la GUI c'è un software sviluppato da un programmatore per rendere più accattivante, semplice e intuitiva l'interazione con il sistema (FIGURA 3).

FIGURA 3 L'interfaccia grafica di Windows 11

Command Line Interface (CLI)

La CLI utilizza una linea di comando per poter introdurre comandi veri e propri (FIGURA 4). In questo caso l'interfaccia fra il sistema operativo e l'utente è costituita da una serie di **comandi digitati su un terminale** e messaggi che il sistema operativo invia al terminale. I comandi possono essere immessi in modo interattivo o letti da file, cioè in modalità batch. Essi vengono letti e interpretati da un **interprete dei comandi**. Sicuramente con questa interfaccia il sistema risulta meno accattivante, ma la CLI permette un maggiore controllo sull'esecuzione del programma. Inoltre, privilegiamo la riga di comando quando si tratta di attività di amministrazione del sistema e risoluzione di problemi, in quanto fornisce un chiaro quadro di ciò che il sistema sta facendo in un dato istante.

```
sysadmin@localhost:~$ ls -l
total 32
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Desktop
drwx------ 4 sysadmin sysadmin 4096 Dec 20  2017 Documents
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Downloads
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Music
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Pictures
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Public
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Templates
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Videos
```

FIGURA 4 Il terminale di Linux

Confronto tra GUI e CLI

Gli utenti non molto esperti possono sentirsi a disagio con un'interfaccia a riga di comando e rassicurati da un'interfaccia grafica, ma i vantaggi e gli svantaggi sono presenti in entrambe, come mostrato nella TABELLA 2.

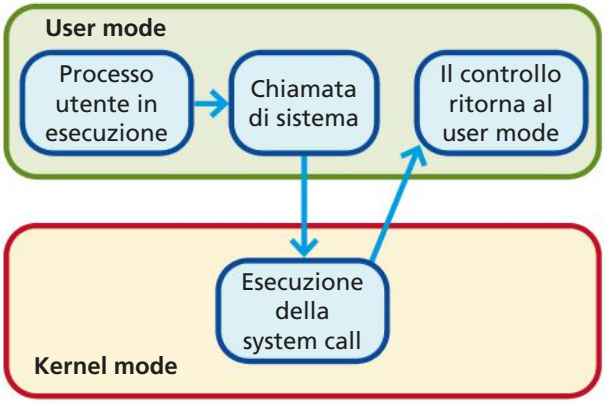
TABELLA 2 Confronto tra GUI e CLI

Caratteristica	GUI	CLI
Facilità d'uso	È molto facile da usare.	Richiede la conoscenza dei comandi e della loro sintassi.
Controllo	Potrebbe non essere sufficiente per svolgere compiti molto specifici.	L'utente ha un controllo più completo del file system e del sistema operativo.
Multitasking	La visualizzazione di più finestre in contemporanea permette all'utente di controllare e manipolare più oggetti e di farlo velocemente.	Molti ambienti a riga di comando sono capaci di multitasking, non permettono però di vedere più oggetti contemporaneamente su uno schermo.
Velocità	L'utente che interagisce con il sistema operativo utilizzando un mouse e una tastiera può essere più lento di chi lavora in una riga di comando.	L'utente CLI esperto lavora velocemente con la sola tastiera e spesso con poche righe di comandi svolge un compito avanzato.
Impiego di risorse	Richiede più risorse di sistema per ciascuno degli elementi che devono essere caricati come icone, font, ecc.	Richiede meno risorse di sistema.
Scripting	Consente all'utente di creare collegamenti, avviare l'esecuzione di un'applicazione, ecc. ma queste operazioni sono svolte una per volta.	L'utente può scrivere una sequenza di comandi per eseguire un compito, avviare l'esecuzione di un'applicazione secondo certe modalità, il tutto in modo programmato.

3.2 Le chiamate di sistema (System Call)

La comunicazione con il sistema operativo può avvenire anche tramite delle chiamate di sistema che fungono da interfaccia tra i programmi in esecuzione e il sistema operativo.

FIGURA 5 Il funzionamento di una chiamata di sistema



Esse sono solitamente disponibili come speciali istruzioni assembler o come delle funzioni nei linguaggi che supportano direttamente la programmazione di sistema (per esempio, il linguaggio C). Di queste istruzioni esistono vari tipi, da quelle relative alla creazione e al controllo di processi, a quelle per la gestione dei file e dei dispositivi, a quelle per il reperimento di informazioni. L'invocazione di una chiamata di sistema serve a ottenere un servizio dal sistema operativo, passando dal user mode al kernel mode, per ragioni di sicurezza. Una volta terminato il compito relativo alla particolare chiamata di sistema invocata, il controllo ritornerà al processo chiamante passando dal kernel mode al user mode (FIGURA 5).

FISSA E APPLICA LE TUE CONOSCENZE

1. **Secondo quali modalità si può interagire con il sistema operativo?**
2. **Descrivi le principali caratteristiche delle interfacce grafiche.**
3. **Descrivi le principali caratteristiche delle interfacce a riga di comando.**
4. **Che cosa sono le chiamate di sistema?**
5. **Le chiamate di sistema sono eseguite in user mode o kernel mode?**

4 IL SISTEMA OPERATIVO LINUX

4.1 "Linux is everywhere!"

"Linux è ovunque" è la frase che meglio di altre rende la potenza e la diffusione attuale di Linux. Infatti, il kernel Linux è presente in tantissimi sistemi: smartphone (Android) e smartwatch, console di videogiochi, computer e supercomputer, router, sistemi di controllo del traffico aereo, televisori, automobili. Come sistema operativo per computer desktop Linux è poco diffuso tra gli utenti, al contrario la percentuale sale molto se consideriamo i server, gli apparati di rete, i dispositivi Internet of Things (IoT) e molti altri device (FIGURA 6).

FIGURA 6 Dispositivi in cui è presente il kernel Linux

Guarda il **video**
Linus Torvalds

Linux è nato nel 1991 come un progetto personale di **Linus Torvalds**, studente all'Università di Helsinki in Finlandia, a partire dal suo studio della multiprogrammazione dei microprocessori i386 e del sistema operativo Unix (lavorò su Minix). Torvalds sviluppò il **kernel Linux**, più semplice di Unix, ma con le stesse importanti funzioni multitasking e multiuser.

#prendinota

Unix è il sistema operativo multitasking e multiuser sviluppato presso AT&T Bell Labs negli anni Settanta. Il codice era modulare e scritto quasi interamente nel linguaggio C così da renderlo facilmente portabile su varie piattaforme hardware (una vera novità per quegli anni).

Il progetto GNU

Nel 1983 **Richard Stallman** creò il progetto **GNU** (è un acronimo ricorsivo: **GNU's Not Unix**) allo scopo di sviluppare un sistema operativo compatibile con Unix ma accessibile a tutti liberamente. Al progetto collaborano molti sviluppatori e i pacchetti software realizzati includono applicazioni orientate all'utente, programmi di utilità, strumenti, librerie e videogiochi.

La filosofia del **software libero** predicata da Stallman prevede che il software debba rispettare la libertà degli utenti e la comunità: è una questione di libertà, non di prezzo. Infatti, potremmo aver pagato per una copia di un programma libero o averla ottenuta gratuitamente, a prescindere da ciò resta la libertà di copiare e modificare il software e anche di commercializzarlo.

Guarda il **video**
Richard Stallman

Consulta i documenti
Differenza tra open source e software libero

Le quattro libertà fondamentali

Libertà 0. Libertà di eseguire il programma come si desidera.

Libertà 1. Libertà di studiare come funziona il programma e di modificarlo in modo da adattarlo alle proprie necessità. L'accesso al codice sorgente ne è un prerequisito.

Libertà 2. Libertà di ridistribuire copie in modo da aiutare gli altri.

Libertà 3. Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti apportati in modo tale che tutta la comunità ne tragga beneficio. L'accesso al codice sorgente ne è un prerequisito.

GNU/Linux

Gli sviluppatori di GNU realizzarono la maggior parte dei programmi tipici di un sistema operativo (compilatori, editor di testo, shell a linea di comando e interfaccia grafica). Lo sviluppo delle componenti di basso livello come il kernel, i driver dei dispositivi, ecc. fu più complesso e richiese parecchi anni. Nel frattempo, uscì il kernel Linux sviluppato da Torvalds che scelse di distribuirlo con la licenza libera GNU. Divenne quindi possibile **includere Linux in GNU**, rendendolo un sistema operativo completo.
La maggior parte dei sistemi che comunemente chiamiamo Linux in realtà dovrebbe essere riferita con il termine GNU/Linux. Fa eccezione Android, che usa il kernel Linux ma non GNU, sostituito con librerie e strumenti proprietari.

Nel seguito faremo riferimento al sistema operativo con il termine Linux, come comunemente usato al posto del più corretto GNU/Linux.

Le distribuzioni

#techwords

Distro
È il termine con cui spesso ci riferiamo a un sistema operativo che include il kernel Linux.

Tutte le distribuzioni (**#distro**) condividono il kernel Linux, anche se in versioni diverse, ma si differenziano tra loro per i pacchetti preparati dai singoli sviluppatori, per il sistema di gestione del software e per i servizi di assistenza e manutenzione.

> Una tipica distribuzione di Linux include un kernel Linux, le librerie e il software GNU, un sistema per la gestione delle finestre, un ambiente desktop e un sistema di gestione dei pacchetti per l'installazione, aggiornamento e configurazione del software.

Il sito web DistroWatch (*https://distrowatch.com/*) offre un elenco aggiornato di distribuzioni di Linux. Molte di queste possono essere ricondotte a tre attori principali:

- **Debian**: è una delle prime distribuzioni GNU/Linux e offre un ottimo sistema di gestione dei pacchetti software; da questa sono derivate altre distribuzioni molto diffuse:
  - **Knoppix**: è stata la prima distribuzione avviabile direttamente da CD (Live CD) senza installazione;
  - **Linux Mint Debian Edition**: utilizza i pacchetti software di Debian, a differenza della versione Linux Mint derivata da Ubuntu;
  - **Ubuntu**: è la versione Linux più diffusa grazie anche alla sua interfaccia semplice e intuitiva; utilizza l'ambiente desktop GNOME.

- **Red Hat**: inizialmente era una distribuzione piuttosto semplice che si caratterizzava per il software Packet Manager, in seguito si focalizzò più sulle applicazioni server per le aziende con la versione Red Hat Enterprise Linux (RHEL). Da Red Hat sono nati due progetti:
  - **Fedora**: è una versione per desktop con software che viene aggiornato più frequentemente di RHEL, ma resta costruito sulle stesse basi della versione enterprise;
  - **CentOS**: costruito con i sorgenti di Red Hat, integra nuovo software seguendo la filosofia del software libero.
- **Slackware**: creata nel 1993, è solitamente vista come la distribuzione più "pura" dal punto di vista degli standard GNU e Linux; infatti resta quella più Unix-like e facilmente modificabile. Da Slackware è derivata:
  - **SUSE**: è una distribuzione molto nota in Europa; la versione **SUSE Linux Enterprise** è usata prevalentemente a livello aziendale e contiene codice proprietario, mentre la versione **openSUSE** è un software libero, usato anche da utenti desktop.

La gestione dei package

Ogni sistema Linux necessita spesso di aggiungere, eliminare e aggiornare il software. In passato queste attività erano piuttosto complesse e dispendiose di tempo, ma ormai tutte le distribuzioni utilizzano i #package: file compressi che contengono tutto il software necessario per semplificare l'installazione dell'applicazione. Infatti, vengono anche create le directory corrette copiandovi i file e creando tutto quanto è necessario per il funzionamento dell'applicazione.
I due software più utilizzati per la gestione dei package sono quelli di Debian e Red Hat, che forniscono un **package management system** avanzato.

#techwords

Package
(pacchetto in italiano)
È un file compresso che raggruppa un'applicazione e i file che necessita (ossia le sue dipendenze) per la sua installazione.

4.2 Le principali caratteristiche di Linux

Alcune delle principali caratteristiche di Linux sono le seguenti:

- **multitasking**: possibilità di eseguire più programmi contemporaneamente;
- **multiuser**: più utenti nella stessa macchina contemporaneamente;
- **multipiattaforma**: Linux è praticamente compatibile con tutti i comuni hardware di PC e può girare anche su processori non Intel;
- prevede funzioni di **protezione della memoria**, in modo che un programma non possa mandare in crash l'intero sistema;
- prevede la gestione della **memoria virtuale** attraverso la paginazione;
- **demand loads executables**: Linux legge dal disco solo le parti di un programma che sono attualmente usate;
- rende disponibili **librerie** statiche e dinamiche;
- prevede una **memoria unificata** per programmi utente e cache del disco, in modo che tutta la memoria possa essere usata per la cache e la cache possa essere ridotta quando funzionano grandi programmi;
- utilizza un proprio tipo di **file system**;
- supporta un'ampia gamma di **protocolli di rete** (TCP/IP, SLIP, PPP, ecc.);
- tutto il **codice sorgente** è disponibile, compreso il kernel e tutti i driver, gli strumenti di sviluppo e tutti i programmi utente;
- prevede **console virtuali multiple**: esse sono allocate dinamicamente e se ne possono usare fino a 64.

4.3 La shell di Linux

Nella precedente Lezione abbiamo definito la shell come un interprete di comandi che legge l'input, lo elabora e ne restituisce l'output. Il primo impatto con i comandi della shell può essere abbastanza impegnativo, essi sono numerosi e non sempre facili da memorizzare. Gli utenti potrebbero quindi essere motivati a interagire con il computer esclusivamente mediante la GUI. Ma le GUI sono quasi sempre meno potenti delle interfacce a riga di comando.
La maggior parte delle distro Linux offre un desktop che risulta familiare agli utenti dei sistemi Windows o MacOS, quindi all'inizio è usuale lavorare con questa GUI. Il passo successivo, però, è imparare a usare la CLI. L'interfaccia testuale di Linux è strettamente integrata con la GUI e, a differenza di Windows, spesso deve essere utilizzata per svolgere alcune operazioni. Man mano che impareremo a utilizzare la CLI di Linux, ci renderemo conto che molti comandi sono logici e intuitivi e che con il tempo diventerà tutto più facile. Molti utenti di Linux hanno una vera passione per la CLI preferendola all'interfaccia grafica.

#prendinota

Esistono applicazioni che permettono di emulare il terminale di Linux all'interno di un web browser, utili per provare i comandi senza dover installare una distribuzione Linux.
I due esempi più diffusi sono: **JSLinux** (*bellard.org/jslinux*) e **copy.sh** (*copy.sh/v86*).

L'ambiente testuale è realizzato da un'applicazione chiamata **terminale**. Il terminale prende quello che l'utente digita da tastiera e lo passa alla shell che interpreta il comando e lo esegue. Se viene prodotto un output, il testo è visualizzato sul terminale.
Esistono due modalità per accedere all'ambiente testuale:

1. l'applicazione **Terminal**: un programma che emula una finestra del terminale aprendola nella GUI;
2. il **terminale virtuale**: un programma che può essere eseguito insieme alla GUI ma richiede che l'utente effettui il login sul terminale virtuale prima di poter digitare i comandi. Di solito si utilizza il terminale virtuale quando si deve lavorare sui driver per la grafica oppure la GUI è bloccata e vogliamo terminare un dato processo che potrebbe essere la causa del problema.

Esercizio

➜ PROBLEMA

Su un computer con installato una distro di Linux avviare la CLI e digitare i comandi corretti per conoscere in quale directory ci troviamo e il suo contenuto.

➜ SVOLGIMENTO

Per svolgere l'esercizio abbiamo bisogno di un computer con installata una delle distribuzioni Linux che abbiamo visto in precedenza. Si può anche optare per un'installazione **live USB** utilizzando una pen-drive o un disco fisso esterno oppure scegliere di creare una **virtual machine**.
Nel nostro caso abbiamo fatto le seguenti scelte:

- distribuzione: **Ubuntu versione 22.04 LTS** (Long Term Support, è una versione stabile che offre supporto e aggiornamenti per 5 anni);
- installazione: creiamo una macchina virtuale con l'applicazione **Oracle VM VirtualBox** e installiamo il sistema operativo Ubuntu selezionando la configurazione minimale che prevede il browser e pochi strumenti di base.

Per entrare nell'ambiente testuale, l'interfaccia CLI di Ubuntu, possiamo utilizzare da GUI l'applicazione Terminal oppure attivare un terminale virtuale.

TERMINAL

Per avviare l'applicazione **Terminal** possiamo scegliere tra quattro modalità diverse:

1. digitare sulla tastiera la combinazione dei tre tasti **Ctrl + Alt + t**;
2. cliccare in un punto vuoto del desktop con il tasto destro del mouse e selezionare **Open in Terminal**;
3. visualizzare le applicazioni installate cliccando sull'icona **Show Application** in basso a sinistra e selezionare **Terminal**;
4. aprire la finestra **Run a command** (Alt + F2) e digitare **gnome-terminal**.

Nella finestra di Terminal scriviamo i due comandi richiesti:

- **`pwd`** (print working directory) per visualizzare la directory in cui ci troviamo;
- **`ls`** (list) per avere l'elenco dei file e delle directory contenuti nella directory in cui ci troviamo.

La FIGURA 7 mostra il risultato dei due comandi: la directory è quella dell'utente **`eb`** e si trova sotto la directory **`home`** dove sono memorizzate le directory degli utenti. L'output del comando **`ls`** elenca il contenuto della directory eb: sono presenti nove directory.
La CLI ci fornisce anche alcune informazioni tramite il **colore del testo**, per esempio le directory sono di colore **blu**, i file di colore **bianco**, i link che rappresentano una scorciatoia per raggiungere una directory o file sono di colore **turchese**, ecc. Nella Lezione 6 vedremo come è possibile personalizzare questi colori.

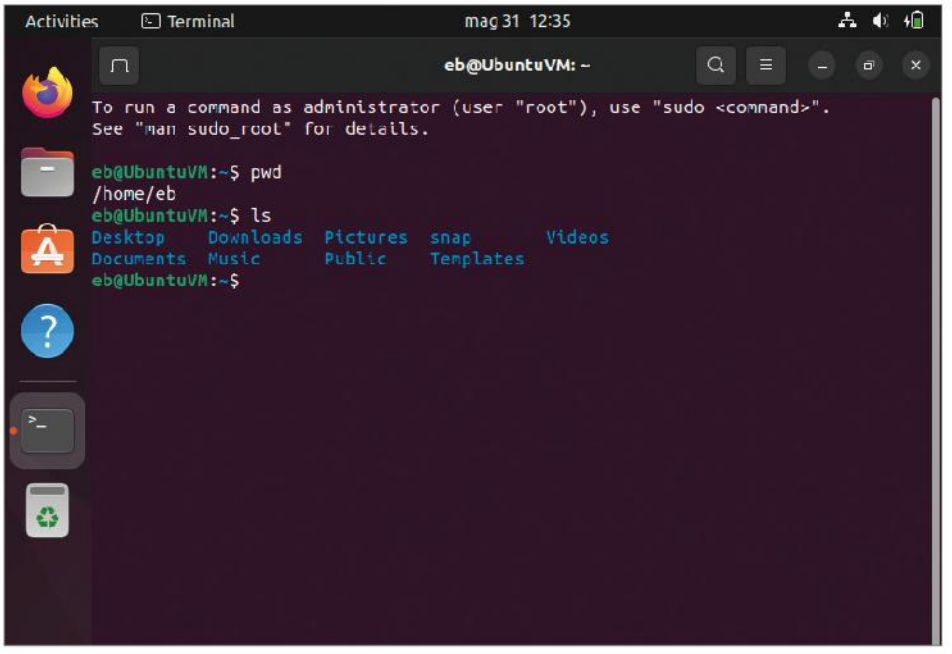


FIGURA 7 L'applicazione Terminal di Ubuntu

Il formato del prompt è:

```
nomeutente@nomecomputer:/path$
```

dove con `path` si intende il percorso a partire dalla radice `/` fino alla directory in cui ci troviamo.
Nella Figura 7 non è presente il path `/home/eb` nel prompt, in quanto per convenzione la directory home dell'utente è indicata con il simbolo tilde.

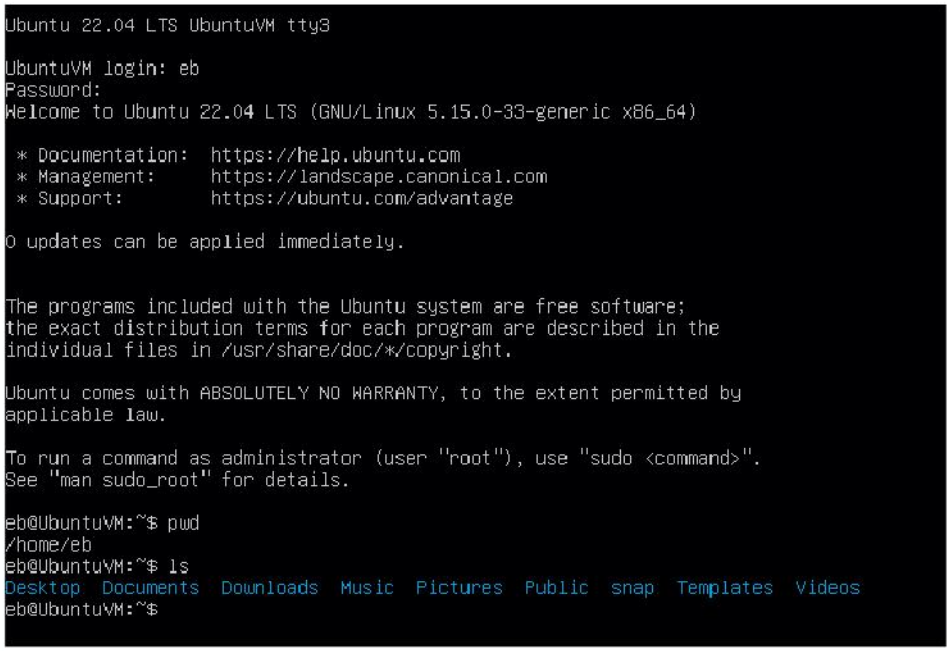
TERMINALE VIRTUALE

In alternativa al programma Terminal si può attivare il **terminale virtuale** (a volte è chiamato **console**):

1. digitare sulla tastiera la combinazione dei tre tasti **Ctrl + Alt + F3**;
2. effettuare il **login** con username e password; notiamo che per la password non sono visualizzati i soliti pallini o asterischi al posto dei caratteri, così non si forniscono informazioni sulla lunghezza della password;
3. per chiudere il terminale virtuale e tornare alla GUI digitare **Ctrl + Alt + F2** (a volte si deve sostituire F2 con F4 o altri tasti funzione).

La FIGURA 8 mostra la prima schermata che si ottiene dopo aver inserito username e password. Nel terminale virtuale abbiamo poi digitato i due comandi **`pwd`** e **`ls`** come in Terminal.

FIGURA 8 Il terminale virtuale di Ubuntu



Tipologie di shell

In Linux sono disponibili diverse shell; nella TABELLA 3 elenchiamo le più diffuse.

TABELLA 3 Tipi di shell

Shell	Comando	Sviluppatore	Descrizione
Bourne shell	sh	Stephen Bourne (Steve)	È stata la prima shell sviluppata.
C shell	csh	William Joy (Bill)	Deriva da sh ma ha una sintassi simile a quella del linguaggio C.
Korn shell	ksh	David Korn	È stata sviluppata presso AT&T Bell Lab; ha proprietà derivate da sh e csh.
Bourne again shell	bash	Chet Ramery	Versione evoluta di sh, è la shell di default di Linux.

La **shell bash** è quella più utilizzata in quanto è la più completa e presenta, in un unico pacchetto, tutte le caratteristiche positive delle altre. Ogni shell svolge lo stesso lavoro, ma ognuna ha una diversa sintassi dei comandi e fornisce diverse funzioni.

I comandi base della shell

I comandi che si impartiscono alla shell possono essere considerati come composti da tre parti: **nome del comando**; **opzioni** (precedute sempre da un trattino); uno o più **argomenti**. Questa suddivisione è molto utile per capire come funzionano i comandi: cambiando un'opzione o l'argomento uno stesso comando può svolgere compiti molto differenti.

I comandi possono essere scritti anche su più righe, digitando il carattere `\` prima dell'invio, e per indicare l'attesa del completamento di un comando viene visualizzato un prompt diverso.
Inoltre si possono scrivere più comandi sulla stessa riga separandoli con un `;` .

Per avere informazioni sui comandi possiamo utilizzare il comando **`man`**, il quale accederà a tutti i comandi presenti sul manuale. Per ogni comando si avrà il formato, le opzioni, i comandi correlati, esempi, e così via. Per uscire dal manuale si deve digitare **`q`**.

La shell fornisce la possibilità di **editing** della riga di comando. Per esempio, premendo **`Tab`**, dopo aver scritto poche lettere si può avere il **completamento automatico** del comando.
I comandi dati durante una sessione del terminale sono numerosi e capita di doverne ripetere alcuni più volte o di eseguirli con una piccola variazione. Con il comando **`history`** è possibile vederli tutti (FIGURA 9). Infatti i comandi che abbiamo dato in precedenza sono salvati nel file `.bash _ history` nella home directory e trattati come eventi a cui è associato un numero con il quale richiamare il comando. Utilizzando la freccia in su si scorre a ritroso nella lista dei comandi digitati così da poterli ripetere o modificare.

```
22  clear
23  echo $PATH
24  nano diskquota.sh
25  man fquota
26  man du
27  ls
28  nano diskquota.sh
29  nano diskquota.sh.save
30  man mv
```

FIGURA 9 Il comando history

È possibile eliminare tutta la cronologia usando l'opzione `-c`:

```
history -c
```

Esempio

Facendo riferimento alla Figura 9, ripetiamo il comando per aprire il file diskquota.sh:

```
!24
```

Per cercare tutti i comandi dati in precedenza che hanno riguardato il file diskquota.sh scriviamo:

```
history | grep diskquota.sh
```

I comandi della shell possono essere gestiti direttamente dal processo shell (siamo nel caso di comandi **interni**) oppure richiedono l'esecuzione di un file eseguibile (comandi **esterni**).

#prendinota

history in realtà non è un comando vero e proprio, se digitiamo `which history` il sistema non lo trova. Infatti `history` è una keyword usata per riferirsi alla libreria della shell che offre alcune funzioni per la gestione della cronologia dei comandi digitati dall'utente.

#techword

Alias
Dà all'utente la possibilità di rinominare un comando.

Gli **#alias** servono per eseguire dei comandi complessi utilizzando stringhe semplificate. Come dice il nome, gli alias vengono definiti sulla base di comandi preesistenti e ne sostituiscono solo il nome (o l'intera stringa).
Per creare un alias per un comando si usa:

```
alias nuovonome = comando
```

Al riavvio del computer l'alias verrà però cancellato. Per evitare che questo avvenga occorre editare un file nascosto di sistema che si trova nella cartella dell'utente ed è chiamato **.bashrc**: il comando digitato in precedenza dal terminale dovrà essere riscritto su tale file. Se vogliamo rimuovere l'alias basta invece utilizzare il comando **unalias**.
Ricordiamo che molti comandi per essere eseguiti necessitano dei privilegi di amministratore (utente root) e quindi dobbiamo effettuare il login come root. Se stiamo usando Ubuntu è sufficiente anteporre ai comandi **sudo**.

Senza entrare nel dettaglio vediamo ora alcuni semplici esempi di comandi:

- **date**: è un comando senza argomenti che ci restituisce la data di sistema; si può specificare il formato;
- **passwd**: è usato per cambiare la propria password digitando quella vecchia; i caratteri digitati non vengono visualizzati. Naturalmente il superuser può cambiare qualsiasi password;
- **su**: è un comando che, senza argomenti, permette di accedere all'utente root; seguito dal nome dell'utente permette di aprire temporaneamente un'altra sessione;
- **login**: è un comando che, seguito dal nome dell'utente, permette di aprire un'altra sessione terminando quella attuale;
- **echo**: è un comando che manda in output sullo schermo la stringa che lo segue;
- **ls**: visualizza la lista dei file contenuti nella cartella in cui ci si trova;
- **man ls**: visualizza il manuale del comando ls.

Nella Lezione che segue studieremo altri utili comandi per la gestione dei file e dell'input/output.

FISSA E APPLICA LE CONOSCENZE

1. **Che cosa significa l'affermazione "Linux is everywhere"?**
2. **Descrivi che cosa si intende con software libero e quali sono le libertà fondamentali elencate da Stallman.**
3. **Quali sono le tre distribuzioni iniziali di Linux dalle quali ne sono nate altre?**
4. **Che cosa si intende per shell? Fai qualche esempio di shell.**
5. **Che differenza c'è tra l'applicazione Terminal e il terminale virtuale?**
6. **Apri il file `.bashrc` nella tua home directory e leggi i valori di default relativi alla dimensione del buffer (HISTSIZE) e alla dimensione del file `.bash_history` (HISTFILESIZE). Modifica questi dati stabilendo di memorizzare 500 eventi nel buffer e 1000 eventi nel file. Ricorda che per scrivere nel file `.bashrc` devi avere i privilegi di `root`.**

5 LA GESTIONE DEI FILE E I/O IN LINUX

5.1 Il file system di Linux

Nel volume per il terzo anno abbiamo approfondito come il sistema operativo gestisce le informazioni memorizzate nella memoria secondaria. I moduli del file management system (abbreviato in **file system**) costituiscono il ring 4 del modello a ring del sistema operativo. Appartengono a questo ring i moduli che controllano il flusso dei segnali e dei dati da e verso le memorie secondarie (HDD, SSD, CD, DVD, USB, ...).

> L'insieme dei programmi del sistema operativo preposto alla gestione delle informazioni memorizzate sulle memorie secondarie costituisce il **file system**.

Un sistema Linux non fa distinzioni tra un file e una directory, visto che una directory è un file che contiene le informazioni di altri file. Programmi, servizi, testi, immagini, ecc. sono tutti file; anche i dispositivi periferici come una stampante sono trattati come file.
Questo concetto viene riassunto nella famosa frase inglese:

in Linux everything is a file!

Nei sistemi Windows la struttura gerarchica delle directory (chiamate folders, o cartelle in italiano) prevede a livello più alto la cartella Questo PC e sotto i dispositivi fisici contrassegnati da una lettera, come mostrato nella FIGURA 10.

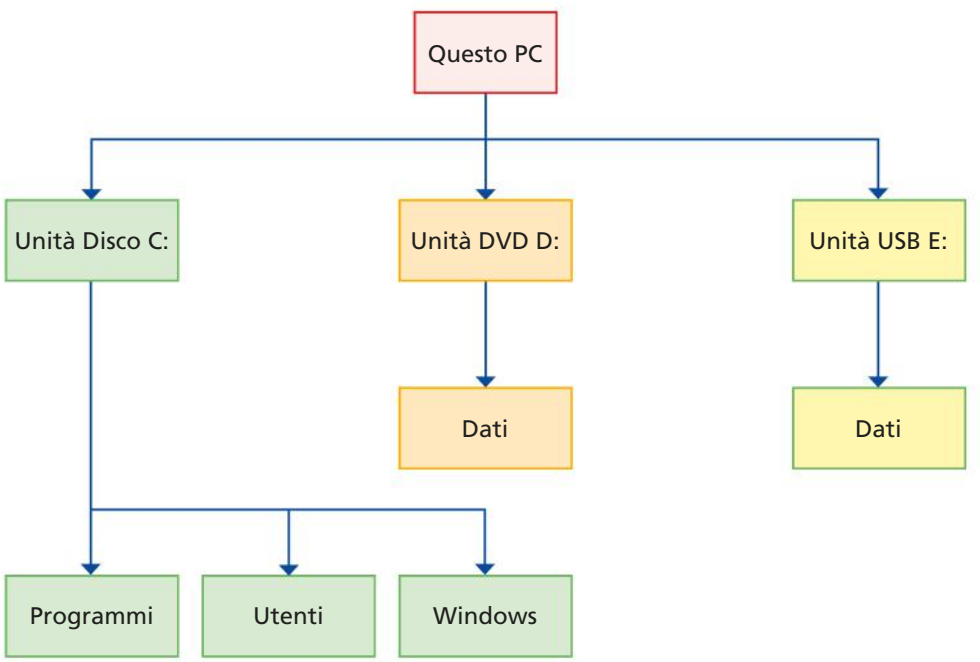
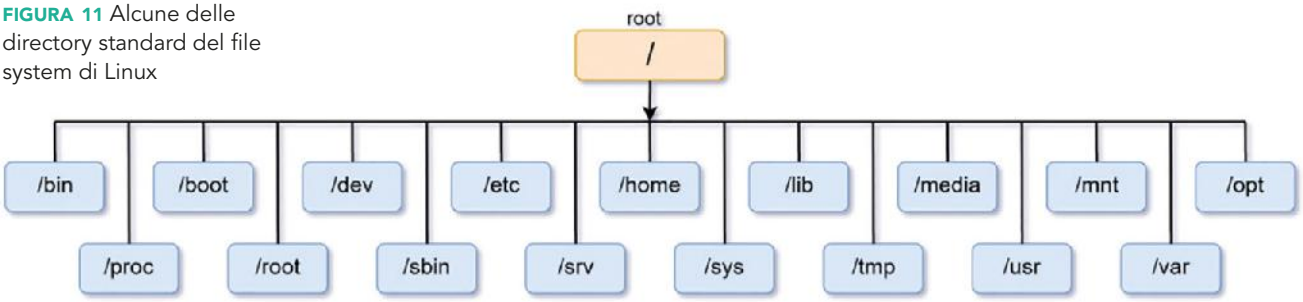


FIGURA 10 La gerarchia delle cartelle di Windows

A differenza di Windows, in Linux non ci sono unità (drive) con lettere che li etichettano; ogni dispositivo è accessibile attraverso una directory. Ogni sistema Linux ha una serie di directory che vengono usate in maniera standard. In ogni sistema troveremo queste directory, indipendentemente dalla distribuzione, e tutte avranno lo stesso scopo.

La FIGURA 11 mostra le directory che normalmente troviamo sotto la directory **root** e in TABELLA 4 è descritto il contenuto di ciascuna di esse.

FIGURA 11 Alcune delle directory standard del file system di Linux



DIRECTORY	CONTENUTO
bin	File eseguibili (binary) per i comandi usati dagli utenti
boot	File usati dal bootloader per caricare il kernel Linux
dev	File che rappresentano i dispositivi collegati al sistema
etc	File di configurazione del sistema
home	Home directory di tutti gli utenti
lib	File di libreria per tutti i comandi memorizzati in /sbin e /bin necessari per l'avvio di Linux
media	Contiene i punti di mount per i supporti removibili (es. cd/dvd-rom)
mnt	Punto di mount per parti del file system temporanee
opt	Packages per installare grandi applicazioni (es. LibreOffice)
proc	Varie informazioni sui processi in esecuzione
root	Home directory dell'utente root
sbin	File eseguibili rappresentanti i comandi usati per svolgere compiti di amministrazione del sistema
srv	Dati per i servizi offerti dal sistema
sys	Informazioni sui dispositivi connessi al sistema
tmp	File temporanei di utenti e di sistema
usr	Sottodirectory per programmi importanti (es. X Window) e manuali
var	Dati di vario tipo (es. log, mail, ...)

TABELLA 4 Elenco di alcune delle directory contenute in root (/)

Per vedere sul terminale l'elenco delle directory descritte nella Tabella 4 si usa il comando **`ls`** seguito dal carattere / che indica appunto la directory root. Il risultato è mostrato nella FIGURA 12.

FIGURA 12 Il comando ls /

```
eb@UbuntuVM:~$ ls /
bin    dev   lib    libx32      mnt   root  snap        sys  var
boot   etc   lib32  lost+found  opt   run   srv         tmp
cdrom  home  lib64  media       proc  sbin  swapfile    usr
eb@UbuntuVM:~$
```

Per quanto riguarda la gestione dei file, si hanno le seguenti differenze fondamentali di Linux rispetto a Windows:

- il carattere di separazione delle directory è il carattere / anziché il carattere \;
- Linux è **case sensitive**, distingue tra lettere minuscole e maiuscole, per esempio se chiamiamo un file project questo sarà diverso da un file denominato Project presente nella stessa directory;

- il carattere **.** non ha un significato particolare, visto che il tipo di file non è indicato dai tre caratteri dopo il punto e addirittura un nome può contenere più punti o nessuno; per esempio, i file e le directory i cui nomi cominciano con un punto sono considerati file nascosti;
- l'**estensione dei file** è spesso utile per determinare il tipo di file, ma non ha significato vincolante per il sistema operativo.

Il partizionamento

Uno degli scopi principali di avere diverse partizioni è quello di raggiungere un livello di sicurezza dei dati maggiore nel caso si verifichino delle anomalie. Dividendo il disco in partizioni, i dati possono essere raggruppati e separati. Quando capita un guasto, solo i dati nella partizione colpita saranno danneggiati, mentre i dati nelle altre partizioni molto probabilmente rimarranno integri.
Un sistema Linux deve essere installato in una partizione del disco. È possibile scegliere tra diverse modalità di installazione:

- si usa un'**unica partizione** in cui installare l'intero sistema operativo, le applicazioni e i file utente;
- si crea un'**area di *swap*** su una partizione separata da quella di sistema, da usare come una memoria virtuale (nelle architetture a 32 bit la dimensione massima è 2 GB);
- si crea un'**area *home*** in cui memorizzare i file utente con i dati, in modo da tenerli separati dalla parte del file system che contiene i file del sistema operativo.

5.2 La struttura dati inode

In Linux possiamo distinguere tre tipi di file:

- **file normali**: sequenza continua di caratteri che terminano con un marcatore di fine file;
- **directory**: file che raggruppano altri file;
- **file speciali**: rappresentano i dispositivi di Input/Output.

> Tutte le informazioni su di un file sono memorizzate in un elemento chiamato **inode** (index node) di una tabella. In Linux il file non viene identificato dal suo nome ma dal **numero di inode** che gli è stato assegnato quando è stato creato e che è unico in tutto il file system.

Gli inode costituiscono il fondamento di un qualsiasi file system in Linux e sono archiviati su disco in una lista detta **inode list**. Come mostrato nella Figura 13 della pagina successiva, la **struttura di un inode** contiene:

- i **metadati**: tipo di file (per esempio directory, device, ecc.) e permessi (mode), id del proprietario (owner) e del gruppo, dimensione in byte (size), data dell'ultimo accesso, dell'ultima modifica dei dati del file e dell'ultimo cambiamento nell'inode (timestamp);
- alcuni **puntatori**; ne sono previsti 15:
  - 12 direct pointers che puntano direttamente ai blocchi dei dati del file;
  - 1 single indirect pointer che punta a un blocco di puntatori che puntano ai dati;
  - 1 double indirect pointer che punta a un blocco di puntatori che puntano ad altri blocchi di puntatori che puntano ai dati;
  - 1 triple indirect pointer che punta a un blocco di puntatori che puntano ad altri blocchi di puntatori che puntano ad altri blocchi di puntatori che infine puntano ai dati.

#prendinota

Un inode non memorizza il nome del file né il suo contenuto.

La FIGURA 13 mostra un esempio di inode e dei relativi blocchi puntati.

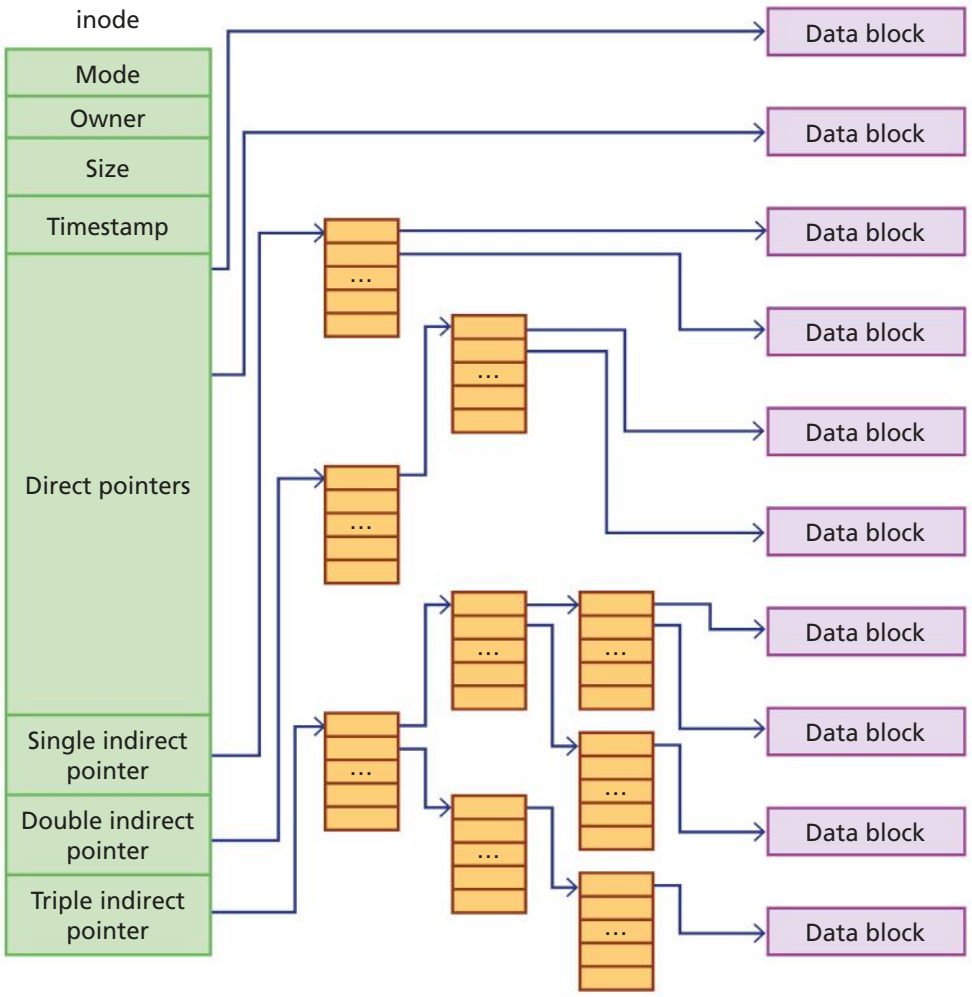


FIGURA 13 La struttura dati inode

#prendinota

L'efficienza della struttura inode risiede nel meccanismo di allocazione dei blocchi logici dei dati: hanno dimensione fissa, sono allocati in modo non contiguo e solo se necessario.

Esercizio

➜ PROBLEMA

Trovare il numero di inode che identifica i file contenuti nella directory Documents che troviamo nella directory Home.

➜ SVOLGIMENTO

Il comando da usare da terminale è **`ls`** che fornisce l'elenco dei file presenti nella directory con l'opzione **`-i`** per conoscere il numero di inode del file, come mostrato nella figura seguente.

```
eb@UbuntuVM:~$ cd Documents
eb@UbuntuVM:~/Documents$ ls -i
1464432 prova.txt
```

La directory Documents contenuta in Home contiene il file prova.txt il cui numero di inode è 1464432. Questo inode quindi contiene i metadati per questo file e i puntatori ai data block che contengono i dati del file sull'hard drive.

Esercizio

➜ PROBLEMA

Trovare quanti inode ci sono nel file system.

➜ SVOLGIMENTO

Il comando da usare da terminale è **`df`** (disk filesystem) che fornisce indicazioni su quanto spazio del disco è utilizzato dal file system. L'opzione **`-i`** permette di avere informazioni sull'elenco degli inode, come mostrato in FIGURA 14.

```
eb@UbuntuVM:~$ df -i
Filesystem      Inodes  IUsed   IFree IUse% Mounted on
tmpfs           502524    907  501617    1% /run
/dev/sda3      1605632 190699 1414933   12% /
tmpfs           502524      1  502523    1% /dev/shm
tmpfs           502524      4  502520    1% /run/lock
/dev/sda2            0      0       0     - /boot/efi
tmpfs           100504    138  100366    1% /run/user/1000
eb@UbuntuVM:~$
```

FIGURA 14 Comando `df -i`

Analizziamo l'output ottenuto:

- Filesystem: elenca i file system, di questi ci interessa l'hard drive /dev/sda3 che monta il file system dalla root, mentre la partizione /dev/sda2 è quella creata per il boot da UEFI;
- Inodes: il numero totale di inode in ogni file system;
- IUsed: il numero di inode in uso;
- IFree: il numero di inode ancora disponibili;
- IUse%: la percentuale di inode utilizzati;
- Mounted on: il punto in cui è stato montato questo file system.

Nell'output di Figura 14 risulta che il sistema sta usando il 12% degli inode disponibili.

5.3 I percorsi

Per indicare la posizione di un file o di una directory all'interno di un sistema operativo si usano i percorsi (**#pathname**); questi possono essere sia assoluti sia relativi:

- i **pathname assoluti** utilizzano come riferimento la directory principale del sistema operativo, quindi la radice /;
- i **pathname relativi** si riferiscono alla posizione corrente dell'utente.

Alcuni caratteri hanno un significato speciale:

- `.` indica la directory corrente;
- `..` indicano la cartella che contiene quella nella quale si è, cioè quella superiore;
- `~` indica la directory `home` dell'utente;
- `/` indica la directory `root`, radice del file system.

#techwords

Pathname
È un percorso per identificare in un modo univoco un file o una directory all'interno del file system.

#prendinota

Il simbolo tilde **~** si ottiene dalla combinazione di tasti: **Alt+126**, inserendo il numero dal tastierino numerico (in assenza di questo si digita **Alt+Fn+126**).

5.4 Utenti e permessi

Linux è un sistema multiuser quindi deve essere garantita la sicurezza delle attività di tutti gli utenti e del sistema stesso. Dal punto di vista dell'accesso a file e directory gli utenti in Linux si distinguono in:

- **proprietario**: chi ha creato il file;
- **gruppo**: i membri che fanno parte del gruppo di appartenenza del file;
- **altri**: tutti gli utenti che non rientrano nelle categorie precedenti.

#prendinota

Per avviare un programma occorre avere permessi di esecuzione sul file che contiene il programma. Per accedere al contenuto di una directory occorre avere permessi di lettura ed esecuzione sulla directory.

I file e le directory hanno un proprietario, cioè l'utente che li ha creati, e un gruppo di appartenenza.
Per ciascun file o directory si definiscono le operazioni che può fare ogni utente:

- **r** = read (lettura);
- **w** = write (scrittura);
- **x** = execute (esecuzione).

Come mostrato nella TABELLA 5, i permessi di un utente sono espressi con 3 cifre che corrispondono, nell'ordine, alle operazioni **r w x**. Ogni cifra può assumere il valore 0 (permesso negato) o 1 (permesso accordato). Il numero risultante è scritto in base 8.

TABELLA 5 Permessi dei file

Numero ottale	Permesso	Descrizione
0	`---`	nessuna autorizzazione
1	`--x`	esecuzione
2	`-w-`	scrittura
3	`-wr`	scrittura e lettura
4	`r--`	lettura
5	`r-x`	lettura ed esecuzione
6	`rw-`	lettura e scrittura
7	`rwx`	lettura, scrittura ed esecuzione

Per vedere i permessi di un file o di una directory si usa il comando **`ls`** con l'opzione **-l**, descritto nel paragrafo successivo.

> Solo il proprietario del file e l'amministratore di sistema possono modificare i permessi di un file con il comando **`chmod`**.

5.5 Operazioni comuni sulle directory

Muoversi fra le directory

Il comando **`cd`** (change directory) permette di muoversi fra le directory e il suo utilizzo è molto semplice, per accedere a una directory contenuta nella cartella in cui siamo digiteremo:

```
cd Directory
```

Volendosi spostare direttamente in una sua sottodirectory (anche infiniti livelli di sottodirectory, non c'è un limite) basta scrivere:

```
cd Directory/Sottodirectory1/Sottodirectory2/
```

Visualizzare il contenuto di una directory

Il comando **ls** (list) serve per visualizzare il contenuto di una directory, cioè listare i file:

```
ls
```

restituirà il contenuto (directory, file e link) della directory in cui ci troviamo.
Ma `ls` ci può fornire molte più informazioni; associando al comando una o più delle sue opzioni è possibile avere molti più dettagli. Alcune tra le principali opzioni sono:

- `-l` nell'elenco sono presenti informazioni aggiuntive;
- `-a` richiede un elenco completo, compresi i file nascosti;
- `-t` ordina l'elenco in base alla data e ora dell'ultima modifica fatta;
- `-u` ordina l'elenco in base alla data e ora dell'ultimo accesso.

#prendinota

Per facilitare il ritorno alla directory home si scrive il comando **cd** senza argomenti. Così si evita di digitare il percorso completo o di inserire il carattere tilde.

Esempio

Eseguiamo il comando:

```
ls -l
```

Il risultato è la visualizzazione di tutti i file o directory, uno per riga, con il formato:

```
drwxr-xr-x 3 eb eb 4096 giu 24 18:37 Documents
```

Esaminiamo i vari campi partendo da sinistra:

1. nel primo campo il primo carattere **d** indica che `Documents` è una directory, nel caso fosse un file ci sarebbe stato un trattino, il carattere **l** indica invece che si tratta di un link; i successivi nove caratteri sono i permessi, nell'ordine, del **proprietario**, del **gruppo** e degli **altri** utenti. Sono specificati come `read(r)`, `write(w)` `executable(x)`. Nel nostro esempio il proprietario ha il controllo completo, mentre il gruppo e gli altri utenti hanno i permessi di lettura ed esecuzione;
2. il numero nel secondo campo indica il numero di link al file o directory;
3. il terzo e quarto campo (nell'esempio eb eb) indicano il proprietario e il gruppo. Ogni utente appartiene a un gruppo privato che ha lo stesso nome dello username; nell'esempio lo username compare due volte, come owner e come group;
4. i campi successivi visualizzano la dimensione in byte; la data e l'ora dell'ultima modifica effettuata; il nome.

Creare una directory

Il comando **mkdir** (make directory) serve per creare una directory:

```
mkdir Directory
```

Per poter creare la cartella bisogna avere il permesso di scrittura nella directory padre.

Cancellare una directory

Il comando **rmdir** (remove directory) cancella una directory e il suo contenuto:

```
rmdir Directory
```

Il comando **rm** (remove) serve per cancellare una directory, ma solo se è vuota:

```
rm Directory
```

È inoltre disponibile il comando **pwd** (print working directory) che visualizza il nome della directory corrente.

5.6 Operazioni comuni sui file

Prima di vedere quali sono i principali comandi di manipolazione dei file dobbiamo ricordare che per formare i nomi dei file si possono usare alcuni metacaratteri i quali sono interpretati in modo particolare dalla shell:

- **`*`** indica un gruppo di caratteri qualsiasi;
- **`?`** indica un carattere qualsiasi;
- **`[ ]`** racchiudono una lista di caratteri da considerare in alternativa.

Copiare un file

Il comando **`cp`** (copy) viene usato per copiare file (comprese le directory):

```
cp file destinazione
```

#prendinota

Bisogna fare attenzione al fatto che `cp` non avverte nel caso che esista già un file con quel nome. Per evitare problemi di sovrascrittura si può usare l'opzione **`-i`** che abilita la richiesta di conferma (viene posta una domanda) in caso che il file esista già.

Un'opzione molto utile è **`-a`**, che conserva nella copia la struttura e gli attributi dei file originali, mantenendo così gli stessi permessi durante un backup di sistema.

Spostare o rinominare un file

Il comando **`mv`** (move) serve per spostare i file, il suo uso più elementare è:

```
mv file destinazione
```

Se la destinazione non è una directory il file viene rinominato:

```
mv nome_file nuovo_nome_file
```

Le principali opzioni sono:

- **`-i`** chiede una conferma per la sovrascrittura nel caso in cui nella destinazione esista già un file (o directory) con quel nome;
- **`-v`** stampa il nome di ogni file prima di spostarlo.

Eliminare un file

Per eliminare un file usiamo il comando **`rm`** visto precedentemente per le directory:

```
rm file
```

È comunque consigliabile usare un'opzione di sicurezza come **`-i`** che chiede conferma per ogni file da cancellare.

Visualizzare il contenuto di un file

Per visualizzare il contenuto di un file possono essere usati tre comandi:

- **`cat`**, che ci mostra semplicemente il contenuto del file:

```
cat nomefile
```

- **`less`**, che visualizza un file di testo una pagina alla volta, consente di scorrere il testo usando le frecce e di uscire dalla lettura premendo il tasto q:

```
less nomefile
```

- **`more`**, simile al comando `less` ma usando le frecce il testo avanza o torna indietro di una pagina (quindi non è possibile scorrere una riga alla volta):

```
more nomefile
```

Stampare un file

Il comando **lpr** serve per stampare uno o più file:

```
lpr nomefile
```

Se si utilizza l'opzione **-pt lpnn** sarà possibile dirigere i file a una determinata stampante (nn), altrimenti viene utilizzata la stampante di default.

Cambiare i permessi ai file

Il comando **chmod** serve per cambiare i permessi:

```
chmod valore file
```

dove il valore dei permessi può essere espresso in due modi:
- con numero ottale;
- con caratteri (u, g, o, a, r, w, x).

Utilizzando il sistema ottale si avrà un numero composto da tre cifre, ognuna per esprimere rispettivamente:
- i permessi del proprietario (la prima cifra);
- i permessi del gruppo (la seconda);
- i permessi di tutti gli altri utenti del sistema (la terza).

Utilizzando i caratteri si seguiranno delle semplici regole:
- u identifica il proprietario (user);
- g identifica gli utenti dello stesso gruppo del proprietario (group);
- o identifica tutti gli altri utenti del sistema (others);
- a identifica tutti gli utenti del sistema (all).

Con:
- **–** si toglie un permesso;
- **+** si concede un permesso.

#prendinota

Permessi di accesso in ottale
Per ogni cifra si userà il valore (da 0 a 7) in base a queste regole:
- nessun permesso → valore 0;
- permesso lettura → valore 4;
- permesso scrittura → valore 2;
- permesso esecuzione → valore 1.

Per ogni soggetto basterà fare la somma dei valori dei permessi che vogliamo assegnargli; per esempio, se desideriamo dare il permesso di lettura e scrittura, la cifra sarà: 4 (lettura) + 2 (scrittura) = 6.

Esercizio

➜ PROBLEMA

Dato il file `myfile` renderlo accessibile in lettura e scrittura solo al proprietario.

➜ SVOLGIMENTO

Il modo più semplice per realizzare quanto richiesto è cancellare i permessi a tutti gli utenti e poi assegnare `r` e `w` al proprietario. Utilizziamo quindi due volte il comando chmod:

```
chmod a-rwx myfile

chmod u+rw myfile
```

Per verificare che i permessi siano stati impostati correttamente eseguire il comando:

```
ls -l myfile
```

Il risultato sarà: `-rw-------` .

Creare un link

In Linux esistono due tipologie di link:

- **soft link** (simbolico): è il collegamento che siamo soliti incontrare anche in ambiente Windows ed è quello che usiamo più di frequente, salvo specifiche necessità;
- **hard link**: è qualcosa di più complesso, che lega il link all'inode del file collegato.

Per creare un link simbolico dobbiamo usare il comando **`ln`** con l'opzione **`-s`**:

```
ln -s file nome_collegamento
```

5.7 Gestione dell'I/O: la ridirezione

Molti comandi Linux leggono l'input, che viene dato da tastiera, come file o altro attributo del comando e scrivono l'output, che appare sullo schermo. La tastiera costituisce il dispositivo di **standard input** (**`stdin`**) e lo schermo o la singola finestra di terminale è invece il dispositivo di **standard output** (**`stdout`**). Esiste inoltre lo **standard error** (**`stderr`**) che permette di visualizzare i messaggi di errore, normalmente sullo schermo.

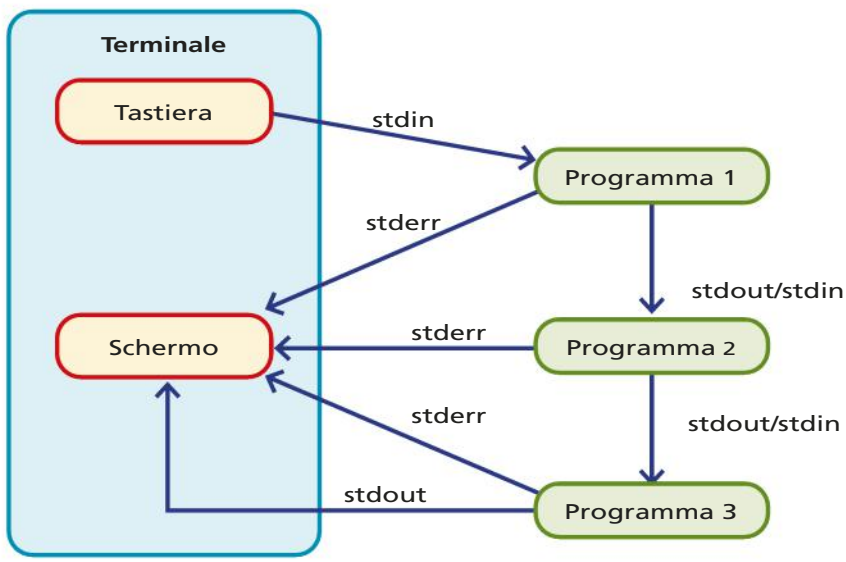


FIGURA 15 Input/Output in Linux

La **FIGURA 15** mostra l'interazione tra programmi e tra programma e tastiera e schermo. In Linux queste impostazioni di base non devono necessariamente essere applicate. Per esempio, lo standard output di un server molto usato in un grande ambiente di lavoro può essere una stampante. In generale si avrà la seguente situazione:

- se non viene specificato alcun file di input, il programma prende il suo input da `stdin`;
- dirige la sua uscita normalmente a `stdout`;
- dirige qualsiasi messaggio di errore a `stderr`.

Ogni flusso tra programma e file viene individuato, all'interno di un processo, da un **descrittore di file**, cioè da un numero intero. Allo standard input è associato il descrittore 0, allo standard output 1 e allo standard error 2. Ciò ci permette di ridirigere in modo permanente uno dei flussi standard usando il suo descrittore.
Si possono ridirigere i flussi standard su file mediante specifici **operatori di ridirezione.**

Ridirezione dell'output

Qualche volta potrebbe essere necessario salvare l'output di un comando in un file oppure applicare un altro comando sull'output di un comando. Questa operazione è nota come ridirezione dell'output.
La ridirezione si fa utilizzando il carattere **>** (simbolo di maggiore) che invia lo standard output a un file; se questo file esiste già il suo contenuto viene sostituito. Per evitare ciò si deve usare **>>** che aggiunge i dati in fondo al file.

```
comando > file
comando >> file
```

Un altro modo per ridirigere l'output è di usare il comando **`cat`** che concatena i file e li invia tutti insieme allo standard output. Ridirigendo questo output a un file, esso verrà creato o sovrascritto se già esistente.

```
cat fileprimo > filenuovo
cat filesecondo >> filenuovo
```

A fine operazione il `filenuovo` conterrà i dati del primo file e in coda quelli del secondo, concatenando così i due file.

Possiamo creare file vuoti usando il comando:

```
touch nome_file
```

Ridirezione dell'input

Questa operazione permette di considerare il file indicato come standard di input, cioè di leggere i dati dal file e non dalla tastiera.
La ridirezione dell'input viene fatta con il carattere **<** (simbolo di minore).

```
comando < file
```

È possibile reindirizzare lo standard input e output contemporaneamente. Infatti, digitando il comando:

```
comando < ingresso > uscita
```

il comando legge il suo input dal file `ingresso` e reindirizza il risultato al file `uscita`.

Ridirezione dell'errore

Lo standard error è usato per visualizzare i messaggi di errore; per default, esso appare sullo schermo, ma può essere anche reindirizzato su file scrivendo:

```
comando 2> file_errore
```

Se si vuole aggiungere stderr alla fine del file si usa >>:

```
comando 2>> file_errore
```

#prendinota

È possibile ridirezionare stdout e stderr aggiungendoli entrambi alla fine di un file scrivendo in un'unica riga:

```
comando >> file 2>&1
```