IL FILE SYSTEM: CESTIONE FILE

Danilo Croce

Dicembre 2024

PROBLEMI E REQUISITI DI MEMORIZZAZIONE NEL COMPUTING

- **Problemi di Memorizzazione:**
  - **Limitazioni della RAM**: La **RAM fisica offre uno spazio limitato** per salvare dati, insufficiente per molte applicazioni che richiedono molto più spazio, a volte fino a terabyte.
  - **Perdita di Dati**: Le informazioni in RAM vanno **perse al termine del processo** o in caso di **crash del computer o blackout**.
  - **Accesso Concorrente**: La necessità di accesso simultaneo alle informazioni da più processi rende inadeguata la memorizzazione in uno spazio di indirizzi di un unico processo.
- **Requisiti Essenziali per la Memorizzazione a Lungo Termine:**
  1. Capacità di **salvare grandi quantità di informazioni**.
  2. **Persistenza delle informazioni** oltre la vita del processo che le utilizza.
  3. **Accessibilità** delle informazioni **da più processi** simultaneamente.

FILE SYSTEMS

- Pensate a un disco come a una sequenza lineare di blocchi di dimensioni fisse che supporta due operazioni:
  - Leggere il blocco k.
  - Scrivere il blocco k
- Domande che sorgono rapidamente:
  - *"Come si trovano le informazioni?"*
  - *"Come si impedisce a un utente di leggere i dati di un altro utente?"*
  - *"Come si fa a sapere quali blocchi sono liberi?"*
  - …

SOLUZIONI DI MEMORIZZAZIONE E RUOLO DEI FILE SYSTEMS

- **Soluzioni di Memorizzazione:**
  - **Uso di Dischi Magnetici e SSD**: Tradizionalmente, si utilizzano dischi magnetici e unità a stato solido (SSD) per memorizzazione a lungo termine.
  - **Operazioni sui Dischi**: I dischi e gli SSD supportano **operazioni essenziali** come la **lettura** e la **scrittura** di blocchi di dati.
- **File System e Gestione delle Informazioni:**
  - **Astrazione del File**: Il file come **astrazione** risolve il problema di memorizzazione, consentendo la **persistenza, l'accesso multiplo, e la gestione di grandi volumi di dati**.

SOLUZIONI DI MEMORIZZAZIONE E RUOLO DEI FILE SYSTEMS (2)

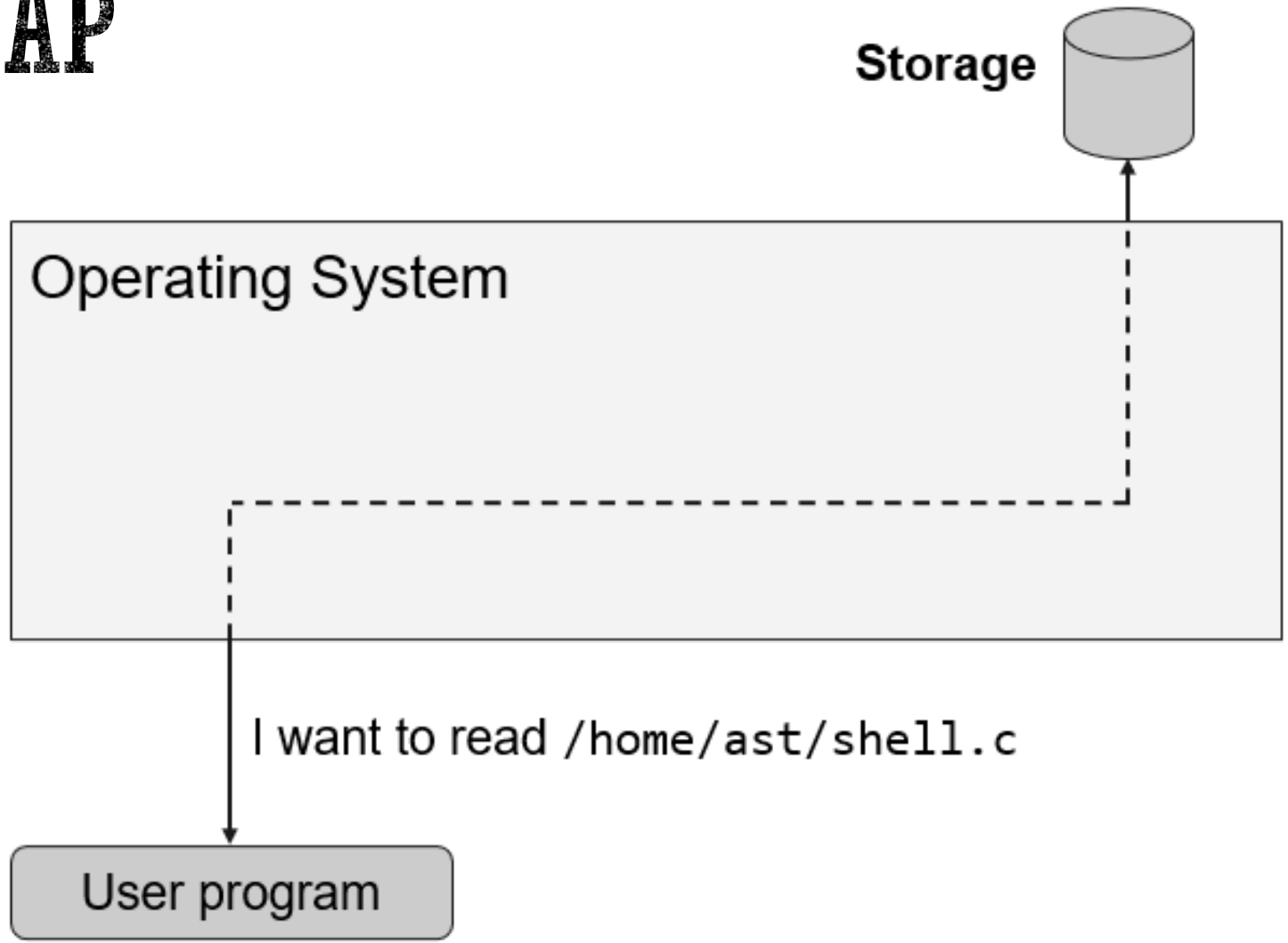
- **Ruolo dei Sistemi Operativi**: I sistemi operativi gestiscono i file attraverso il file system, che si occupa della
  - ***struttura***
  - ***denominazione***
  - ***accesso***
  - ***protezione***
  - ***implementazione***

  dei file.
- **Interfaccia Utente VS Implementazione Tecnica**
  - **Aspetti visibili all'utente** (nomi dei file, operazioni consentite)
  - **Aspetti tecnici rilevanti** per i progettisti del sistema (gestione della memoria, struttura interna del file system).

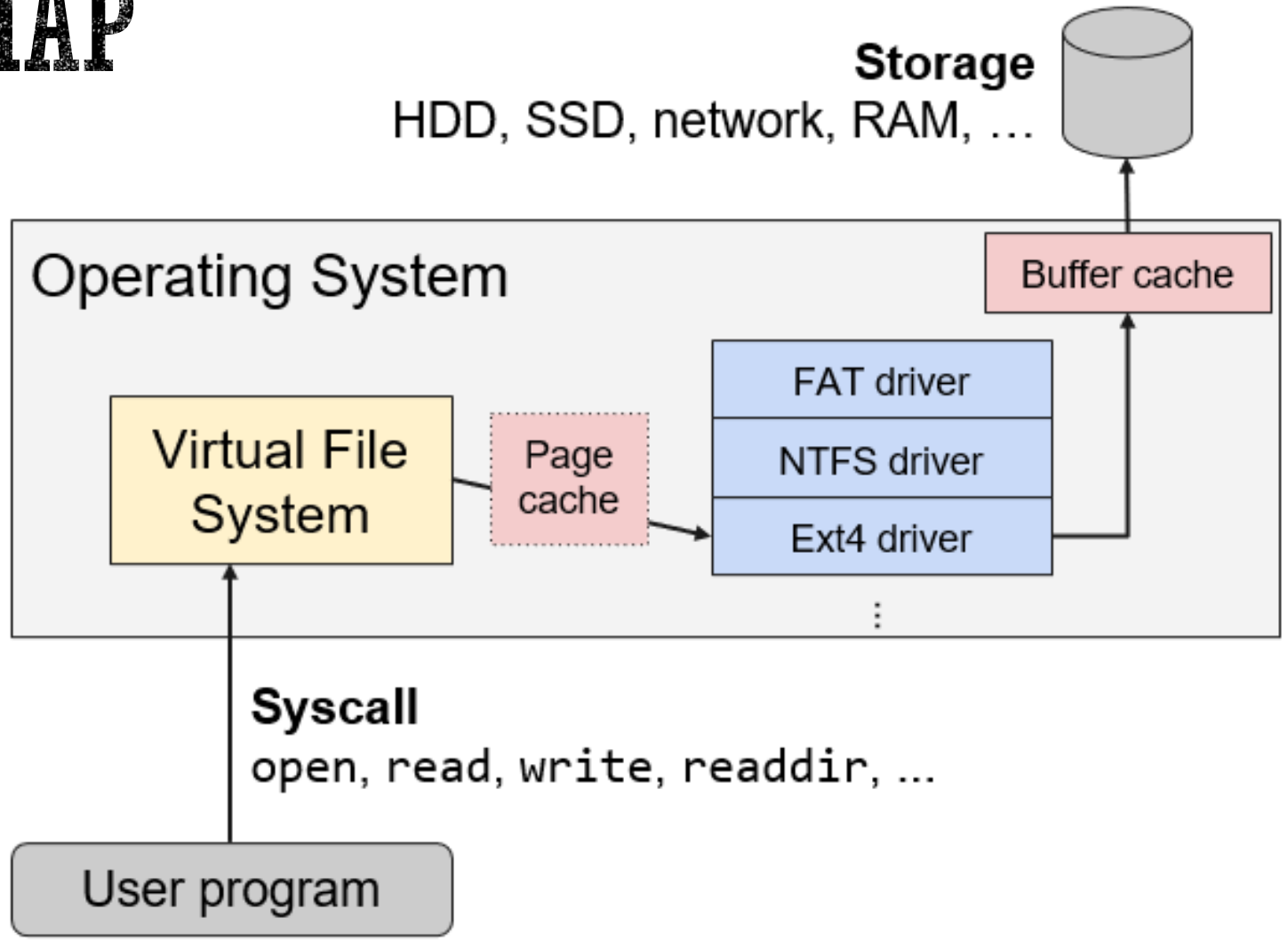
COSA SONO I FILE SYSTEM?

- I **file system** sono:
  - Un modo per **organizzare e memorizzare (in modo persistente) le informazioni.**
  - Un'**astrazione sui dispositivi di memorizzazione**:
    - Disco rigido, SSD, rete, RAM, …
- **Organizzati in file e (tipicamente) directory.**
- **Esempi:**
  - FAT12/FAT16: MS-DOS
  - NTFS: Windows
  - Ext4: Linux
  - APFS: mac OS/iOS

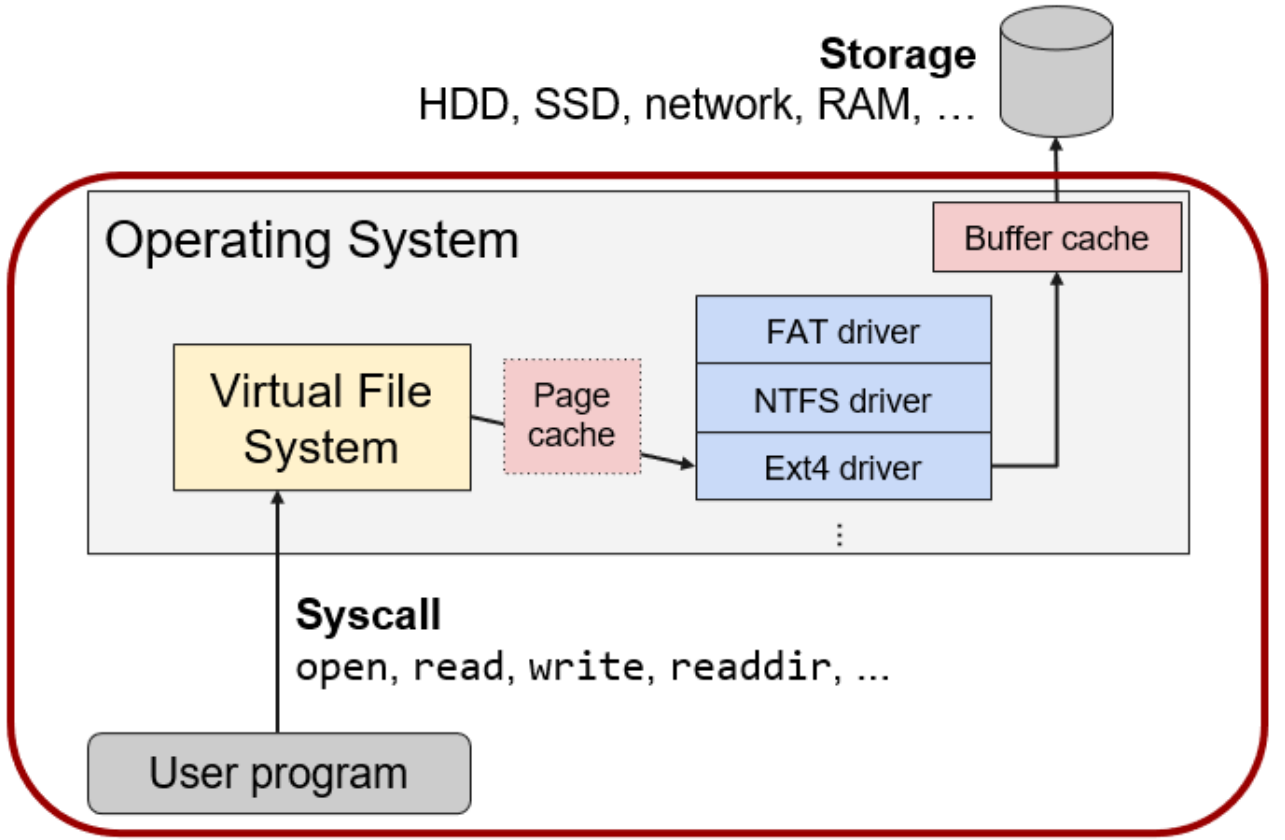
ROADMAP


Storage
Operating System
I want to read /home/ast/shell.c
User program

ROADMAP

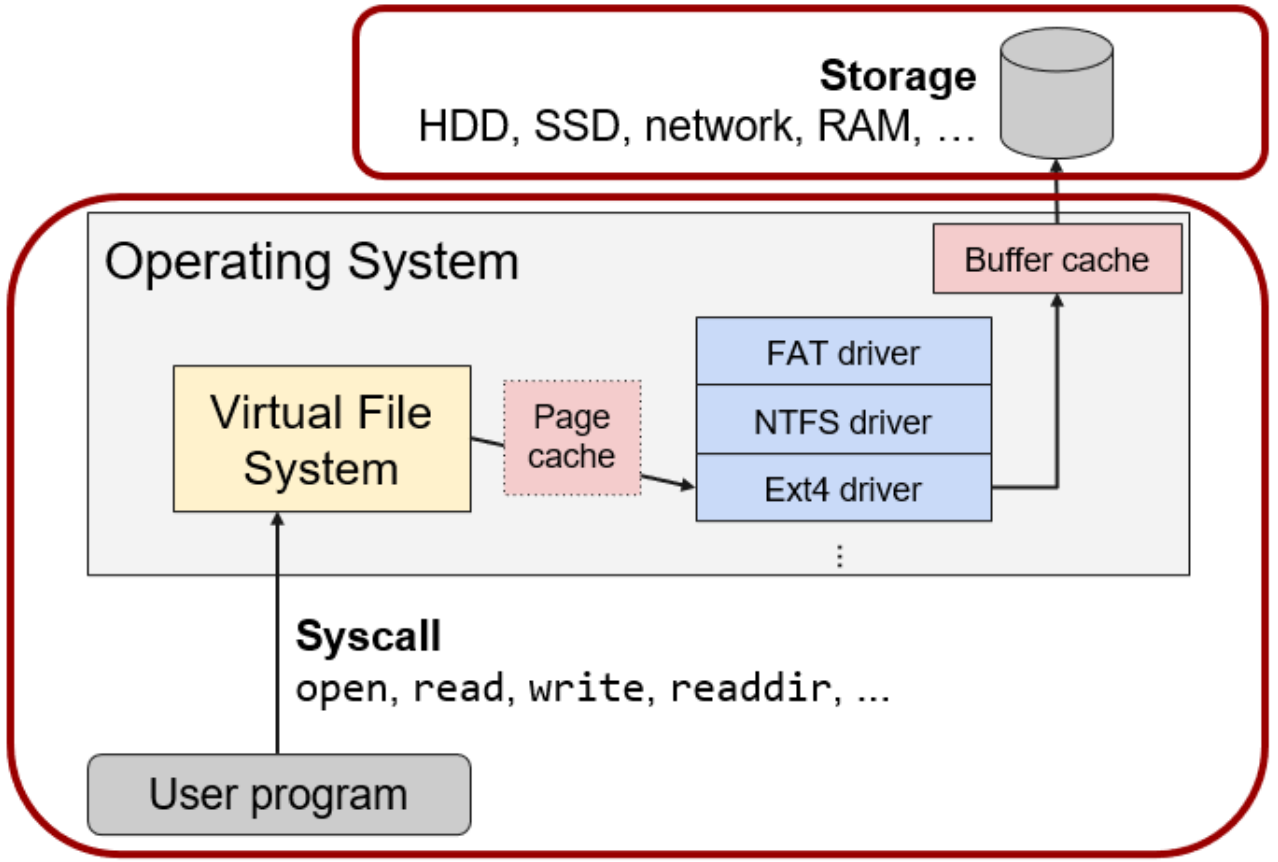

Storage
HDD, SSD, network, RAM, …
Operating System
Buffer cache
FAT driver
NTFS driver
Ext4 driver
Virtual File System
Page cache
Syscall
open, read, write, readdir, ...
User program

ROADMAP



Lezioni su File System

ROADMAP



Lezioni su I/O

Lezioni su File System

I FILE

CARATTERISTICHE DEI NOMI DEI FILE

- **File e astrazione**: I file fungono da metodo di **astrazione per salvare e leggere informazioni** su disco
  - **Nascondendo i dettagli tecnici** all'utente.
- **Nomenclatura dei File**: I file vengono identificati tramite nomi, che possono variare in base al sistema operativo.
  - Esempi comuni includono nomi con lettere, numeri e caratteri speciali.
- **Lunghezza e «Sensibilità» dei Nomi**: Alcuni sistemi operativi limitano la lunghezza dei nomi dei file (es. 8 lettere in MS-DOS) mentre altri supportano nomi più lunghi.
- **Evoluzione dei File System**: Vari file system sono stati sviluppati nel corso del tempo, tra cui FAT-16, FAT-32 e NTFS.
  - Questi sistemi variano in termini di proprietà come la costruzione dei nomi dei file e il supporto per Unicode.

CARATTERISTICHE DEI NOMI DEI FILE (2)

- **Caratteri speciali nei nomi dei file**
  - FAT12: No "*|<>?\" e altro
  - Ext4: No 10' e ", o i nomi speciali "." e "..".
- **Case sensitivity:** Sistemi come UNIX distinguono tra maiuscole e minuscole, a differenza di MS-DOS.

ESTENSIONI DEI FILE E LORO SIGNIFICATO

- **Estensioni di File**: Le estensioni sono parti di nomi di file che seguono un punto, indicando generalmente una caratteristica specifica del file
  - Esempio: .jpg per immagini JPEG, .mp3 per musica MPEG layer 3).
- **Ruolo delle Estensioni**: In alcuni sistemi, le estensioni sono puramente convenzionali e non richieste dal sistema operativo (come in UNIX), mentre in altri (come Windows) hanno un significato specifico e sono associati a programmi specifici.
- **Gestione delle Estensioni in Windows**: In Windows, **le estensioni** dei file sono **registrate nel sistema operativo e associate a programmi specifici** che si avviano quando l'utente interagisce con il file
  - **Esempio**: apertura di un file .docx con Microsoft Word

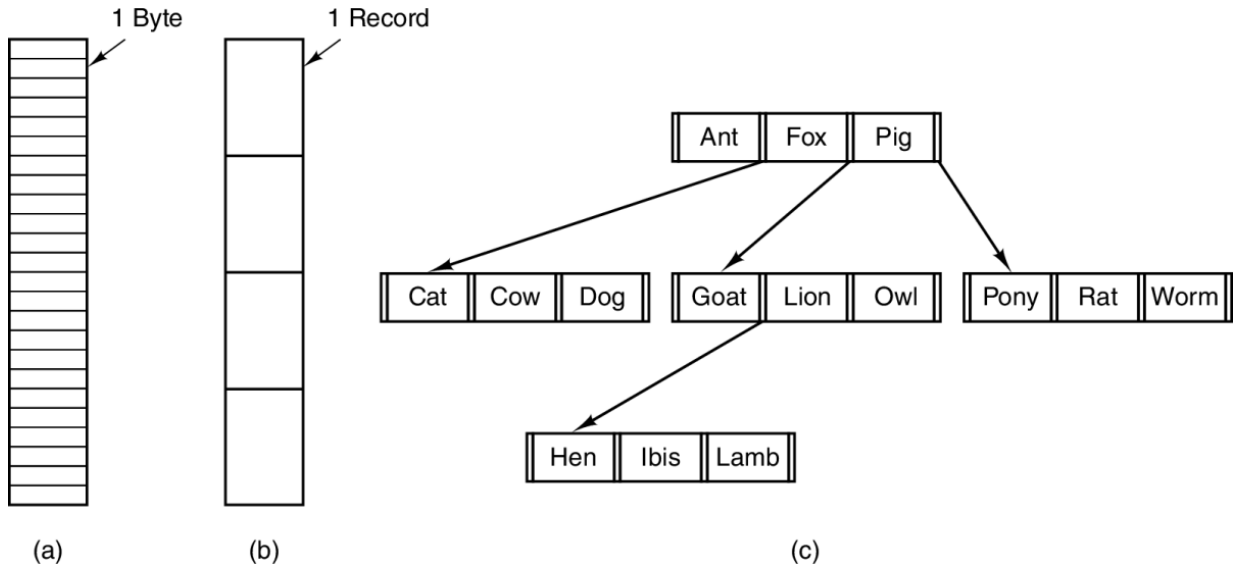
ALCUNE TIPICHE ESTENSIONI DI FILE

Estensione	Significato
.bak	File di backup
.c	Programma sorgente in linguaggio C
.gif	Immagine in Compuserve Graphical Interchange Format
.html	Documento HTML (world wide web hypertext markup language)
.jpg	Immagine codificata con lo standard JPEG
.mp3	Musica codificata in formato audio MPEG layer 3
.mpg	Filmato codificato in formato audio MPEG standard
.o	File oggetto (output da compilatore, non ancora linkato)
.pdf	Documento in formato Adobe PDF (portable document format)
.ps	File PostScript
.tex	Input per il programma di formattazione TEX
.txt	File di testo generico
.zip	Archivio compresso

TIPOLOGIE DI STRUTTURA DEI FILE

a) **Sequenza Non Strutturata di Byte:**
- I file sono **visti dal sistema operativo come una serie non strutturata** di byte.
- Il **significato dei dati è determinato dai programmi a livello utente**, non dal sistema operativo.
- Questo approccio è adottato da sistemi operativi come UNIX, Linux, macOS e Windows, offrendo massima flessibilità.

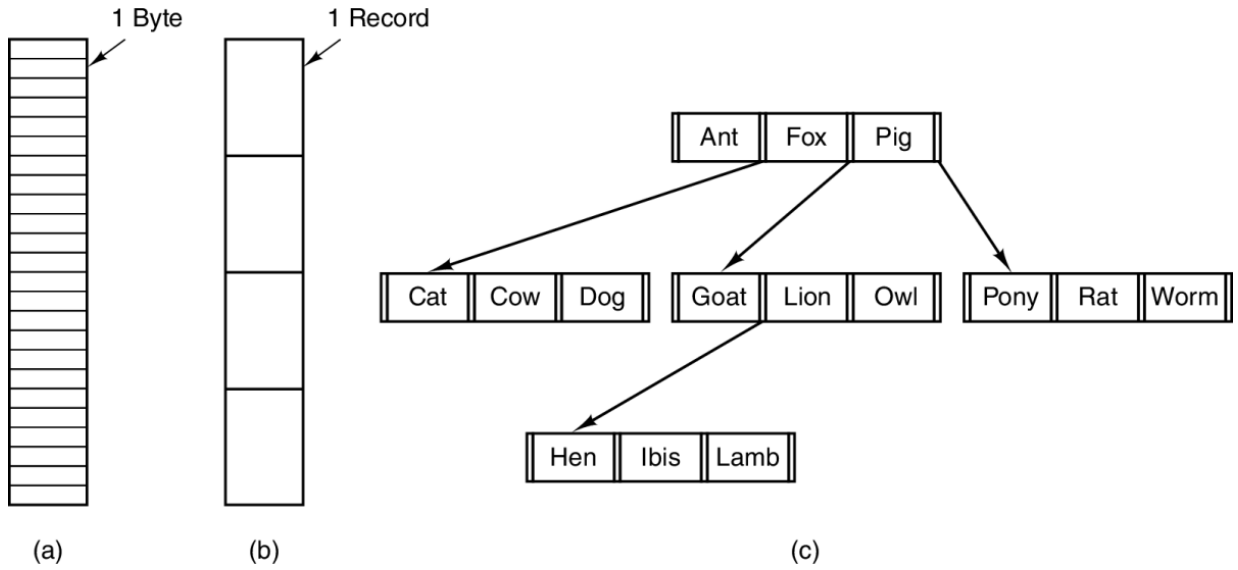
b) **Sequenza di Record di Lunghezza Fissa:**
- Un file è **una sequenza di record con una struttura interna definita** e lunghezza fissa.
- Il modello storico basato su schede perforate a 80 colonne in mainframe.
- **Letture e scritture avvengono a unità di record**, meno comune nei sistemi operativi moderni ma era prevalente nei mainframe del passato.



TIPOLOGIE DI STRUTTURA DEI FILE (2)

c) **File come Albero di Record:**

- Il file è **organizzato come un albero di record**, con lunghezze variabili e un campo chiave in posizione fissa.
- L'organizzazione **consente ricerche rapide** basate su chiavi specifiche.
- Utilizzato **principalmente in sistemi mainframe** per elaborazioni dati di carattere commerciale, diverso dalle sequenze non strutturate di UNIX e Windows.



TIPI DI FILE E LORO STRUTTURE

- **File e Directory Normali:**
  - **Sistemi Operativi:** Utilizzati in UNIX (inclusi macOS e Linux) e Windows.
  - **File Normali:** Contengono informazioni utente e sono la forma più comune.
  - **Directory:** File di sistema per mantenere la struttura del file system.
- **File Speciali:**
  - **File Speciali a Caratteri:** Usati per modellare dispositivi seriali di I/O come terminali e stampanti.
  - **File Speciali a Blocchi:** Usati per modellare dischi.
- **Tipi di File Normali:**
  - **File ASCII:** Composti da righe di testo, visualizzabili e stampabili; variano nella terminazione delle righe (carattere di "a capo" o "nuova riga").
  - **File Binari:** Non leggibili come testo; hanno una struttura interna conosciuta dai programmi che li utilizzano. Esempi includono file eseguibili e archivi.

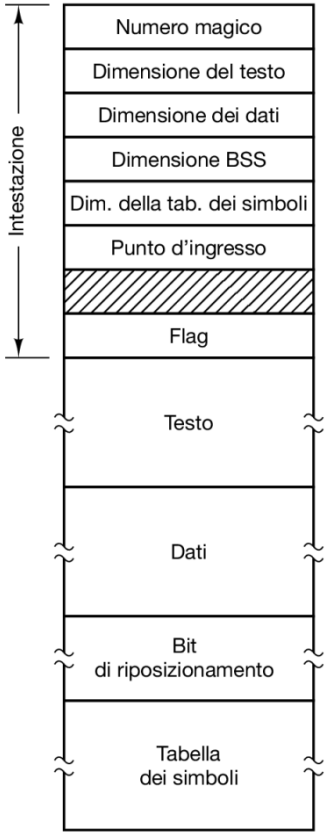
FILE E STRUTTURE INTERNE

a) **File Eseguibile :**

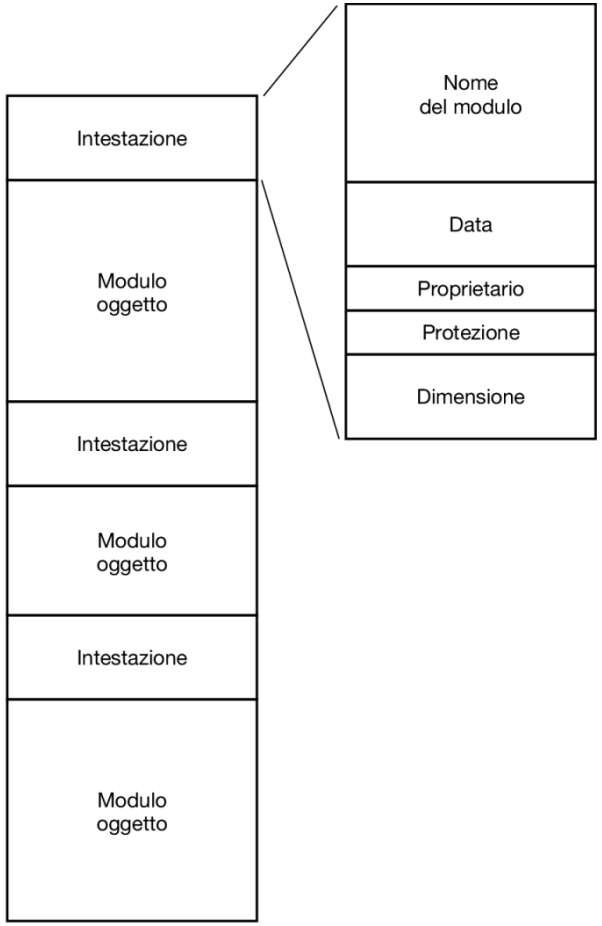
- Esempio da una delle prime versioni di UNIX.
- Sequenza di byte con formato specifico per l'esecuzione.
- **Componenti:**
  - **Intestazione (Header):** Contiene un '***numero magico***' per identificare il file come eseguibile (e non eseguire file «non eseguibili»), dimensioni delle parti del file, indirizzo di esecuzione iniziale e flag.
  - **Testo e Dati:** Parti effettive del programma, caricate e rilocate in memoria.
  - **Tabella dei Simboli:** Utilizzata per il debug.

b) **File di Archivio**

- **Descrizione:** Raccolta di procedure di libreria (moduli) compilate ma non collegate.
- **Intestazioni dei Moduli:** Indicano nome, data di creazione, proprietario, codice di protezione e dimensione.
- **Carattere Binario:** Stampare questi file produrrebbe caratteri incomprensibili.
- Per creare archivi: vedi ultime slide



(a)



(b)

RICONOSCIMENTO E GESTIONE DEI TIPI DI FILE

- **Riconoscimento dei Tipi di File:**
  - **Esempi di File Binari:** File eseguibili con intestazioni e dati specifici; archivi con moduli di libreria e intestazioni dettagliate.
  - **Sistemi Operativi e File Tipizzati:** Alcuni sistemi, come il vecchio TOPS-20, avevano meccanismi complessi per riconoscere i tipi di file, che potevano portare a limitazioni nell'uso dei file.
- **Strumenti di Esame dei File:**
  - **Utility in UNIX:** L'utility `file` in UNIX usa euristiche per determinare il tipo di file (testo, directory, eseguibile, ecc.),

METODI DI ACCESSO AI FILE

- **Evoluzione dell'Accesso ai File:**
  - **Accesso Sequenziale:** Nei primi sistemi operativi, era l'unico metodo disponibile.
  - Consentiva la lettura dei file dall'inizio alla fine, adatto ai nastri magnetici.
  - **Accesso Casuale:** Introdotta con l'avvento dei dischi, permette la lettura di byte o record in qualsiasi ordine, senza seguire una sequenza.
- **Importanza dell'Accesso Casuale:**
  - Cruciale per **applicazioni** come i sistemi di database, dove è necessario accedere rapidamente a record specifici senza attraversare l'intero file.
- **Metodi di Specificazione della Posizione di Lettura:**
  - **Lettura con Posizione Specifica:** Ogni operazione di lettura inizia da una posizione definita all'interno del file.
  - **Utilizzo di seek:** Un'operazione speciale per impostare la posizione corrente nel file, dopo la quale il file può essere letto sequenzialmente dalla posizione impostata.
  - **Implementazione:** Questo metodo di accesso è adottato sia in UNIX sia in Windows.

ATTRIBUTI DEI FILE

- **Definizione:** Oltre a nome e dati, ogni file ha attributi (o metadati) che variano a seconda del sistema operativo.
- **Esempi di Attributi Comuni:**
  - **Protezione e Accesso:** Indica chi può accedere al file e come (es. proprietario, creatore, password).
  - **Flag Specifici:** Diversi flag per controllo (es. sola lettura, file nascosto, file di sistema, file di backup).
  - **Tipologia del File:** Indica se il file è ASCII o binario, accesso casuale o sequenziale.
  - **Attributi Temporali:** Data e ora di creazione, ultimo accesso, ultima modifica.
  - **Dimensione:** Dimensione attuale e massima del file.
  - **Gestione dei Record (per file basati su record):** Lunghezza del record, posizione e lunghezza della chiave.
- **Importanza:** Gli attributi dei file sono cruciali per
  - la protezione, il controllo dell'accesso
  - la gestione efficace dei file nei sistemi operativi.

ESEMPI DI ATTRIBUTI

Attributo	Significato	Attributo	Significato
Protezione	Chi può accedere al file e in che modalità	Flag temporaneo	0 per normale; 1 per cancellare il file al termine del processo
Password	Password necessaria per accedere al file	Flag di file bloccato	0 per non bloccato; non zero per bloccato
Creatore	ID della persona che ha creato il file	Lunghezza del record	Numero di byte nel record
Proprietario	Proprietario attuale	Posizione della chiave	Offset della chiave in ciascun record
Flag di sola lettura	0 per lettura/scrittura; 1 per sola lettura	Lunghezza della chiave	Numero di byte del campo chiave
Flag di file nascosto	0 per normale; 1 per non visualizzare negli elenchi	Data e ora di creazione del file	Data e ora di quando il file è stato creato
Flag di file di sistema	0 per file normali; 1 per file di sistema	Data e ora di ultimo accesso al file	Data e ora di quando è avvenuto l'ultimo accesso al file
Flag di file archivio	0 per già sottoposto a backup; 1 per file di cui fare il backup	Data e ora di ultima modifica al file	Data e ora di quando è avvenuta l'ultima modifica al file
Flag ASCII/binario	0 per file ASCII; 1 per file binari	Dimensione attuale	Numero di byte nel file
Flag di accesso casuale	0 per accesso sequenziale; 1 per accesso casuale	Dimensione massima	Numero di byte di cui può aumentare il file

OPERAZIONI SU FILE

1. **Create:** Creazione di un file senza dati
   - principalmente per «annunciare» la presenza del file e impostare alcuni attributi.
2. **Delete:** Eliminazione di un file per liberare spazio su disco, attraverso una specifica chiamata di sistema.
3. **Open:** Apertura di un file per consentire al sistema di caricare in memoria gli attributi e gli indirizzi del disco
   - facilita l'accesso rapido in seguito.
4. **Close:** Chiusura del file al termine degli accessi per liberare spazio nelle tabelle interne
   - forza anche la scrittura dell'ultimo blocco del file.
5. **Read:** Lettura dei dati da un file, generalmente dalla posizione corrente
   - con specificazione della quantità di dati richiesti e fornitura di un buffer per la loro memorizzazione.

OPERAZIONI SU FILE (2)

6. **Write:** Scrittura di dati nel file, tipicamente alla posizione corrente,
   - può comportare l'ampliamento del file o la sovrascrittura dei dati esistenti.
7. **Append:** Aggiunta di dati solo alla fine del file,
   - usata in alcuni sistemi operativi come forma limitata di scrittura.
8. **Seek:** Riposizionamento del puntatore del file su una posizione specifica per file ad accesso casuale, permettendo la lettura o la scrittura da quella posizione.
9. **Get Attributes:** Lettura degli attributi di un file,
   - necessaria per alcuni processi per svolgere le loro funzioni (es. il programma UNIX make per la gestione dei progetti software).
10. **Set Attributes:** Modifica degli attributi di un file da parte dell'utente,
    - come la modalità di protezione o altri flag, dopo la creazione del file.
11. **Rename:** Ridenominazione di un file,
    - utilizzata come alternativa al processo di copia ed eliminazione del file originale, specialmente utile per file di grandi dimensioni.

OPERAZIONI SU FILE IN UNIX (1)

- Aprire e leggere un file:

```
int fd = open("foo.txt", O_RDONLY);
char buf[512];
ssize_t bytes_read = read(fd, buf, 512);
close(fd);
printf("read %zd: %s\n", bytes_read, buf);
```

- L'apertura di un file restituisce un handle (descrittore di file) per le operazioni future.
- Qualsiasi funzione può restituire un errore, ad es:
  - `ENOENT`: File does not exist
  - `EBADF`: Bad file descriptor

OPERAZIONI SU FILE IN UNIX (2)

- Posizionamento (seek) nei file :

```
int fd = open("foo.txt", O_RDONLY);
lseek(fd, 128, SEEK_CUR);
char buf[8];
read(fd, buf, 8);
close(fd);
```

- Sposta la posizione corrente nel file in avanti o indietro:
  - Sposta il puntatore del file di 128 byte in avanti dalla posizione corrente nel file.
    - L'uso di `SEEK_CUR` indica che il movimento è relativo alla posizione corrente del puntatore nel file.
  - Dichiara un buffer `buf` di dimensione 8 byte e poi legge 8 byte dal file, a partire dalla nuova posizione (che è 128 byte oltre la posizione corrente a causa del precedente `lseek`), e li mette nel buffer buf.

OPERAZIONI SU FILE IN UNIX (3)

- **Apre un file e scrive**

```
int fd = open("foo.txt", O_WRONLY | O_CREAT |  O_TRUNC);
char buf[] = "Hi there";
write(fd, buf, strlen(buf));
close(fd);
```

- `O_CREAT` **e** `O_TRUNC` **sono flag che controllano il comportamento dell'apertura del file.**
  - `O_WRONLY`: **Apre il file in modalità scrittura.**
  - `O_CREAT`: **Crea il file se non esiste.**
  - `O_TRUNC`: **Se il file esiste, ne tronca il contenuto a dimensione 0.**

  sovrascriverà il contenuto esistente di `foo.txt` **con "**`Hi there`**", o creerà un nuovo file con questo contenuto se** `foo.txt` **non esiste.**

GESTIONE DI FILE IN POSIX – ESEMPIO PRATICO

- **Obiettivi dell'esempio**
  - 1. Dimostrare l'uso delle funzioni POSIX per la gestione di file:
    - **open**, **write**, **read**, **close**.
  - 2. Confrontare i metodi di scrittura/lettura binaria e leggibile:
    - **Binario**: Più efficiente, ma non leggibile dall'utente.
    - **Testuale**: Più leggibile, ma meno efficiente.
  - 3. Introdurre una funzione custom read_line:
    - Simula la lettura di una riga da un file, fino al carattere `\n`.

File di riferimento: `11_write_and_read_POSIX.c`

ALTRE OPERAZIONI SU FILE IN UNIX

- Rimuovere file:
  - `unlink("foo.txt");`
- Rinominare file:
  - `rename("foo.txt", "bar.txt");`
- Cambiare i *file permission attribute*:
  - `chmod("foo.txt", 0755);`
- Cambiare la proprietà del file:
  - `chown("foo.txt", uid, gid);`

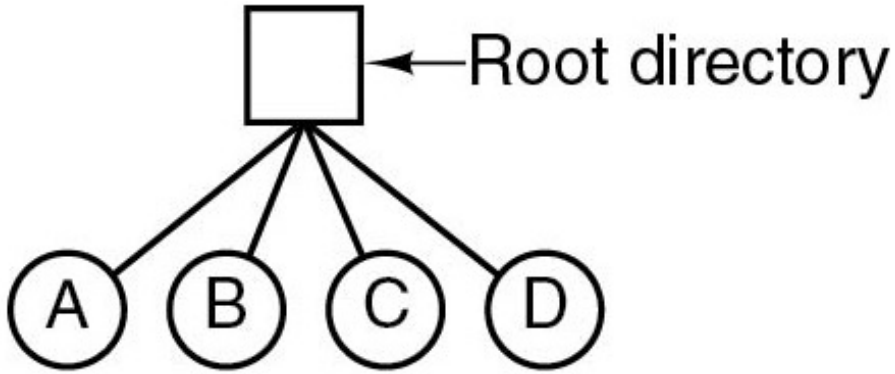
UN PROGRAMMA PER COPIARE FILE
SINTESI DEL PROGRAMMA `11_COPYFILE.C`

- **Funzionalità**: Programma in C per copiare il contenuto di un file di input in un file di output.
- **Controlli**: Verifica la correttezza del numero di parametri forniti all'applicazione.
- **Operazioni di File**:
  - Apertura del file sorgente per la lettura.
  - Creazione/apertura del file destinazione con permessi specifici.
  - Copia del contenuto tramite un buffer di 4096 byte.
- **Gestione degli Errori**: Esce con un codice di errore specifico se incontra problemi nell'apertura, nella lettura, nella scrittura o nella chiusura dei file.
- **Chiusura**: I file vengono chiusi e il programma termina con un codice di stato appropriato a seconda dell'esito delle operazioni.

DIRECTORY

LE DIRECTORY NEI FILE SYSTEM

- **Concetto di Directory**:
  - Le directory, o cartelle, sono file che tengono traccia degli altri file all'interno di un file system.
- **Sistemi di Directory a Livello Singolo**:
  - **Struttura Semplice**: Una singola directory, talvolta chiamata root directory, contiene tutti i file.
  - **Esempi Storici**: Comune nei primi PC e nel supercomputer CDC 6600.
  - **Vantaggi**: Semplicità e rapidità nella localizzazione dei file.



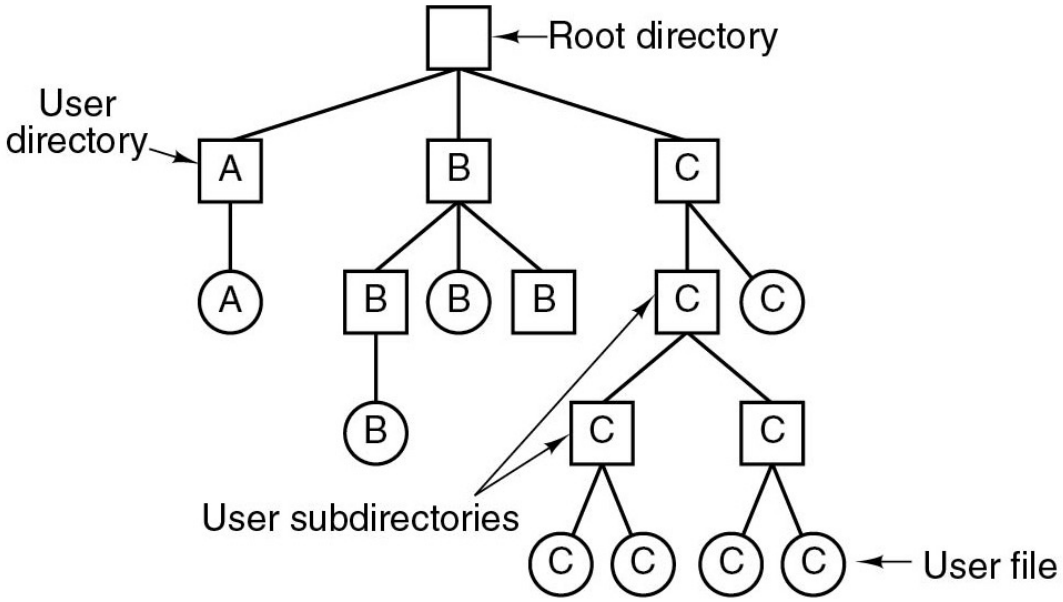
Esempio con quattro file in un sistema a directory singola.

EVOLUZIONE E APPLICABILITÀ DEI SISTEMI DI DIRECTORY A LIVELLO SINGOLO

- **Evoluzione dei Concetti di File System**:
  - Molti concetti, come la directory singola, sono ciclici: emergono, cadono in disuso, e riemergono in nuovi contesti.
- **Applicabilità Moderna**:
  - **Dispositivi Embedded**: Concetti semplici di file system sono ancora utili in dispositivi come fotocamere digitali o riproduttori MP3.
  - **Tecnologie RFID**: Sistemi di directory semplici possono essere adatti per chip RFID o carte di credito e tessere di trasporto.
  - **Riflessione**: Idee apparentemente obsolete possono essere rilevanti in contesti moderni e dispositivi a basso costo.

SISTEMI DI DIRECTORY GERARCHICI

- **Limiti dei Sistemi a Singolo Livello**:
  - **Non pratici** per utenti con **migliaia di file**.
  - Difficoltà nel rintracciare i file in un unico spazio.
- **Introduzione della Gerarchia**:
  - Organizzazione dei file in gruppi correlati mediante directory ramificate.
  - Struttura ad albero per separare e organizzare logicamente i file.
  - **Ogni utente può avere una directory principale** privata in ambienti condivisi come reti aziendali.
- **Importanza nei File System Moderni**:
  - Tutti i file system moderni utilizzano una struttura gerarchica per la loro flessibilità e potenziale organizzativo.
  - Storicamente, il file system gerarchico è stato sperimentato inizialmente in Multics negli anni '60.



La directory principale (Root) divisa in directory A, B e C, ognuna appartenente a utenti diversi.

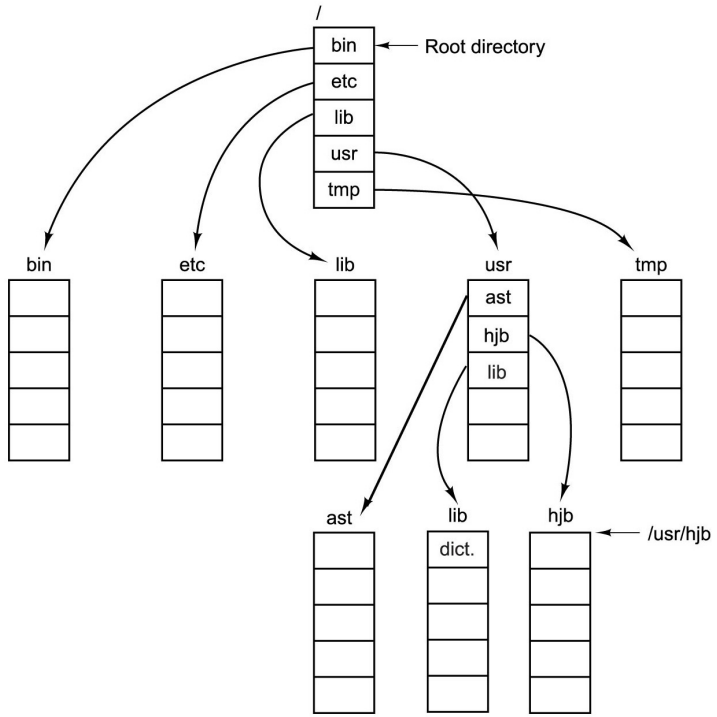
- Possibilità di creare sottodirectory per progetti specifici o categorie di file.

NOMI DI PERCORSO NEI FILE SYSTEM GERARCHICI

- **Specificare i Nomi dei File**:
  - Necessità di definire i percorsi dei file in un sistema di directory ad albero.
- **Nomi di Percorso Assoluti**:
  - Percorsi che iniziano dalla directory principale e conducono al file.
  - Unici per ogni file (es. `/usr/ast/mailbox`).
  - Separatore di percorso: / per UNIX, \ per Windows, > per MULTICS.
- **Nomi di Percorso Relativi**:
  - Basati sulla directory di lavoro (directory corrente) dell'utente.
  - Percorsi non iniziano con il separatore sono considerati relativi (es. mailbox).
  - Esempi di comandi equivalenti in una data directory di lavoro:
    - `cp /usr/hjb/mailbox /usr/hjb/mailbox.bak`
    - `cp mailbox mailbox.bak`

UTILIZZO PRATICO E IMPLICAZIONI

- **Directory di Lavoro (Working Directory)**:
  - Cambia dinamicamente per ciascun processo.
  - Non influisce sugli altri processi o sul file system dopo l'uscita del processo.
- **Procedure di Libreria**:
  - Evitano di cambiare la directory di lavoro o la ripristinano dopo il loro uso.
- **Voci Speciali**:
  - `.` (punto): Rappresenta la directory corrente.
  - `..` (punto punto): Rappresenta la directory genitore.
  - Usati per navigare nell'albero dei file
    - **Esempio** `cp ../lib/dictionary .`



Un esempio di albero di directory UNIX.

OPERAZIONI SULLE DIRECTORY

- **Operazioni di Base**:
  - `create`: Creazione di una directory vuota con le voci "." e ".." predefinite.
  - `delete`: Eliminazione di una directory, possibile solo se la directory è vuota.
  - `opendir`: Apertura di una directory per la lettura del suo contenuto.
  - `closedir`: Chiusura di una directory dopo la lettura per liberare risorse.
- **Lettura e Modifica**:
  - `readdir`: Restituisce la prossima voce in una directory aperta senza esporre la struttura interna.
  - `rename`: Rinomina di una directory, simile al rinomino di un file.

GESTIONE DEI LINK E ACCESSI AVANZATI

- **Linking e Unlinking**:
  - `link`: Crea un hard link, collegando un file esistente a un nuovo percorso, condividendone l'i-node.
  - `unlink`: Rimuove una voce di directory, cancellando il file se è l'unico link.
- **Link Simbolici** (*vedi lezioni successive*)
  - Varianti dei hard link che possono puntare a file su dischi o computer diversi.
  - Rappresentano un file tramite un riferimento indiretto che il file system risolve all'uso
    - (meglio nella prossima lezione)
- **Considerazioni Aggiuntive**:
  - Esistono altre chiamate per gestire dettagli come le informazioni di protezione di una directory.
  - I link simbolici offrono flessibilità oltre i limiti dei dischi ma possono essere meno efficienti rispetto agli hard link.

DESCRIZIONE DEL PROGRAMMA

11_SHOW_DIR_CONTENT.C

- **Funzionalità Aggiunta**: Oltre a stampare i nomi, mostra informazioni dettagliate sui file nella directory, simili al comando `ls -lh`.
- **Informazioni Incluse**:
  - Dimensione del file.
  - Permessi di accesso (lettura, scrittura, esecuzione).
  - Proprietario e gruppo del file.
  - Data e ora dell'ultima modifica.
- **Implementazione**:
  - Uso della funzione `stat()` per ottenere i metadati dei file.
  - Formattazione e stampa delle informazioni in modo chiaro e leggibile.

CREAZIONE DI ARCHIVI

COMPRENDERE I FILE TAR.GZ PER LA COMPRESSIONE IN LINUX

- **File TAR: Cosa Sono e Perché si Usano**
  - TAR (Tape Archive) è un formato usato per raccogliere più file e cartelle in un unico archivio, mantenendo la struttura e i permessi originali.
  - Utilizzato comunemente per raggruppare file correlati per backup, trasferimento o archiviazione.
- **Comprimere con GZ**
  - Dopo l'archiviazione con tar, l'archivio viene compresso con gzip per ridurre lo spazio su disco.
  - gzip è un algoritmo di compressione che riduce efficacemente la dimensione del file senza perdita di dati.

UTILIZZO DI TAR E GZ

- **Creazione di un Archivio tar.gz**
  - Comando di Base: `tar -czvf nome-archivio.tar.gz /percorso/della/cartella`
    - `c`: crea un nuovo archivio.
    - `z`: comprime l'archivio usando gzip.
    - `v`: visualizza un output verboso.
    - `f`: specifica il nome del file di archivio.
- **Estrazione di un Archivio tar.gz**
  - Comando di Base: `tar -xzvf nome-archivio.tar.gz`
    - `x`: estrae il contenuto dall'archivio.
    - `z`: decomprime l'archivio usando gzip.
    - `v`: visualizza un output verboso.
    - `f`: specifica il nome del file di archivio.
- In realtà viene creato un file con il comando tar e poi compresso con gzip.
  - Nulla vieta di comprimere con `gzip` un qualsiasi file

CONFRONTO TRA ZIP/UNZIP E TAR.GZ PER LA COMPRESSIONE IN LINUX

- **ZIP e UNZIP: Caratteristiche**
  - ZIP è un formato di compressione che riduce la dimensione dei file singolarmente prima di archiviarli insieme.
  - UNZIP è utilizzato per decomprimere e estrarre i file dagli archivi ZIP.
  - Comandi Comuni:
    - `zip nome-archivio.zip file1 file2,`
    - `unzip nome-archivio.zip.`
  - Vantaggi: Compatibilità ampia con diversi sistemi operativi, compressione individuale dei file.
- **TAR.GZ: Caratteristiche**
  - TAR raccoglie molti file in un unico archivio, poi GZ (gzip) comprime l'intero archivio.
  - **Vantaggi**: Elevata compressione, conservazione della struttura delle directory e dei permessi dei file.
- **Confronto tra ZIP e TAR.GZ**
  - **Efficacia di Compressione**: TAR.GZ tende ad avere un tasso di compressione più alto, specialmente per archivi di grandi dimensioni.
  - **Velocità**: ZIP può essere più veloce nella compressione di file individuali.
  - **Conservazione dei Metadati**: TAR.GZ mantiene meglio la struttura originale e i permessi dei file.
  - **Compatibilità Universale**: ZIP è più comunemente supportato su diverse piattaforme, incluse Windows e macOS.