Modelli e problematiche di *file system*
Aspetti generali – 1

- La maggior parte dell'informazione applicativa (i dati) ha durata, ambito e dimensione più ampi della vita delle applicazioni che la usano
  - 3 le esigenze più evidenti
    - Nessun limite di dimensione fissato a priori
    - Persistenza dei dati
    - Condivisione dei dati tra applicazioni distinte
- Il *file system* è il servizio di S/O progettato per soddisfare questi bisogni

Modelli e problematiche di *file system*
Aspetti generali – 2

- Il termine *file* designa un insieme di dati correlati, residenti in memoria secondaria e trattati unitariamente
  - *File* = raccoglitore, *dossier*
- Il termine *file system* (FS) designa la parte di S/O che si occupa di organizzazione, gestione, realizzazione ed accesso ai *file*

Modelli e problematiche di *file system*
Aspetti generali – 3

- La progettazione di FS affronta 2 problemi chiave
  - Cosa occorre offrire all'utente applicativo e secondo quali forme concrete
    - Modalità di accesso a *file*
    - Struttura logica e fisica di *file*
    - Operazioni ammissibili su *file*
  - Come ciò possa essere convenientemente realizzato
    - Massima indipendenza dall'architettura fisica di supporto

Modelli e problematiche di *file system*
File

- Il *file* è un **meccanismo di astrazione**
  - Per salvare informazione su memoria secondaria e ritrovarvela in seguito senza doverne conoscere né la struttura logica e fisica né il funzionamento
  - All'utente non interessa come ciò avviene
  - Interessa invece poter designare le proprie unità di informazione mediante nomi logici unici e distinti
  - L'utente vede e tratta solo nomi di *file*
  - Le caratteristiche distintive di un *file* sono
    - Attributi (tra cui il nome)
    - Struttura interna
    - Operazioni ammesse

Modelli e problematiche di *file system*
Attributi – 1

- **Nome**
  - Stringa composta da 8 – 255 caratteri, inclusi numeri e caratteri speciali
  - Con ≥ 1 estensioni che possono designare il "tipo" di *file* come visto dall'utente
    - **MS-DOS** (base di Windows 95 e Windows 98)
      - Nomi da 1 – 8 caratteri, con ≤ 1 estensione da 1 – 3 caratteri, designante, senza distinzione tra maiuscolo e minuscolo (*case insensitive*)
    - **UNIX** (base di GNU/Linux)
      - Nomi fino a 14 (ora 255) caratteri, *case sensitive*, con estensioni, solo informative, senza limite di numero e di ampiezza
  - In generale, l'utente può configurare presso il S/O l'associazione tra l'ultima estensione del *file* ed il tipo applicativo corrispondente

Modelli e problematiche di *file system*
Attributi – 2

- Altri attributi significativi
  - **Dimensione corrente**
  - **Data di creazione** (può non essere mostrata)
  - **Data di ultima modifica**
    - Indica la "freschezza" del contenuto
  - **Creatore e possessore** (anche distinti)
    - P.es.: il compilatore crea *file* di proprietà dell'utente
  - **Permessi di accesso**
    - Lettura, scrittura, esecuzione

Modelli e problematiche di *file system* *Attributi – 3*

			Flag
Protezione	Permesso di accesso al *file*		
Password	Chiave di accesso al *file*		
Creatore	Identità del processo che ha creato il *file*		
Proprietario	Identità del processo utilizzatore del *file*		
Uso	0 – lettura/scrittura	1 – sola lettura (*read-only*)	
Visibilità	0 – normale	1 – *file* non visibile (*hidden*)	
Livello	0 – normale	1 – *file* di sistema	
Archiviazione	0 – salvato (*backed up*)	1 – non salvato	
Tipo di contenuto	0 – ASCII	1 – binario	
Tipo di accesso	0 – sequenziale	1 – casuale (*random*)	
Permanenza	0 – normale	1 – da eliminare dopo l'uso (*temporary*)	
Accesso esclusivo	0 – libero	≠ 0 – bloccato (*locked*)	

Modelli e problematiche di *file system* *Struttura – 1*

- La struttura dei dati all'interno di un *file* può essere vista da 3 livelli di astrazione distinti
  - Livello **utente**
    - Il programma utente associa significato al contenuto grezzo del *file*
  - Livello di **struttura logica**
    - I dati grezzi (non interpretati) sono raggruppati dal S/O in strutture logiche per facilitarne il trattamento
  - Livello di **struttura fisica**
    - Il S/O mappa le strutture logiche sulle strutture fisiche della memoria secondaria disponibile (p.es.: settori o blocchi su disco)
- Le possibili strutture logiche di un *file* sono
  - A sequenza di *byte*
  - A *record* di lunghezza e struttura interna fissa
  - A *record* di lunghezza e struttura interna variabile

Modelli e problematiche di *file system* *Struttura – 2*

- **Sequenza di *byte*** (*byte stream*)
  - La strutturazione logica più rudimentale e flessibile
    - La scelta di UNIX (→ GNU/Linux) e Windows
  - Il programma utente sa come dare significato al contenuto informativo del *file*
    - Minimo sforzo per il S/O
  - L'accesso ai dati utilizza un puntatore relativo all'inizio del *file*
  - Lettura e scrittura operano a blocchi di *byte*

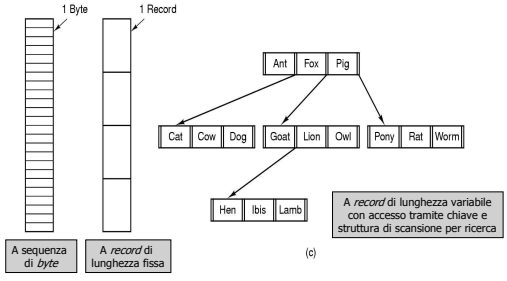
Modelli e problematiche di *file system* *Struttura – 3*

- ***Record* di lunghezza e struttura fissa**
  - Gli spazi non utilizzati sono riempiti da caratteri speciali (p.es.: NULL o SPACE)
  - Il S/O conosce la struttura del *file*
  - L'accesso ai dati è sequenziale e utilizza un puntatore al *record* corrente
  - Lettura e scrittura operano su *record* singoli
  - Scelta obsoleta e legata a specifiche limitazioni dell'architettura di sistema

Modelli e problematiche di *file system* *Struttura – 4*

- ***Record* di lunghezza e struttura variabile**
  - Struttura interna di ogni *record* descritta e identificata univocamente da una chiave (*key*) posta in posizione fissa e nota nel *record*
  - Chiavi raccolte in una tabella a se stante, ordinata per chiave, contenente anche i puntatori all'inizio di ciascun *record*
  - Accesso ai dati per chiave di *record*
  - Uso abbastanza diffuso in sistemi *mainframe*

Modelli e problematiche di *file system* *Struttura – 5*



Modelli e problematiche di *file system*
Modalità di accesso – 1

- **Accesso sequenziale**
  - Viene trattato un gruppo di *byte* (un *record*) alla volta
  - Un puntatore indirizza il *record* corrente, e avanza a ogni lettura o scrittura
    - La lettura può avvenire in qualunque posizione del *file*, la quale però deve essere raggiunta sequenzialmente
      - Come con un nastro
    - La scrittura può avvenire solo in coda al *file* (*Append*)
  - Sul *file* si può operare solo sequenzialmente
    - Ogni nuova operazione fa ripartire il puntatore dall'inizio

Modelli e problematiche di *file system*
Modalità di accesso – 2

- **Accesso diretto**
  - Opera su *record* di dati in posizione arbitraria nel *file*
    - Posizione determinata rispetto alla base (*offset* = 0)
- **Accesso indicizzato**
  - Per ogni *file* una tabella di chiavi ordinate contenenti gli *offset* dei rispettivi *record* nel *file*
    - Informazione di navigazione non più nei *record* ma in una struttura a parte (principio delle base di dati)
  - Ricerca binaria della chiave e poi accesso diretto
  - Denominato **ISAM** (*indexed sequential access method*) da IBM
    - Consente accesso sia indicizzato che sequenziale

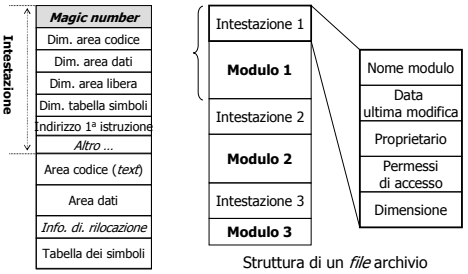
Modelli e problematiche di *file system*
Classificazione

UNIX → GNU/Linux

Windows

- Il FS può trattare diversi tipi di *file*
  - Classificazione distinta da quella dell'utente!
  - *File* normali (*regular*), sui quali l'utente può operare la propria classificazione
    - Contenuto ASCII (testo) o binario (eseguibile)
  - *File* catalogo (*directory*), con i quali il FS consente di descrivere l'organizzazione di gruppi di *file*
  - *File* speciali, con i quali il FS modella dispositivi orientati a carattere (p.es.: terminale) o a blocco (p.es.: disco)

Esempio 1
File binari in UNIX e GNU/Linux



Struttura di un *file* eseguibile

Struttura di un *file* archivio (**tar** : *tape archive*)

Modelli e problematiche di *file system*
Operazioni ammesse – 1

- **Creazione**
  - Inizialmente vuoto; inizializzazione attributi
- **Apertura**
  - Deve precedere l'uso; permette di predisporre le informazioni utili all'accesso
- **Cerca posizione** (*seek*)
  - Solo per accesso casuale
- **Cambia nome**
  - *rename* (può implicare spostamento nella struttura logica del FS)
- **Distruzione**
  - Rilascio della memoria occupata
- **Chiusura**
  - Rilascio delle strutture di controllo usate per l'accesso ed il salvataggio dei dati
- **Lettura**. **Scrittura**
  - *read, write, append*
- **Trova attributi**
  Modifica attributi
- Azioni più complesse (p.es.: copia) si ottengono tramite combinazione delle operazioni di base

Modelli e problematiche di *file system*
Operazioni ammesse – 2

- **Sessione d'uso di un *file***
  - Si può accedere in uso solo ad un *file* già aperto
  - Mediante l'apertura, il S/O predispone uno specifico strumento (*handle*) di accesso a quel *file*
  - Dopo l'uso il *file* dovrà essere chiuso
  - UNIX e GNU/Linux hanno una tabella dei *file* aperti a due livelli
    - Livello I: informazioni sul *file* comuni a più processi
      - Da *handle* verso attributi, posizione su disco, punto di R/W
    - Livello II: dati specifici del particolare processo
      - E puntatore alla voce corrispondente in tabella di livello I

Modelli e problematiche di *file system*
Esempio d'uso con chiamate di sistema

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc,
         char *argv[]){

  FILE *fp;
  char dato;

  if (argc != 2) {
    printf("Nome del file?");
    exit(1);
  }

  // continua …
```

1/2

```
  if ((fp = fopen(argv[1],"w"))
      == NULL){
    printf("File non aperto.\n");
    exit(1);
  }
  do {
    dato = getchar();
    if (EOF == putc(dato, fp)) {
      printf("Errore di lettura.\n");
    break;
    };
  } while (dato != 'c');
  fclose(fp);
}
```

2/2

Modelli e problematiche di *file system*
File mappati in memoria

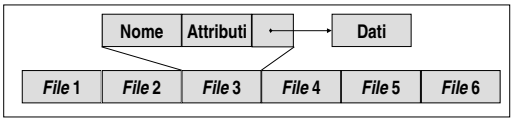
- Il S/O può mappare un *file* in memoria virtuale
  - Il *file* continua a risiedere in memoria secondaria
  - All'indirizzo di ogni suo dato corrisponde un indirizzo di memoria virtuale (base + *offset*)
    - Con memoria segmentata si ha {*file* = segmento} potendo così usare lo stesso *offset* per entrambi
  - Le operazioni su *file* avvengono in memoria principale
    - Chiamata di indirizzo → *page fault* → caricamento → operazione → salvataggio in memoria secondaria
  - A fine sessione le modifiche effettuate in memoria primaria vengono riportate in memoria secondaria
- Riduce gli accessi a disco ma comporta problemi con la condivisione e con i *file* di enorme dimensione

Modelli e problematiche di *file system*
Struttura della directory – 1

- Ogni FS usa *directory* (catalogo) o *folder* (cartella) per tener traccia dei suoi *file* regolari
- Le *directory* possono essere classificate, rispetto alla loro struttura, come
  - A livello singolo
  - A due livelli
  - A albero
  - A grafo aperto
  - A grafo generalizzato (con cicli)

Modelli e problematiche di *file system*
Struttura della directory – 2

- ***Directory* a livello singolo**
  - Tutti i *file* sono elencati su un'unica lista lineare, ciascuno con il proprio nome
    - I nomi devono pertanto essere unici
  - Semplice da capire e da realizzare
  - Gestione onerosa all'aumentare dei *file*



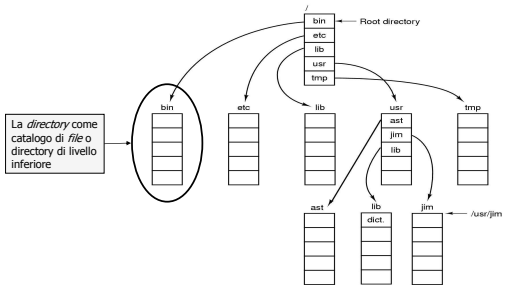
Modelli e problematiche di *file system*
Struttura della directory – 3

- ***Directory* a due livelli**
  - Una *Root Directory* contiene una *User File Directory* (UFD) per ciascun utente di sistema
  - L'utente registrato può vedere solo la propria UFD
    - Le UFD di altri solo se esplicitamente autorizzato
    - Buona soluzione per isolare utenti in sistemi multiprogrammati
  - *File* localizzati tramite percorso (*path name*)
  - I programmi di sistema possono essere copiati su tutte le UFD, oppure (meglio) posti in una *directory* di sistema condivisa ed ivi localizzati mediante cammini di ricerca predefiniti (*search path*)

Modelli e problematiche di *file system*
Struttura della directory – 4

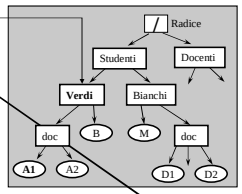
- ***Directory* ad albero**
  - Numero arbitrario di livelli
  - Il livello superiore viene detto radice (*root*)
  - Ogni *directory* può contenere *file* regolari o *directory* di livello inferiore
  - Ogni utente ha la sua *directory* corrente, che può essere cambiata con comandi di sistema
  - Se non si specifica il cammino (*path*), si assume la directory corrente
  - Il cammino può essere **assoluto** (espresso rispetto alla radice) o **relativo** (rispetto alla posizione corrente)

Modelli e problematiche di *file system*
Esempio di directory ad albero



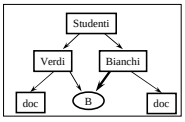
Modelli e problematiche di *file system*
Esempio (/ per UNIX/GNU/Linux, \ per Windows)

Livello corrente: *directory* **`Verdi`** = . (*dot*)
Livello superiore (*directory* padre) = ..
Livello inferiore (*directory* figlio) = ./ (*slash*)
Il *file* **A1** identificato come
`[./]doc/A1` (cammino relativo)
`/studenti/Verdi/doc/A1` (cammino assoluto)
Il *file* **D1** di un altro ramo (purché condiviso)
`../studenti/Bianchi/doc/D1` (relativo)
`/studenti/Bianchi/doc/D1` (assoluto)



Modelli e problematiche di *file system*
Struttura della directory – 5

- ***Directory* a grafo aperto** (e generalizzato)
  - L'albero diventa grafo, consentendo allo stesso *file* di appartenere simultaneamente a più *directory*
  - UNIX e GNU/Linux utilizzano collegamenti simbolici (***link***) tra il nome reale del *file* e la sua presenza virtuale
  - La forma generalizzata consente collegamenti ciclici (riferimenti circolari)
  - Un S/O potrebbe duplicare gli identificatori di accesso al *file* (*handle*) → nomi distinti
    - Questo però rende più difficile assicurare la coerenza del *file*



Modelli e problematiche di *file system*
Operazioni su directory (GNU/Linux)

Crea *directory*	`mkdir`	Create
Cancella *directory*	`rmdir`	Delete
Cambia nome a *directory*	`mv`	Rename
Apri, chiudi, leggi *directory*		Opendir, Closedir, Readdir
Crea collegamento a *file*	`ln`	Link
Rimuovi collegamento a *file*	`rm`	Unlink

Hard link : un puntatore al *file* originario viene inserito nella *directory* remota; questo crea 2 vie d'accesso distinte allo stesso *file*
Symbolic link : *file* speciale il cui contenuto è il cammino del *file* originario; questo mantiene 1 sola via d'accesso al *file*