Modelli e problematiche di file system
Parte 1 - Indice

1. Attributi
2. Struttura logica interna di *file*
3. Operazioni ammesse su *file*
4. Struttura logica interna di *directory*
5. Operazioni ammesse su *directory*

Modelli e problematiche di file system
Generalità - 1

- La maggior parte dell'informazione strutturata (i dati) ha durata, ambito e dimensione assai più ampi della vita delle applicazioni in memoria principale
  - 3 sono le esigenze più evidenti
    - Nessun limite di dimensione
    - Persistenza dei dati
    - Condivisione dei dati tra applicazioni distinte
- Il *file system* è la struttura dati progettata per soddisfare tali bisogni

Modelli e problematiche di file system
Generalità - 2

- Il termine *file* designa un gruppo di dati correlati, posti in memoria secondaria e trattati unitariamente
- Il termine *file system* (FS) designa la parte di S/O che si occupa dell'organizzazione, gestione, realizzazione ed accesso ai *file*
- La progettazione di FS incontra 2 problemi chiave
  - Come e cosa occorre offrire all'utente
    - Accesso a file, struttura logica e fisica, operazioni ammesse
  - Come ciò può essere implementato
    - Massima indipendenza dalla architettura fisica di supporto

Modelli e problematiche di file system
File

- Il *file* è un meccanismo di astrazione
  - Consente di salvare informazione su memoria secondaria per ritrovarvela in un momento successivo, senza conoscere struttura logica e fisica e funzionamento del supporto utilizzato
  - All'utente non interessa come ciò avviene
  - E' invece necessario per l'utente poter designare le unità di informazione mediante nomi unici e distinti
  - L'utente vede e tratta nomi di *file*
  - Le caratteristiche distintive di un file sono
    - Attributi (a partire dal nome)
    - Struttura interna
    - Operazioni ammesse

Modelli e problematiche di file system
Attributi - 1

- Nome
  - Stringa composta da 8 a 255 caratteri, inclusi numeri e caratteri speciali
  - Con una o più estensioni che possono designare il "tipo" di file come visto dall'utente
    - **MS-DOS** (base di Windows 95 e Windows 98)
      - Nomi da 1 a 8 caratteri, con al più 1 estensione da 1 a 3 caratteri, designante, senza distinzione tra maiuscolo e minuscolo (*case insensitive*)
    - **UNIX** (base di 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 estensione e tipo

Modelli e problematiche di file system
Attributi - 2

- Altri attributi significativi
  - Dimensione corrente
  - Data di creazione (può non essere mostrata)
  - Data di ultima modifica
    - Essenziale per indicare la "freschezza" del contenuto
  - Creatore e Possessore (possono essere 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

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*)

Flag

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
    - Dove il programma utente associa significato al contenuto informativo del *file*
  - Livello di **struttura logica**
    - Dove i dati grezzi (non interpretati) sono raggruppati dal S/O in strutture logiche per facilitarne il trattamento
  - Livello di **struttura fisica**
    - Dove il S/O mappa le strutture logiche sulle strutture fisiche che rappresentano la memoria secondaria disponibile (p.es.: 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

- A sequenza di *byte* (*byte stream*)
  - La strutturazione logica più rudimentale e flessibile
    - La scelta di UNIX (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

- A *record* di lunghezza e struttura interna 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 ed 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

- A *record* di lunghezza e struttura interna variabile
  - La struttura interna di ogni record è descritta ed identificata univocamente da una chiave (*key*)
  - Tutte le chiavi sono raccolte in una tabella a se stante, ordinata per chiave, contenente anche i puntatori all'inizio del record
  - L'accesso ai dati è per chiave
  - Di uso abbastanza diffuso in sistemi *mainframe*

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, ed avanza ad 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
  - Consente di operare su gruppi di dati (*record*) posti in posizione qualsiasi nel *file*
  - La posizione nel *file* è determinata rispetto alla sua base (*offset* = 0)
- Accesso indicizzato
  - Per ogni file, una tabella di chiavi ordinate contenenti gli *offset* dei rispettivi record nel *file*
  - Ricerca binaria della chiave e poi accesso diretto
  - Denominato **ISAM** (*indexed sequential access method*) perché consente accesso sia indicizzato che sequenziale

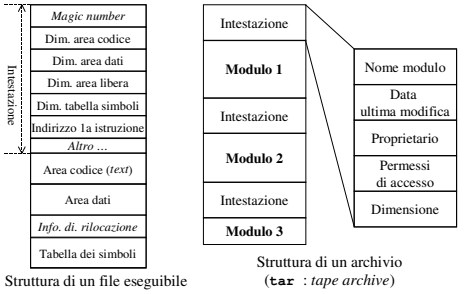
Modelli e problematiche di file system
Classificazione

UNIX/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 sua classificazione
    - Contenuto ASCII (testo) o binario (eseguibile)
  - *File* catalogo (*directory*), con i quali il FS descrive a suo uso l'organizzazione dei *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/Linux



Struttura di un file eseguibile

Struttura di un 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 date

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, chiamato *handle*, di accesso a quel *file*
  - Dopo l'uso, il file dovrà essere chiuso
  - UNIX/Linux ha una tabella dei file aperti a due livelli
    - Nel primo ci sono le informazioni del file comuni a tutti i processi
    - Nel secondo i dati specifici del particolare processo

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

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

void 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

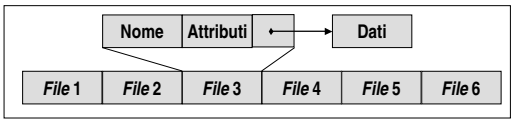
- Il S/O può mappare un *file* in memoria virtuale
  - Il file continua a risiedere in memoria secondaria
  - All'indirizzo di ogni suo dato corrisponde un indirizzo di memoria virtuale (base + *offset*)
    - Con memoria segmentata si ha {*file* = segmento} potendo così usare lo stesso *offset* per entrambi
  - Le operazioni avvengono in memoria principale
    - Chiamata di indirizzo → page fault → caricamento → operazione
  - A fine sessione le modifiche apportate al *file* in memoria primaria vengono riportate sul *file* in memoria secondaria
- Riduce gli accessi a disco, ma ha 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, per struttura, come
  - A livello singolo
  - A due livelli
  - Ad albero
  - A grafo aperto
  - A grafo generalizzato

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 essere unici
  - Semplice da capire ed implementare
  - Gestione onerosa all'aumentare dei *file*



Modelli e problematiche di file system
Struttura della directory - 3

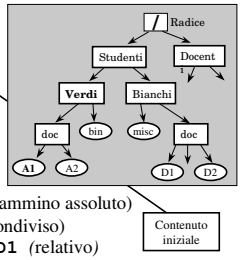
- Directory a due livelli
  - Una *Root Directory* contenente una *User File Directory* (UFD) per utente
  - L'utente deve registrarsi e può vedere solo la propria UFD
    - Se esplicitamente consentito possono vedere anche le UFD di altri
    - Buona soluzione per isolare utenti di sistemi multiprogrammati
  - I file vengono localizzati indicandone il percorso (*path name*)
  - I programmi di sistema possono essere copiati su tutte le UFD, altrimenti (meglio) posti in una *directory* di sistema condivisa e localizzati mediante cammini di ricerca predefiniti (*search path*)

Modelli e problematiche di file system
Struttura della directory - 4

- Directory ad albero
  - Consente un 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)

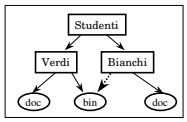
Modelli e problematiche di file system
Esempio 2 (/ per UNIX/Linux, \ per Windows)

Livello corrente:
directory **`Verdi`** = `.` (*dot*)
Livello superiore
(*directory* padre) = `..`
Livello inferiore
(*directory* figlio) = *./* (*slash*)
Il file **A1** può essere identificato
`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 - 6

- *Directory* a grafo aperto (e generalizzato)
  - Potenzia la struttura ad albero consentendo ad un *file* di essere simultaneamente presente in più *directory*
  - UNIX/Linux utilizzano collegamenti simbolici (*link*) tra il nome reale e la sua immagine
  - La forma generalizzata consente collegamenti ciclici (riferimenti circolari)
  - Altri S/O duplicano gli identificatori di accesso al *file* (*handle*)
    - Rende difficile assicurare la coerenza del *file*



Modelli e problematiche di file system
Operazioni UNIX/Linux su directory

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 : nuovo nome di *file* inserito in *directory*;
2 identità distinte per lo stesso *file*, nessuna copia
Symbolic link : nuovo nome, nessuna nuova identità