Lezione 22 [06/12/22]

Capitolo 11: Interfaccia del File-System

- Concetto di file
- Metodi di accesso
- Struttura della directory e del disco
- Montaggio di un file system
- Condivisione di file
- Protezione

Concetto di file

Il file system è l'aspetto più visibile di un sistema operativo

Esso fornisce il meccanismo per la memorizzazione in linea di dati e programmi appartenenti al S.O. e a tutti gli utenti del sistema elaborativo

Il file system consiste di due parti distinte:

- Un insieme di **file**, ciascuno dei quali contiene i dati
- Una **struttura della directory**, che organizza tutti i file nel sistema e fornisce le informazioni relative

Il S.O. fornisce un'astrazione delle caratteristiche fisiche dei propri dispositivi di memoria definendo un unità di memorizzazione logica vista dall'utente: il **file**

Un **file** è un insieme di informazioni correlate e registrate su memoria secondaria, cui è stato assegnato un nome

Dal punto di vista utente, un file è la più piccola porzione di memoria logica secondaria: i dati si possono scrivere in memoria secondaria solo attraverso un file

Il sistema operativo provvede ad allocare i file su dispositivi di memorizzazione e ad eseguire le operazioni sui file richieste dall'utente

Tipi di file:

- Quelli che rappresentano **dati**
  - Numerici
  - Alfabetici
  - Binari
- Quelli che rappresentano **programmi**
  - Sorgente
  - Oggetto

Struttura dei file

Le informazioni contenute in un file sono definite dal suo creatore e possono essere di molti tipi

Un file ha una **struttura** definita secondo il tipo:

- Un file di **testo** è formato da una sequenza di caratteri organizzati in righe, ed eventualmente pagine
- Un file **sorgente** è formato da una sequenza di funzioni, ciascuna delle quali a sua volta organizzata in dichiarazioni seguite da istruzioni eseguibili
- Un file **eseguibile** consiste in una serie di sezioni di codice che il loader può caricare in memoria ed eseguire

UNIX considera ciascun file come una sequenza di byte, lasciando ai programmi applicativi il compito di contenere il codice per interpretare la struttura di un file di input

In generale un file potrà essere:

- Vuoto
- Visto come una sequenza di parole o byte
- Visto come una sequenza di record logici

Attributi dei file

Un file ha attributi che possono variare secondo il sistema operativo, ma che tipicamente comprendono i seguenti:

- **Nome** - il nome simbolico del file è l'unica informazione in forma umanamente leggibile
- **Identificatore (file descriptor)** - si tratta di un'etichetta unica, di solito un numero, che identifica il file all'interno del file system; è il nome impiegato dal sistema per il file
- **Tipo** - questa informazione è necessaria ai sistemi che gestiscono tipi di file diversi
- **Locazione** - si tratta di un puntatore al dispositivo e alla locazione del file in tale dispositivo
- **Dimensione** - si tratta della dimensione corrente del file (in byte, parole o blocchi) ed eventualmente della massima dimensione consentita
- **Protezione** - le informazioni di controllo degli accessi controllano chi può leggere, scrivere o eseguire il file
- **Ora, data e identificazione dell'utente** - queste informazioni possono essere relative alla creazione, l'ultima modifica e l'ultimo uso. Questi dati possono essere utili ai fini della protezione, della sicurezza e del monitoraggio del suo utilizzo

Le informazioni sui file sono conservate nella struttura della directory, che risiede a sua volta in memoria secondaria

Di solito un elemento di directory consiste di un nome di file e di un identificatore unico, che a sua volta individua gli altri attributi del file

Operazioni sui file

Un file è un **tipo di dato astratto**: per definirlo è necessario considerare le operazioni che si possono eseguire su di esso

Il S.O. offre chiamate di sistema per operare sui file: ad es. per creare, scrivere, leggere, spostare, cancellare e troncare un file, …

Creazione di un file - necessita di due passaggi:

- Trovare lo spazio nel file system

- Creare un nuovo elemento nella directory in cui registrare il nome del file, la sua posizione ed altre informazioni

Scrittura di un file - una chiamata al sistema specificherà il nome del file e le informazioni che si vogliono scrivere

- Dato il nome del file, il sistema cerca la sua posizione nella directory mantenendo un puntatore alla posizione del file in cui deve avvenire la prossima scrittura
    - Il puntatore si deve aggiornare ogniqualvolta si esegue una scrittura

Lettura di un file - una chiamata al sistema specificherà il nome del file e la posizione nella memoria dove leggere

- Dato il nome del file, il sistema cerca la sua posizione nella directory, mantenendo un puntatore alla posizione del file in cui deve avvenire la prossima lettura
    - Una volta completata la lettura si aggiorna il puntatore

Di solito un processo o legge o scrive in un file, la posizione corrente è mantenuta come un **puntatore alla posizione corrente del file** specifico del processo. Sia le operazioni di lettura sia quelle di scrittura adoperano lo stesso puntatore

Riposizionamento in un file (seek) - si ricerca l'elemento appropriato nella directory si assegna un nuovo valore al puntatore alla posizione corrente nel file

- Non richiede nessuna operazione di I/O

Cancellazione di un file - si cerca l'elemento della directory associato al file designato, si rilascia lo spazio associato al file e si elimina l'elemento della directory

Troncamento di un file - consente di mantenere immutati gli altri attributi del file pur azzerando la lunghezza del file e rilasciando lo spazio occupato

Altre operazioni di base comprendono:

- L'aggiunta di nuove informazioni alla fine di un file esistente (**append)**
- La ridenominazione di un file esistente (**rename**)

Esistono inoltre operazioni che consentono a un utente di leggere e impostare i vari attributi di un file (come ad es. impostare il proprietario)

La maggior parte delle operazioni sopra citate richiede una ricerca nella directory dell'elemento associato al file specificato

Per evitare questa continua ricerca, molti sistemi richiedono l'impiego di una chiamata a sistema **open** prima che un file venga utilizzato

Il S.O. mantiene una tabella contente informazioni riguardanti tutti i file aperti **(tabella dei file aperti)**

Quando si richiede un'operazione su un file, questo viene individuato tramite un indice in tale tabella, in questo modo si evita qualsiasi ricerca

Quando il file non è più attivamente usato viene **chiuso** dal processo, e il sistema operativo rimuove l'elemento a esso associato dalla tabella dei file aperti

Le chiamate di sistema che lavorano su file chiusi invece che aperti sono **create** e **delete**

Un'operazione di **open** inserirà una nuova entry nella tabella dei file aperti, una di **close** ne rimuoverà una

- L’operazione open riceve il nome del file
- Lo cerca nella directory
- Copia l’elemento della directory a esso associato nella tabella dei file aperti
- Può accettare anche informazioni sui modi d’accesso (creazione, sola lettura, lettura e scrittura, ecc.)
- Si controllano i permessi relativi al file e se la modalità d’accesso richiesta è consentita, si apre il file per il processo

La realizzazione di **open** e **close** in un ambiente multiutente è complicata dal fatto che più utenti possono aprire un file contemporaneamente

Il S.O. introduce due livelli di tabelle interne:

- Una per ciascun processo:
  - Contiene i riferimenti a tutti i file aperti da quel processo
  - Vengono memorizzate le informazioni sull’uso del file da parte del processo (puntatore alla posizione corrente per ciascun file, informazioni sui diritti d’accesso ai file, …)
- Una di sistema (*a cui puntano le tabelle dei processi*):
  - Contiene le informazioni indipendenti dai processi
    - Posizione del file nei dischi
    - Date degli accessi
    - Dimensione dei file

Quando un file è stato aperto da un processo, la tabella dei file aperti del sistema contiene un elemento relativo al file

- Una open da parte di un altro processo comporta solamente l’aggiunta di un nuovo elemento nella tabella dei file aperti associata al processo, che punta al corrispondente elemento nella tabella di sistema

La tabella dei file aperti ha anche un **contatore delle aperture** associato ad ogni file, indicante il numero di processi che hanno aperto quel file

- Ogni close decrementa questo contatore

Quando raggiunge il valore zero il file non è più in uso e si elimina l’elemento corrispondente dalla tabella dei file aperti

Informazioni associate ad un file aperto

Riassumendo, a ciascun file aperto sono associate le diverse seguenti informazioni:

- **Puntatore al file:** ultima posizione di lettura e scrittura sotto forma di un puntatore alla posizione corrente nel file
  - Unico per ogni processo che opera sul file
- **Contatore dei file aperti:** tiene traccia del numero di open e close
  - Raggiunge il valore zero dopo l’ultima chiusura indicando che il sistema può rimuovere l’elemento dalla tabella
- **Posizione nel disco del file:** l’informazione necessaria per localizzare il file
  - È mantenuta nella memoria, per evitare di doverla prelevare dal disco ad ogni operazione
- **Diritto di accesso:** ciascun processo apre un file in una delle modalità di accesso

    - Quest’informazione è contenuta nella tabella del processo in modo che il S.O. possa permettere o negare le successive richieste di I/O

Lock ai file

Alcuni S.O. offrono la possibilità di applicare **lock** a un file aperto (o parti di esso)

Ciò serve a garantire la protezione del file dall’accesso concorrente di altri processi

L’utilità dei lock dei file emerge nel caso di file condivisi da diversi processi

- Un file di log, per esempio, può subire modifiche da parte di molti processi nel sistema

I lock dei file sono simili ai lock di lettura-scrittura

Un **lock condiviso** è simile ai lock di lettura

- Consente a più processi di appropriarsene

Un **lock esclusivo** è invece analogo ai lock di scrittura

- Un solo processo per volta può acquisire questo tipo di lock

Inoltre i S.O. forniscono meccanismi di lock dei **file obbligatori (mandatory**) oppure **consultivi (advisory)**

- Se un lock è obbligatorio, il S.O. impedirà a qualunque altro processo di accedere al file interessato una volta che il lock è stato acquisito
- Se un lock è consultivo, il S.O. non impedirà l’accesso accesso ai file, tuttavia, per potervi accedere il programma utilizzante il file dovrà essere scritto in modo tale da acquisire esplicitamente il lock

In linea generale se il lock è obbligatorio il S.O. assicura l’integrità dei dati soggetti a lock; se il lock è consultivo, è compito dei programmatori garantire la corretta acquisizione e cessazione dei lock

I sistemi operativi Windows adottano i lock obbligatori, mentre i sistemi UNIX impiegano i lock consultivi

Tipi di file

Il file system e in generale il S.O. devono saper riconoscere e gestire i diversi tipi di file

La maggior parte dei sistemi operativi permette agli utenti di specificare i nomi dei file come sequenze di caratteri seguite da un punto e concluse da un’estensione formata da caratteri aggiuntivi

- **nome.estensione**

Il sistema (come le varie applicazioni) usa l’estensione per stabilire il **tipo** del file e le operazioni che si possono eseguire su tale file

Comuni tipi di file:

Tipo di file	Estensione usuale	Funzione
Eseguibile	exe, com, bin, o nessuna	Programma eseguibile, in linguaggio macchina
Oggetto	obj, o	Compilato, in linguaggio di macchina, non linkato
Codice sorgente	c, cc, java, perl, asm	Codice sorgente in vari linguaggi di programmazione
Batch	bat, sh	Comandi per l'interprete dei comandi
Markup	xml, html, tex	Dati testuali, documenti
Word processor	xml, rtf, docx	Vari formati di word processor
Libreria	lib, a, so, dll	Librerie di procedure per la programmzione
Stampa o visualizzazione	gif, pdf, jpg	File ASCII o binari in formato per la stampa o la visualizzazione
Archivio	rar, zip, tar	File contenenti più file tra loro correlati, talvolta compressi, per archiviazione o memorizzazione
Multimediali	mpeg, mov, mp3, mp4, avi	File binari contenenti informazioni audio o A/V

Cosa succedeva in DOS

In MS-DOS i file possono essere suddivisi in:

- **File eseguibili** che contengono programmi eseguibili sotto il controllo del DOS
- **File dati** che contengono informazioni utilizzabili dai file eseguibili

I file eseguibili possono avere tre possibili estensioni:

- COM o EXE - se contengono programmi eseguibili
- BAT - se contengono comandi DOS

I file possono avere una estensione qualsiasi (anche mancante) e contengono informazioni non direttamente utilizzabili dall'elaboratore Il loro utilizzo è legato alla esecuzione di un programma capace di interpretarne il contenuto

In alcuni casi il programma che utilizza il file dati imposta o richiede una particolare estensione

I file dati sono creati da file eseguibili

Cosa succede in UNIX

In UNIX, un file è sequenza di byte, ed è eseguito solo se ha il formato appropriato. Nella figura vediamo un semplice file binario eseguibile

Il file ha cinque sezioni: **intestazione, testo, dati, bit di rilocazione, tabella dei simboli**

L'intestazione inizia con una parte detta **magic number** che identifica un file come file eseguibile (per prevenire l'esecuzione accidentale di un file non in questo formato)

Seguono alcuni interi di 16 bit che danno la dimensione delle varie parti del file, l'indirizzo per la partenza dell'esecuzione e qualche bit di flag

Dopo troviamo il testo e i dati del programma: queste parti sono caricate in memoria e rilocate usando il bit di rilocazione. La tabella dei simboli è usata per il debugging

Metodi di accesso

Esistono molti metodi per accedere alle informazioni dei file

n = numero di blocco relativo

Accesso sequenziale:

- read_next()
- write_next()
- reset()
- no_read_after_last write - rewrite

Accesso diretto:

- read(n)
- write(n)
- position_file(n)
- read_next()
- write_next()
- rewrite(n)

Altri metodi di accesso

Accesso sequenziale

Il più semplice metodo d'accesso è **l'accesso sequenziale**

Le informazioni del file si elaborano ordinatamente, un record dopo l'altro

Risulta essere il metodo più comune ed utilizzato

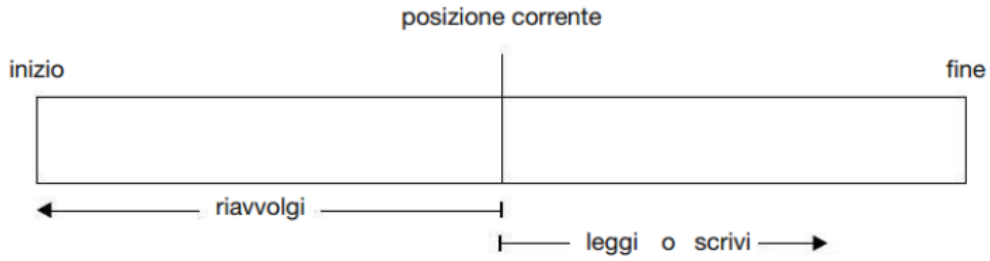
Le operazioni di lettura e scrittura sono definite come segue:

- **read_next()** - legge la prossima porzione di file e fa avanzare automaticamente il puntatore del file che tiene traccia della locazione di I/O
- **write_next()** - fa un'aggiunta in coda al file e avanza fino alla fine delle informazioni appena scritte, che costituisce la nuova fine del file

Tale file si può reimpostare sull'inizio ed è possibile andare avanti o indietro di n record, con n intero (in alcuni casi solo per $n = 1$)

L'accesso sequenziale utilizza per il file il modello del nastro magnetico; funziona nei dispositivi ad accesso sequenziale così come nei dispostivi ad accesso diretto

File ad accesso sequenziale:



Accesso diretto

Nell'**accesso diretto** (o **accesso relativo**) un file è formato da elementi logici **(record)** di lunghezza fissa

- Ciò consente ai programmi di leggere e scrivere rapidamente tali elementi senza un ordine particolare

Il metodo ad accesso diretto si fonda su un modello di file che si rifà al disco: i dischi permettono infatti l'accesso diretto a ogni blocco del file

Il file si considera come una sequenza numerata di blocchi o record

- Si può per esempio leggere il blocco 14, quindi il blocco 53 e poi scrivere il blocco 7

Non esistono restrizioni all'ordine di lettura o scrittura di un file ad accesso diretto

I file ad accesso diretto sono molti utili quando è necessario accedere immediatamente a grandi quantità di informazioni (spesso le basi di dati sono di questo tipo)

Per il metodo ad accesso diretto si devono modificare le operazioni sui file per inserire il numero del blocco, si avranno quindi:

- **read(n)**
- **write(n)**

Dove n è il numero del blocco

Un metodo alternativo prevede di mantenere read_next() e write_next(), come nell'accesso sequenziale, e di aggiungere un'operazione **position_file(n)**, dove n è il numero del blocco

- Quindi un'operazione read(n) corrisponde a:
  - position_file(n), seguito da
  - read_next()

Il numero del blocco fornito dall'utente al sistema operativo è normalmente un **numero di blocco relativo**

Si tratta di un indice relativo all'inizio del file, quindi il primo blocco relativo del file è 0, anche se **l'indirizzo assoluto** nel disco del blocco può essere 14703

L’uso dei numeri di blocco relativi permette al sistema di decidere dove posizionare il file e aiuta a impedire che l’utente acceda a porzioni del file system riservate

Non tutti i sistemi operativi gestiscono ambedue i tipi di accesso

Alcuni permettono solo l’accesso sequenziale, altri solo quello diretto

Alcuni sistemi richiedono che si definisca il tipo d’accesso al file al momento della sua creazione

- A tale file si può accedere soltanto nel modo definito

Tuttavia si può facilmente simulare l’accesso sequenziale a un file ad accesso diretto mantenendo una variabile *cp* che definisce la posizione corrente

Tale operazione risulta tuttavia inefficiente e macchinosa

Simulazione dell’accesso sequenziale su un file ad accesso diretto:

Accesso sequenziale	Realizzazione nel caso di accesso diretto
`reset`	`cp = 0;`
`read_next()`	`read cp;` `cp = cp + 1;`
`write_next()`	`write cp;` `cp = cp + 1;`

Altri metodi d’accesso

Sulla base di un metodo d’accesso diretto se ne possono costruire altri, che implicano generalmente la costruzione di un indice per i file

L’indice contiene puntatori ai vari blocchi; per trovare un elemento del file occorre prima cercare nell’indice, per poi usare il puntatore per accedere direttamente al file e trovare l’elemento desiderato

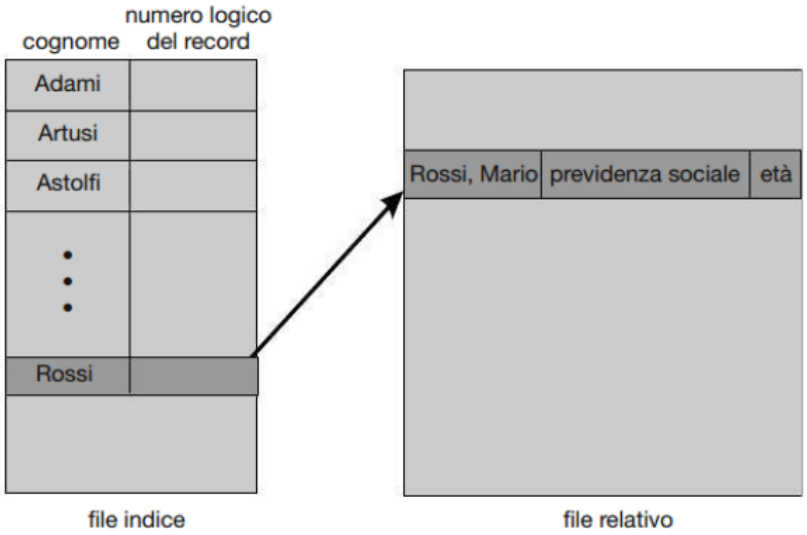
Questa struttura permette di compiere ricerche in file molto grandi limitando il numero di operazioni di I/O

Nel caso di file molto lunghi, lo stesso file indice può diventare troppo lungo perché sia tenuto in memoria

Una soluzione a questo problema è data dalla creazione di un indice per il file indice

- Il file indice principale contiene puntatori ai file indice secondari, che puntano agli effettivi elementi di dati

Esempio di indice e relativo file:



Il metodo ad accesso sequenziale indicizzato di IBM (Index Sequential Access Method, ISAM) per esempio, usa un piccolo indice principale che punta ai blocchi del disco di un indice secondario, mentre i blocchi dell'indice secondario puntano ai blocchi del file effettivo

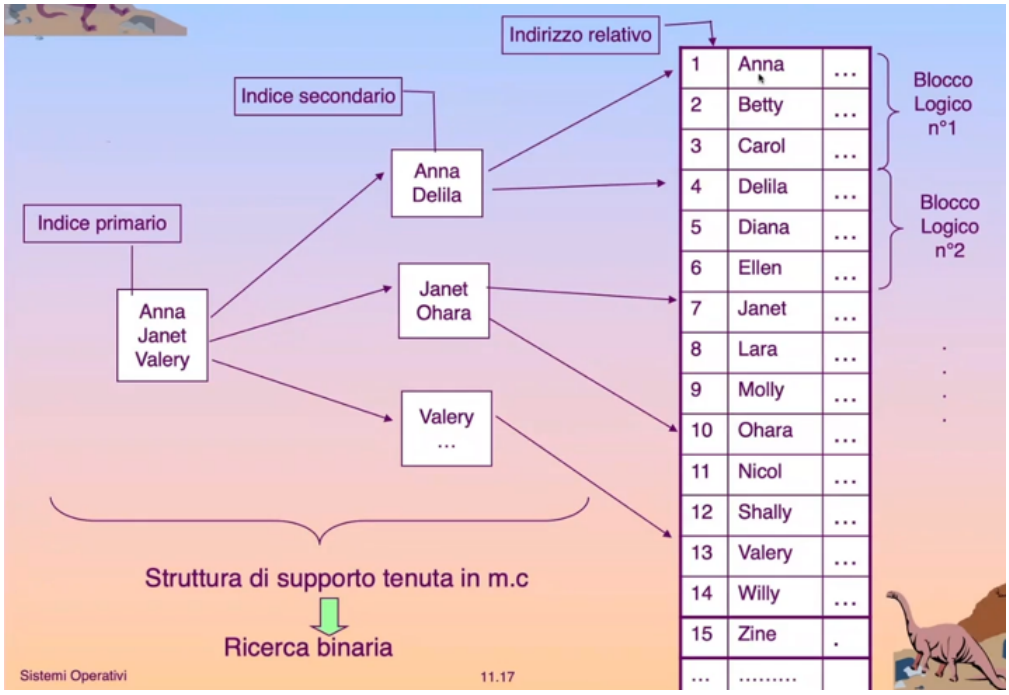
Il file è ordinato rispetto a una chiave definita

Per trovare un elemento:

- Si fa una ricerca binaria nell'indice principale che fornisce il numero del blocco dell'indice secondario
- Il blocco ottenuto viene letto e sottoposto a una seconda ricerca binaria per individuare il blocco dell'elemento richiesto
- Si fa infine una ricerca sequenziale sul blocco fino a localizzare l'elemento richiesto tramite la sua chiave

Si può quindi localizzare ogni elemento tramite la sua chiave con al massimo due letture ad accesso diretto

Metodo ISAM:



Struttura della directory

I dischi (sui quali risiedono i file) possono essere **partizionati** e ogni partizione può contenere un file system

La suddivisione in partizioni è utile anche per:

- Limitare la dimensione dei singoli file system
- Mettere sullo stesso dispositivo diversi tipi di file system
- Liberare per altri scopi una parte del dispositivo
    - Ad es. per lo spazio di swap e dello spazio di disco non formattato (raw)

Ogni unità contenente un file system è generalmente nota come **volume**

Ogni volume può essere pensato come un disco virtuale, inoltre ognuno può contenere diversi sistemi operativi, permettendo al sistema di avviare ed eseguire più di un sistema operativo

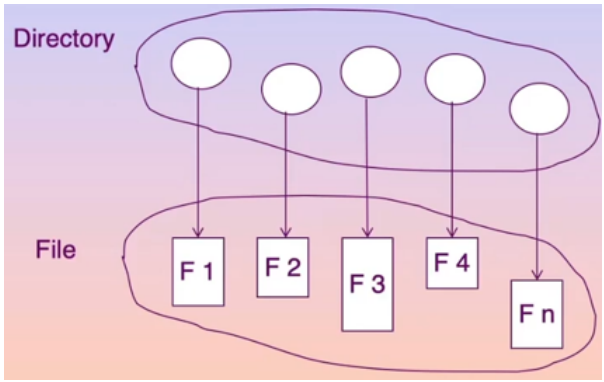
Ciascun volume contente un file system deve anche avere in sé le informazioni sui file presenti nel sistema

- Tali informazioni risiedono in una **directory del dispositivo** o **indice del volume**

La directory del dispositivo (in breve **directory**) registra informazioni, quali nome, posizione, dimensione e tipo, di tutti i file del volume

Il file sono di solito raggruppati in base alle loro caratteristiche o secondo i più vari criteri di comodità

Le directory consentono di ordinare il contenuto del disco secondo la necessità dello stesso utente



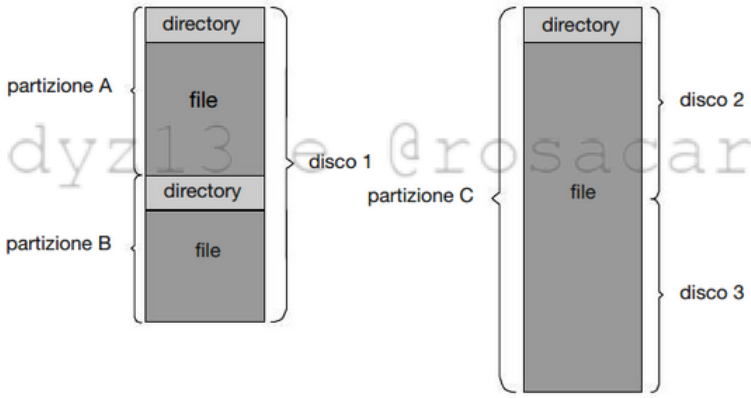
Sia la struttura della directory che i file risiedono sul disco

Una tipica organizzazione di file system

Il file system viene suddiviso in aree separate dette partizioni, ognuna delle quali contiene file e directory

Tipicamente ciascun disco contiene almeno una partizione

Tipica organizzazione di un file system:



Informazioni nella directory

Le informazioni sui file di ciascuna partizione sono mantenute negli elementi della **directory del dispositivo** (device directory) o **tabella dei contenuti del volume**

Questa registra le informazioni di tutti i file della partizione:

- Nome
- Tipo
- Indirizzo
- Lunghezza corrente
- Lunghezza massima
- Data di ultimo accesso
- Data di ultima modifica
- ID del proprietario
- Informazioni di protezione

Operazioni che si possono eseguire su una directory

La directory può essere considerata come una tabella di simboli che traduce i nomi dei file negli elementi in essa contenuti

Può essere organizzata in diversi modi per permettere operazioni diverse

Operazioni che si possono eseguire su una directory sono:

- Ricerca di un file
- Creazione di un file
- Cancellazione di un file
- Elencazione di una directory
- Ridenominazione di un file
- Attraversamento del file system
  - Si potrebbe voler accedere a ogni directory e a ciascun file contenuto in una directory. Per motivi di affidabilità, è opportuno salvare il contenuto e la struttura dell'intero file system a intervalli regolari, effettuando quindi copie di backup

Vantaggi dell'organizzazione logica di directory

Efficienza - un file viene localizzato velocemente

Nomi - conveniente uso per gli utenti

- Due utenti possono assegnare lo stesso nome a file differenti
- Lo stesso file può essere raggiungibile con nomi diversi

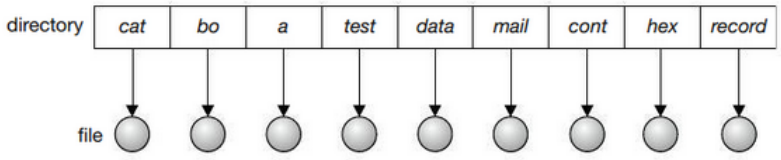
Gruppo - gruppi logici di file in base alle proprietà (es, tutti i programma Java, tutti i giochi, …)

Directory a singolo livello

La struttura di una directory a un livello è il tipo di struttura più semplice

Tutti i file sono contenuti nella stessa directory

Directory a livello singolo:



Problemi se aumentano il numero di file e utenti

- Diventa difficile ricordare i nomi dei file e quindi trovarli

Problema dei nomi (file con nomi unici)

- Non è possibile avere file con lo stesso nome all'interno della stessa directory

Problema dei gruppi (nessuna distinzione di proprietà)

Directory a due livelli

Directory separate per ogni utente, dette **UFD (User File Directory)**

Ogni utente dispone quindi della propria UFD

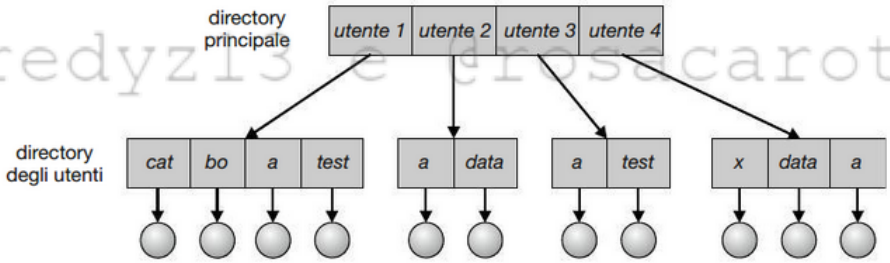
Tutte le UFD hanno una struttura simile e contengono solo i file del proprietario

Appena un utente effettua un login la ricerca viene fatta nella **directory principale (master file directory, MFD)** del sistema

La directory principale viene indicizzata con il nome dell'utente o il numero di account e ogni suo elemento punta alla relativa directory utente

- Quando un utente fa un riferimento a un file particolare, il sistema operativo esegue la ricerca solo nella directory di quell'utente
  - In questo modo gli utenti possono avere file con lo stesso nome dei file di altri utenti (quindi purché i nomi siano differenti all'interno della stessa directory)
- Tutte le operazioni risultano quindi più veloci
  - Per creare un file il S.O. controlla la presenza di nomi uguali sono nella stessa directory
  - Per cancellare i file la ricerca è limitata alla directory locale dell'utente
    - Di conseguenza non si può cancellare per errore il file di un altro utente

Struttura della directory a due livelli:



Tuttavia anche questa struttura presenta dei problemi

- A volte l'isolamento dei file utenti può essere un problema se gli utenti vogliono cooperare e accedere ai rispettivi file
  - In alcuni sistemi ciò non è possibile

In altri casi invece ciò è possibile tramite accesso autorizzato

- Si cerca il file a partire dalla directory principale tramite un percorso, quindi specificando un nome utente e un nome di file
  - Quindi un nome utente e un nome di file definiscono un **nome di percorso (path name)**

Ogni file del sistema ha un nome di percorso

Directory con struttura ad albero

La struttura delle directory a due livelli può essere ulteriormente generalizzata creando la **struttura ad albero**

- Essa permette agli utenti di creare proprie sottodirectory e di organizzare i file di conseguenza

L’albero ha una radice **(root directory)** e ogni file del sistema ha un unico nome di percorso

Una directory, o una sottodirectory, contiene un insieme di file o sottodirectory

- Le directory sono semplicemente file, trattati però in modo speciale
- Esse hanno tutte lo stesso formato interno
- *La distinzione tra file e directory è data da un bit di ogni elemento della directory*
    - Per creare e cancellare le directory si adoperano speciali chiamate di sistema

Ogni utente dispone di una **directory corrente** che dovrebbe contenere la maggior parte dei file di interesse corrente per il processo

Quando si fa un riferimento a un file, si esegue una ricerca nella directory corrente

- Se non si trova in tale directory, l’utente deve specificare un nome di percorso oppure cambiare la directory corrente (facendo diventare tale la directory contenente il file desiderato)

Quindi l’utente può cambiare la propria directory corrente ogni volta che lo desidera

Ciò avviene tramite la chiamata di sistema **change_directory()**

La directory corrente iniziale di un utente è stabilita quando viene lanciato un job dall’utente oppure quando inizia una sessione di lavoro

I nomi di percorso possono essere di due tipi:

- **Nomi di percorso assoluti**
    - Comincia dalla radice dell’albero di directory lungo il percorso
- **Nomi di percorso relativi**
    - Definisce un percorso che parte dalla directory corrente

Ad es. se la directory corrente risulta essere *root/spell/mail*

- Il nome di percorso relativo *prt/first* si riferisce allo stesso file indicato dal percorso assoluto *root/spell/mail/prt/first*

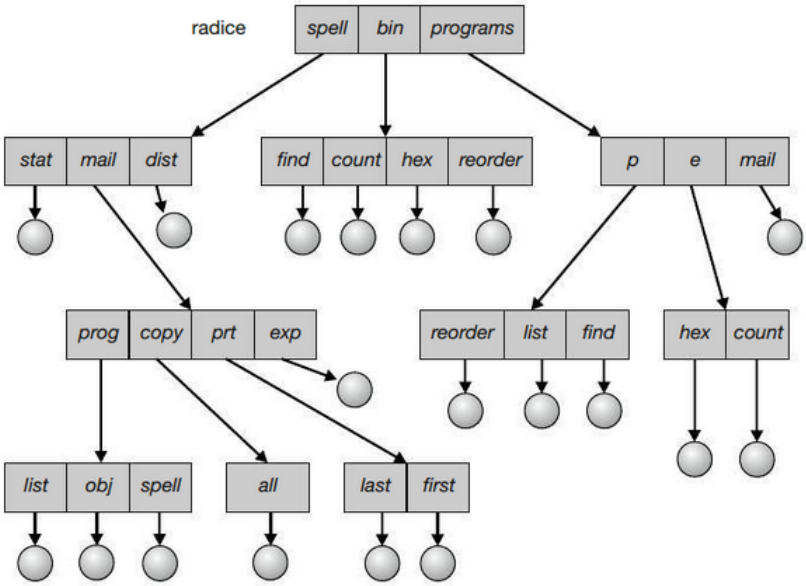
Risulta inoltre più semplice per gli utenti accedere alle directory di altri utenti tramite l’utilizzo dei percorsi

Alcuni sistemi impediscono di cancellare una directory a meno che essa non sia vuota

- Quindi è necessario cancellare ogni file di ogni sottodirectory per eliminare la directory principale

Altri sistemi forniscono invece dei comandi per una cancellazione ricorsiva, cancellando così automaticamente tutte le sottodirectory

Directory con struttura ad albero:



Directory con struttura ad albero (Unix)

Utenti diversi possono creare sottodirectory e file direttamente dalla directory corrente, esempio:

`mdkdir <dir-name>`

Se la directory corrente è */mail* e si esegue `mkdir count`:

- Cancellando *mail* viene rimosso anche il sottoalbero che contiene

Il funzionamento è quello precedentemente descritto:

- Ogni utente dispone di una directory corrente che contiene la maggior parte di dei file di interesse corrente
- Se si vuole un file che non è nella directory corrente bisogna o indicare il nome di percorso completo (pathname) oppure cambiare directory corrente, facendo diventare tale la directory che contiene il file desiderato

Cancellazione di un file: `rm <file-name>`

Cancellazione di una directory: `rmdir <dir-name>`

Directory con struttura a grafo aciclico

Permette alle directory di avere **sottodirectory** e **file condivisi**

Si pensi a una situazione dove ognuno vuole accedere agli stessi file ma dalle proprie directory

- In quel caso la sottodirectory deve essere **condivisa**

Un grafo aciclico (cioè senza cicli) permette alle directory di avere sottodirectory e file condivisi

Essendo il file condiviso esiste una sola copia dello stesso (quindi le informazioni non sono duplicate)

- Tutte le modifiche applicate dai vari utenti sono immediatamente visibili

Il metodo che viene utilizzato generalmente per permettere la condivisione è quello della creazione di un nuovo **elemento di directory**, chiamato anche **collegamento (link)**

- Il link è un puntatore a un altro file o un'altra directory
- Si può realizzare ad es. come un nome di percorso assoluto o relativo

Se l'elemento cercato è contrassegnato come collegamento, si **risolve** il collegamento usando il nome di percorso per localizzare il file reale

I collegamenti vengono facilmente identificati tramite il loro formato

Durante l'attraversamento delle directory il S.O. ignora questi collegamenti, preservando così la struttura aciclica

Un altro metodo comune per realizzare i file condivisi prevede invece la duplicazione di tutte le informazioni relative ai file in entrambe le directory che lo condividono

- I due elementi delle directory sono identici
- Sorge il problema di mantenere la coerenza se un file viene modificato

Nell struttura di directory a grafo aciclico si presentano però diversi problemi:

- Un file può avere più pathname assoluti **(aliasing)**
  - Quindi diversi nomi possono riferirsi allo stesso file
  - Ciò causa problemi nella ricerca e nei backup dati

La cancellazione causa molti problemi:

- Se un file condiviso viene cancellato riallocando lo spazio che occupava, potrebbero esistere puntatori ad un file che ormai non esiste più **(puntatori sospesi - dangling pointer)**
- Se i puntatori contengono indirizzi effettivi del disco e lo spazio viene poi riutilizzato per altri file, i puntatori potrebbero puntare nel mezzo di questi altri file (i collegamenti risulterebbero poi scorretti)

La cancellazione di un collegamento non influisce quindi sul file originale, poiché si rimuove solo il collegamento, ma causa varie incoerenze

In UNIX, quando si cancella un file, i collegamenti simbolici restano, è l'utente che deve rendersi conto che il file originale è scomparso o è stato restituito (situazione analoga in Windows)

Altre soluzioni prevedono la conservazione del file finché non sono stati cancellati tutti i collegamenti ad esso

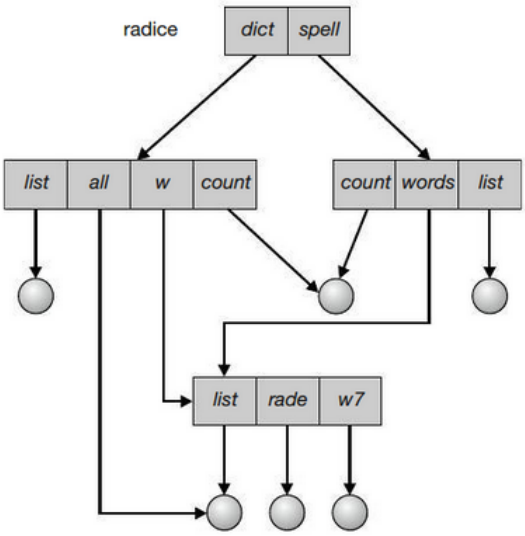
- Ciò richiede un meccanismo che permetta di determinare che l'ultimo riferimento è stato cancellato
- Si può tenere una lista di tutti i riferimenti a un file con un contatore di riferimenti
  - È possibile rimuovere il file se il contatore raggiunge valore 0 (cioè non esistono più collegamenti al file)

UNIX usa questo metodo per i collegamenti non simbolici, o **collegamenti effettivi (hard link)**

- Il contatore dei riferimenti è tenuto nel **blocco di controllo del file** o **inode**

- *L’hard link è la copia speculare del file* e presenta lo stesso numero inode del file originale

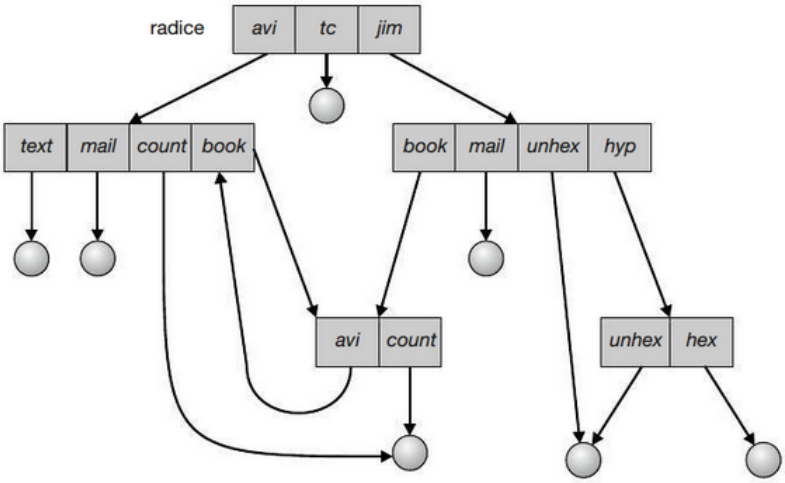
Struttura della directory a grafo aciclico:



Directory con struttura a grafo generale

Quando si aggiungono dei collegamenti a una directory con struttura ad albero, tale struttura si trasforma in una semplice **struttura a grafo**

Directory a grafo generale:



La presenza di cicli causa problemi relativi all’attraversamento

Per evitare problemi dovuti ai cicli esistono vari modi:

- Consentire solo link ai file e non alle sottodirectory
- **Garbage collection** (attraversamento del grafo per eliminare tutto ciò che non è più accessibile, cioè quando il contatore dei riferimenti è zero)

- Ogni volta che viene aggiunto un nuovo link bisogna usare un algoritmo che individui la presenza di cicli nei grafi
  - Se ne trova uno vieta l'operazione, altrimenti la consente
  - Sono algoritmi molto onoresi dal punto di vista del calcolo
- Utilizzo di un semplice algoritmo che prevede di non percorrere i collegamenti durante l'attraversamento delle directory
  - Si evitano i cicli senza alcun carico ulteriore

Montaggio di un file system

Il file system deve essere **montato** prima di poter essere messo a disposizione dei processi

In particolare deve essere costruita la struttura della directory, composta di volumi

- I volumi devono essere montati affinché siano disponibili nello spazio dei nomi del file system

La procedura di montaggio consiste nel fornire al sistema operativo il nome del dispositivo e la sua locazione (detta **punto di montaggio** - **mount point**) nella struttura di file e directory alla quale agganciare il file system

Alcuni S.O. richiedono che venga specificato il tipo di file system, mentre altri lo determinano ispezionando le strutture del dispositivo

Di solito, un punto di montaggio è una directory vuota

- Ad esempio in UNIX, un file system contente le directory iniziali degli utenti si potrebbe montare come */home*
  - Quindi per avere accesso alla struttura della directory all'interno di quel file system, si permette */home* ai nomi della direcory
    - Ad esempio */home/jane*
    - Se lo stesso file system si montasse come */users*, il percorso per quella stessa directory sarebbe */users/jane*

In pratica per accedere ai file e le sottodirectory di quel file system, sarà necessario far precedere il nome delle directory da */home*

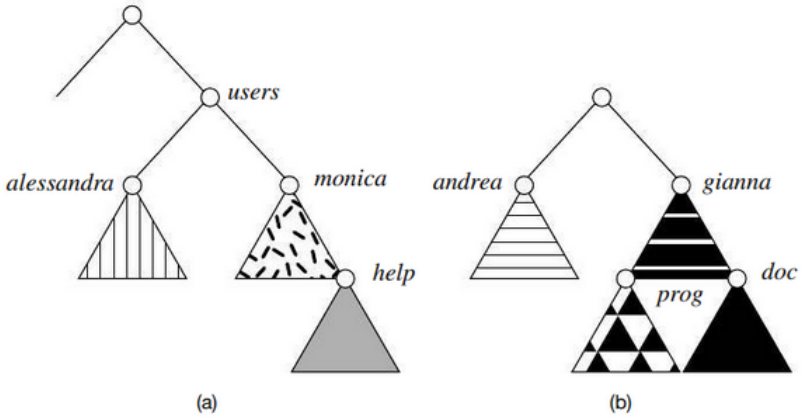
Il passo successivo consiste nella verifica da parte del S.O. della validità del file system contenuto nel dispositivo

Infine il S.O. annota nella sua struttura della directory che il file system è montato al punto di montaggio specificato

File System:

a) Sistema esistente

b) Volume non montato

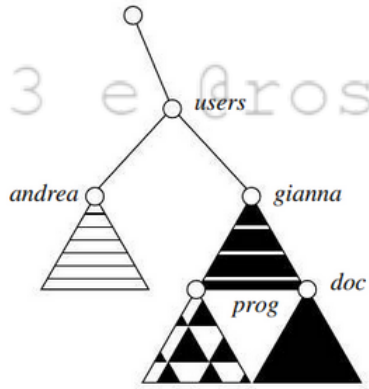


Nella figura illustrata, i triangoli rappresentano i sottoalberi di directory di interesse

Nella figura a) è mostrato un file system esistente, mentre nella figura b) è raffigurato un volume non ancora montato che risiede in */device/dsk*

A questo punto è possibile accedere solo ai file del file system esistente

Punto di montaggio:



Nella figura del punto di montaggio si possono vedere gli effetti dell'operazione di montaggio del volume residente in */device/dsk* al punto di montaggio */users*

Se si smonta il volume, il file system ritorna alla situazione rappresentata precedentemente

Condivisione dei file

Data una struttura della directory che permette la condivisione di file da parte degli utenti, il sistema deve intermediare questa condivisione

- Il sistema può permettere a ogni utente di accedere ai file degli altri utenti per default
- Oppure può richiedere che un utente debba concedere i permessi di accesso ai file

Per realizzare i meccanismi di condivisione e protezione, il sistema deve memorizzare e gestire più attributi per le directory e i file

La maggior parte di essi è arrivata ad adottare per ciascun file o directory i concetti di **proprietario (owner)** o **utente** e **gruppo**

- Il proprietario è l'utente che può cambiare gli attrubuti, concedere l'accesso e che ha maggior controllo sul file

L'attributo di gruppo di un file si usa per definire il sottoinsieme di utenti autorizzati a condividere il file

- In UNIX ad es. il proprietario può fare qualsiasi operazioni sul file, mentre i membri del gruppo hanno azioni limitate

Gli identificatori **(ID)** del gruppo e del proprietario di un file o directory sono memorizzati insieme agli altri attributi del file

Quando un utente richiede un'operazione, per verificare che egli sia il proprietario, si può confrontare l'ID utente con l'attributo che specifica il proprietario

- Analogamente si confrontano gli ID di gruppo

La tecnica della condivisone inoltre ottimizza l'uso della memoria e permette maggiore throughput

File system remoti

L'avvento delle reti ha permesso la comunicazione tra calcolatori remoti

Il primo metodo utilizzato consiste nel trasferimento dei file richiesto in modo esplicito dagli utenti, attraverso programmi come **l'ftp**

- Permette sia **accesso anonimo** che autenticato
  - L'accesso anonimo permette di trasferire file senza avere un account nel sistema remoto

Un secondo metodo è quello del **file system distribuito (distributed file system, DFS)** che permette la visibilità nel calcolatore locale delle directory remote

Il terzo metodo è il **World Wide Web**, usato quasi esclusivamente per lo scambio di file anonimo

Per accedere ai file remoti si una un browser e operazioni distinte per trasferirli

Un altro metodo molto diffuso per la condivisione di file è il **cloud computing**

Modello client-server

I file system remoti permettono il montaggio di uno o più file system in uno o più calcolatori remoti

- I calcolatori contenti i file si chiamano **server**
- I calcolatori che richiedono l'accesso ai file si chiamano **client**
- Un server potrà avere più client ed un client accedere a più server

Il server dichiara che determinate risorse sono disponibili ai client, specificando quali file sono disponibili a quali client

L'identificazione dei client può avvenire tramite i relativi nomi simbolici di rete, o tramite altri identificatori come un indirizzo IP (che possono essere facilmente imitati con tecniche come lo spoofing)

Esistono però soluzioni che permettono ai client l'accesso tramite chiavi di cifratura

Nel caso di UNIX e del suo file system di rete **(network file system, NFS)**, l'autenticazione avviene, per default, tramite le informazioni di rete relative al client

- In questo schema gli ID dell'utente devono coincidere nel client e nel server
- Il server non può determinare i diritti d'accesso ai file
  - Dovrà quindi fidarsi del client e assumere che quest'ultimo presenti l'identificatore corretto

Sistemi di informazione distribuiti

Per semplificare la gestione dei servizi client-server, i **sistemi di informazione distribuiti**, noti anche come **servizi di naming distribuiti**, sono stati concepiti per fornire un accesso unificato alle informazioni necessarie per il calcolo remoto

- Il **sistema dei nomi di dominio (domain name system, DNS)**, fornisce le traduzioni dai nomi dei calcolatori agli indirizzi di rete per l'intera internet

Tipi di malfunzionamento

I file system locali possono presentare vari malfunzionamenti:

- Alterazioni dei dati
- Informazioni necessarie alla gestione dei dischi **(metadati)**
- Malfunzionamenti dei controllori dei dischi
- Problemi ai cavi di connessione

I file system remoti ne introducono altri, come ad es. la caduta di un sistema dalla connessione **(crash)**

In particolare per i file system remoti è necessario mantenere alcune **informazioni di stato** tra i client e i server

- Client e server, tenendo traccia delle loro attività possono riprendersi da malfunzionamenti come i crash

Semantica della coerenza

La **semantica della coerenza** è un importante criterio per la valutazione di file system che consentano la condivisione dei file

- Specifica il modo in cui più utenti devono accedere contemporaneamente a un file condiviso
- Serve a specificare quando le modifiche apportate ai dati da un utente possono essere osservate dagli altri

Una serie d'accessi, cioè letture e scritture, tentati da un utente a uno stesso file è compresa tra una coppia di operazioni di *open()* e *close()*

- Tale serie di accessi si chiama **sessione di file**

Esistono vari esempi di semantica della coerenza per illustrare questo concetto:

- **Semantica UNIX**
  - Le scritture in un file aperto da parte di un utente sono immediatamente visibili ad altri utenti che hanno lo stesso file contemporaneamente aperto
  - Il file ha quindi una singola immagine e tutti gli accessi si alterano
  - Esiste un metodo di condivisione in cui gli utenti condividono il puntatore alla locazione corrente del file

- **Semantica delle sessione** (proveniente dal file system Andrew)
  - Le scritture in un file aperto da un utente non sono visibili immediatamente ad altri utenti che hanno contemporaneamente lo stesso file aperto
  - Una volta chiuso il file, le modifiche apportate saranno visibili solo nelle sessioni che iniziano successivamente
    - Le istanze di file già aperte non riportano queste modifiche
    - Il file è associato a parecchie immagini, probabilmente diverse
- **Semantica dei file condivisi immutabili**
  - Una volta che un file è stato dichiarato **condiviso** dal suo creatore, non può essere modificato
  - Il file immutabile presenta due caratteristiche:
    - Il suo nome non si può riutilizzare
    - Il suo contenuto non può essere modificato
    - La condivisione permette quindi solo la lettura

Protezione

Le informazioni di un sistema devono essere protette dai danni fisici (affidabilità) e da accessi impropri (protezione)

L'affidabilità è generalmente data da più copie dei file (meccanismi di backup)

La protezione si può invece ottenere in molti modi

Il proprietario/creatore di un file deve poter controllare:

- Le operazioni possibili sui file
- A chi permetterne l'esecuzione

Se ogni file fosse accessibile a tutti gli utenti, potrebbero verificarsi modifiche o cancellazioni non desiderate

Quindi la necessità di proteggere i file deriva dalla possibilità di accedervi

È necessario quindi un **accesso controllato**

Tipi di accesso

Gli accessi si permettono o si negano secondo diversi fattori

Si possono controllare diversi tipi di operazione:

- Lettura
  - Lettura da file
- Scrittura
  - Scrittura o riscrittura di file
- Esecuzione
  - Caricamento di file in memoria ed esecuzione
- Aggiunta (append)
  - Scrittura di nuove informazioni in coda ai file
- Cancellazione

    - Cancellazione di file
- Elencazione
    - Elencazione del nome e degli attributi dei file

Si possono controllare anche altre operazioni, come ridenominazione, copiatura o modifica dei file

Controllo degli accessi

Il problema della protezione comunemente si affronta rendendo l'accesso dipendente dall'identità dell'utente

- Esistono vari metodi, come ad es. l'associazione di una **parola chiave (password)** a ciascun file

Tuttavia lo schema più generale per realizzare gli accessi dipendenti dall'identità consiste nell'associare un **elenco di controllo degli accessi (access control list, ACL)** a ogni file e directory

- In tale lista sono specificati i nomi degli utenti e i relativi tipi di accesso consentiti

Se un utente richiede un accesso a un file, il S.O. esamina la lista di controllo degli accessi associata al file

- Se l'utente è presente nella lista per quel tipo di accesso viene autorizzato
- Altrimenti si verifica una violazione della protezione e si nega l'accesso al file

Il problema maggiore riguarda la lunghezza degli elenchi

Per condensarne la lunghezza, molti sistemi raggruppano gli utenti di ogni file in tre classi distinte:

- **Proprietario:** è l'utente che ha creato il file
- **Gruppo:** è un insieme di utenti che condividono il file e hanno bisogno di tipi di accesso simili
- **Universo:** tutti gli altri utenti del sistema

Si unisce quindi lo schema di controllo degli accessi secondo queste tre classi

In UNIX solo il **superuser** può creare e modificare gruppi

Per definire la protezione occorrono solo tre campi

Ogni campo è formato da un insieme di bit, ciascuno dei quali permette o impedisce l'accesso associato

In UNIX sono definiti tre campi di tre bit ciascuno: **rwx**

- *r* controlla l'accesso per la lettura
- *w* per la scrittura
- *x* per l'esecuzione

Tre campi separati sono riservati al proprietario del file, al gruppo proprietario e a tutti gli altri utenti

Si consideri il seguente esempio

- Le tre classi di utenti sono indicate come segue:
    - Accesso proprietario 7 → rwx(1, 1, 1)
    - Accesso gruppo 6 → rwx(1, 1, 0)
    - Accesso pubblico 1 → rwx(0, 0, 1)

- Si chiede al superuser di creare un gruppo *G*, e aggiungere alcuni utenti al gruppo
  - Per definire un gruppo di un file
    - `chgrp G game`
  - Per definire un accesso appropriato
    - `chmod 761 game`

Le proprietà d'accesso al file saranno quindi:

rwx per il proprietario, *rw* per il gruppo *G* e *x* per tutti gli altri utenti