SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

L'interfaccia del file system

Prof. Luca Gherardi

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

Sommario

- Il concetto di file
- Metodi di accesso
- Struttura delle directory
- Montaggio del file system
- Condivisione dei file
- Protezione

Il file system

- È l'aspetto del SO più visibile dal punto di vista dell'utente
- È organizzato in due parti distinte
  - Un insieme di **file** contenti informazioni
  - Una **struttura della directory**, che organizza i file in una struttura gerarchica e fornisce informazioni sui file stessi
- Fornisce gli strumenti per:
  - La creazione e la manipolazione dei file,
  - Garantire l'affidabilità dei file quando si verificano guasti
  - Permettere la protezione dei file e la condivisione tra gli utenti (anche per l'accesso concorrente)

Gestione dei file

Il SO gestisce i file attraverso due componenti:

- Il **file system**
- Il **sotto-sistema di I/O**
  - accesso efficiente ai dati memorizzati nei dispositivi di I/O,
  - gestione efficiente dei dispositivi di I/O

Il concetto di file (1)

- Visione logica uniforme della memorizzazione delle informazioni (logica: vista utente)
- Il file nasconde all'utente le caratteristiche fisiche dei vari dispositivi di memoria e come essi memorizzano l'informazione
  - Non dipende dal tipo di dispositivo fisico
    - Disco magnetico, disco a strato solido, nastro, cd, dvd, …
    - Diversa velocità di acceso

Il concetto di file (2)

- Dal punto di vista dell'utente il file è l'unità di memorizzazione elementare
- Le informazioni contenute in un file possono essere di vario tipo:
  - Dati
    - numerici, alfabetici, binari
  - Programmi
    - Sorgenti e/o oggetto
  - Riferimenti (shortcut .lnk)

Attributi del file (1)

- Un file consiste di due tipi di dati:
  - i **dati** contenuti nei file
  - i **meta-dati** (o dati di controllo o attributi) usati per accedere al file

- I meta-dati comprendono:
  - **Nome**: identificatore simbolico del file
    - Unica informazione mantenuta in una forma leggibile dall'utente
    - Alcuni sistemi sono case-sensitive, altri no
  - **Identificatore**: etichetta univoca che che identifica il file
    - Tipicamente è un numero
    - Il SO identifica il file attraverso l'identificatore e non attraverso il nome

Attributi del file (2)

- I meta-dati comprendono:
  - **Tipo**: necessario per supportare differenti tipi di file
    - E capire come interpretarli
  - **Locazione**: locazione fisica del file in uno specifico dispositivo
    - È un puntatore
  - **Dimensione**: dimensione corrente del file (byte o blocchi)
    - Può memorizzare anche la massima dimensione consentita
  - **Protezione**: determina chi può leggere, scrivere, eseguire
  - **Ora e data e identificativo dell'utente**:
    - Momento della creazione, modifica ed ultimo accesso
    - Dati per implementare protezione, sicurezza e controllo d'uso

Attributi del file (3)

- I meta-dati sono mantenuti nella **struttura della directory** (struttura dati *anch'essa mantenuta in memoria di massa*)
- Ogni elemento della directory può occupare più di 1KB
- Negli attuali sistemi quindi la directory può occupare svariati MB
- Per questo motivo anche essa viene caricata in memoria centrale con cautela
  - Possibilmente non in tutta la sua estensione
  - Quando necessario

Operazioni di base sui file (1)

- Essendo un file un'astrazione di una parte dell'informazione definita nella memoria fisica è necessario definire l'insieme delle operazioni ammissibili
  - **Creazione**: è composta da due step
    - Ricerca dello spazio necessario nel file system
    - Creazione dell'elemento nella directory dei file (nome, posizione, dimensione, …)
  - **Scrittura di un file**: effettuata attraverso una chiamata di sistema `write(nome_file, data)`
    - Il SO cerca la posizione del file nel file system basandosi su `nome_file`
    - Il SO mantiene un puntatore alla successiva area di scrittura

Operazioni di base sui file (2)

- Operazioni ammissibili:
  - **Lettura di un file**: effettuata attraverso una chiamata di sistema `read(nome_file, &data)`
    - Il SO cerca la posizione del file nel file system basandosi su `nome_file`
    - Copia il contenuto del file nell'area di memoria indicata da `&data`
    - Il SO mantiene un puntatore alla successiva area di lettura
    - Tipicamente si usa un solo puntatore per lettura e scrittura
  - **Riposizionamento in un file**: sposta il puntatore di lettura/scrittura in una nuova posizione all'interno del file
    - Anche questa operazione richiede la ricerca della posizione del file nel file system
    - Non richiede operazioni di I/O

Operazioni di base sui file (3)

- Operazioni ammissibili:
  - **Cancellazione di un file**
    - Il SO cerca la posizione del file nel file system
    - Rilascia lo spazio associato al file
    - E elimina il corrispondente elemento dalla directory
  - **Troncamento di un file**: elimina il contenuto di un file senza eliminare il file stesso
    - Preserva il valore degli attributi (eccetto dimensione)
    - Rilascia lo spazio occupato
- Altre operazioni che nascono dalla combinazione delle precedenti:
  - E.g. copia di un file (creazione del nuovo file + lettura del vecchio + scrittura del nuovo)

Operazioni di base sui file (4)

- Tutte le operazioni viste richiedono al SO la ricerca del file sul dispositivo
- Per rendere più efficiente la ricerca, il SO mantiene una **tabella dei file in uso (o aperti)**
  - La tabella contiene il descrittore del file contenuto nella directory
  - Un file viene inserito nella tabella al momento della chiamata `open()` e tolto al momento della chiamata `close()`
- Alcuni sistemi eseguono in modo automatico la chiamata `open()` nel momento in cui un file viene referenziato per la prima volta
- Altri richiedono una chiamata esplicita prima di poter operare sul file
- La `open()` può ricevere come parametro la modalità di accesso
  - Il successo dell'apertura dipende dal confronto tra la modalità richiesta e i permessi del file

Operazioni di base sui file (5)

- In sistemi multi-utente (e.g. Unix-like) la gestione dei file aperti è più complessa:
  - La tabella viene organizzata su due livelli
  - **Una tabella per il SO**:
    - Contenente le informazioni sui file aperti che non dipendono dal processo (posizione sul disco)
    - Un contatore di aperture per ogni file (incrementato sulle `open()` e decrementato sulle `close()`)
  - **Una tabella per ciascun processo**:
    - Contenente i riferimenti ai file aperti da quel processo (puntatore di lettura/scrittura, permessi, …)
    - E per ogni file un puntatore all'elemento che lo descrive nella tabella del SO
  - La prima chiamata `open()` su un file modifica entrambe le tabelle mentre le successive solo le tabelle dei processi

Operazioni di base sui file (6)

- In alcuni SO il file system offre dei **lock** anche per i file
  - **Lock condivisi**, simili ai lock di lettura
  - **Lock esclusivi**, simili ai lock di scrittura
  - Non tutti i SO offrono entrambi i lock
- Il SO può offrire un meccanismo di protezione dei file
  - **A lock obbligatorio**:
    - Impedisce a ciascun processo di accedere a un file sul quale è stato già acquisito il lock
    - Indipendentemente da come sia programmato il processo (i.e. anche se non cerca di acquisire il lock)
  - **A lock consigliato**:
    - Non impedisce l'accesso a file sui quali il lock è già stato acquisito
    - Il programmatore deve garantire la corretta acquisizione del lock

Tipi di file

- È importante che il SO sappia riconoscere i diversi tipi di file
  - Possono essere gestiti in modo migliore
  - Ad esempio si può impedire la stampa di un file oggetto in formato binario
- Il tipo del file viene tipicamente rappresentato con la tecnica dell'**estensione**
  - Il nome del file è diviso in due parti, separate da un punto
  - nome.estensione (e.g. testo.doc)
- In base all'estensione sull'estensione un file
  - Può essere aperto con l'applicativo giusto (tabelle associative app/ext)
  - Può essere interpretato in modo corretto

Tipi di file – Nome ed estensione

Tipo di file	Estensione usuale	Funzione
Eseguibile	`.exe`, `.com`, `.bin` o nessuna	Programma in linguaggio macchina pronto per essere eseguito
Oggetto	`.obj`, `.o`	Compilato, linguaggio macchina, non linkato
Codice sorgente	`.c`, `.cc`, `.java`, `.pas`, `.asm`, `.a`	Codice sorgente in vari linguaggi
Batch	`.bat`, `.sh`	Comandi per l'interprete dei comandi
Testo	`.txt`, `.doc`	Dati testuali, documenti
Programma di elaborazione testi	`.wp`, `.tex`, `.rtf`, `.doc`	Vari formati di programmi di elaborazione testi
Libreria	`.lib`, `.a`, `.so`, `.dll`	Librerie di procedure per programmatori
Stampa o visualizzazione	`.ps`, `.pdf`, `.jpg`	File in formato ASCII o binario per la stampa o la visualizzazione
Archivio	`.arc`, `.zip`, `.tar`	File collegati, raggruppati in un unico file, talvolta compressi, a scopo di archiviazione o di memorizzazione
Multimedia	`.mpeg`, `.mov`, `.rm`	File binari contenenti informazioni audio o audio/video

Struttura dei file

Un file ha:

- **Struttura logica**
  - I dati sono un flusso (insieme) di unità logiche di lunghezza fissa detti **blocchi logici** (o **record**)
  - Ad es. nei sistemi UNIX-like un record è un byte
- **Struttura fisica**
  - I dati sono organizzati in unità fisiche di lunghezza fissa e dipendente dal dispositivo dette **blocchi fisici**
  - Tutti gli I/O si eseguono in unità di blocchi fisici
  - Dimensioni tipiche dei blocchi di unità a disco rigido 512 byte
- **La struttura logica e fisica sono differenti!**
  - I file system **impacca** le unità logiche in unità fisiche prima di fare una scrittura sul dispositivo
    - Possibilità di **frammentazione interna**

Metodi di accesso

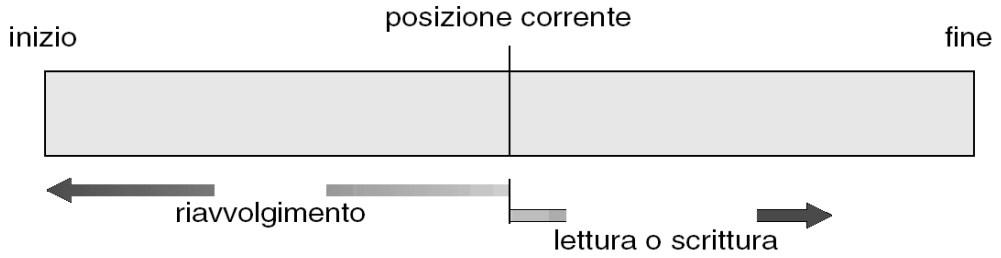
- Le informazioni memorizzate in un file devono essere caricate in memoria centrale per poter essere utilizzate
- Una volta in memoria i processi possono accedere ai vari blocchi logici del file

- Vedremo diversi metodi di accesso alle informazioni contenute in un file:
  - Accesso sequenziale
  - Accesso diretto
  - Basati su indice

Accesso sequenziale

- Le informazioni contenute del file vengono elaborate in modo ordinato, una dopo l'altra

- Le operazioni di lettura e scrittura muovono il puntatore del file alla cella successiva o precedente
  - `read_next`
  - `write_next`
  - `read_previous`
  - `write_previous`



Accesso diretto (1)

- I processi accedo **direttamente** ad uno specifico blocco logico di un file
- Ogni blocco logico ha un **identificatore univoco**, che identifica lo scostamento del blocco dall'inizio del file
  - In realtà i blocchi possono essere sparsi a piacere nel disco (vedremo le problematiche di allocazione dei file nella prossima lezione)
- Non è necessario leggere i file in base all'ordine sequenziale
- Primitive di lettura e scrittura:
  - `read_n`
  - `write_n`
- In alternativa
  - `position_n + read_next o write_next`

Accesso diretto (2)

- Nei sistemi ad accesso diretto è facile implementare un accesso sequenziale

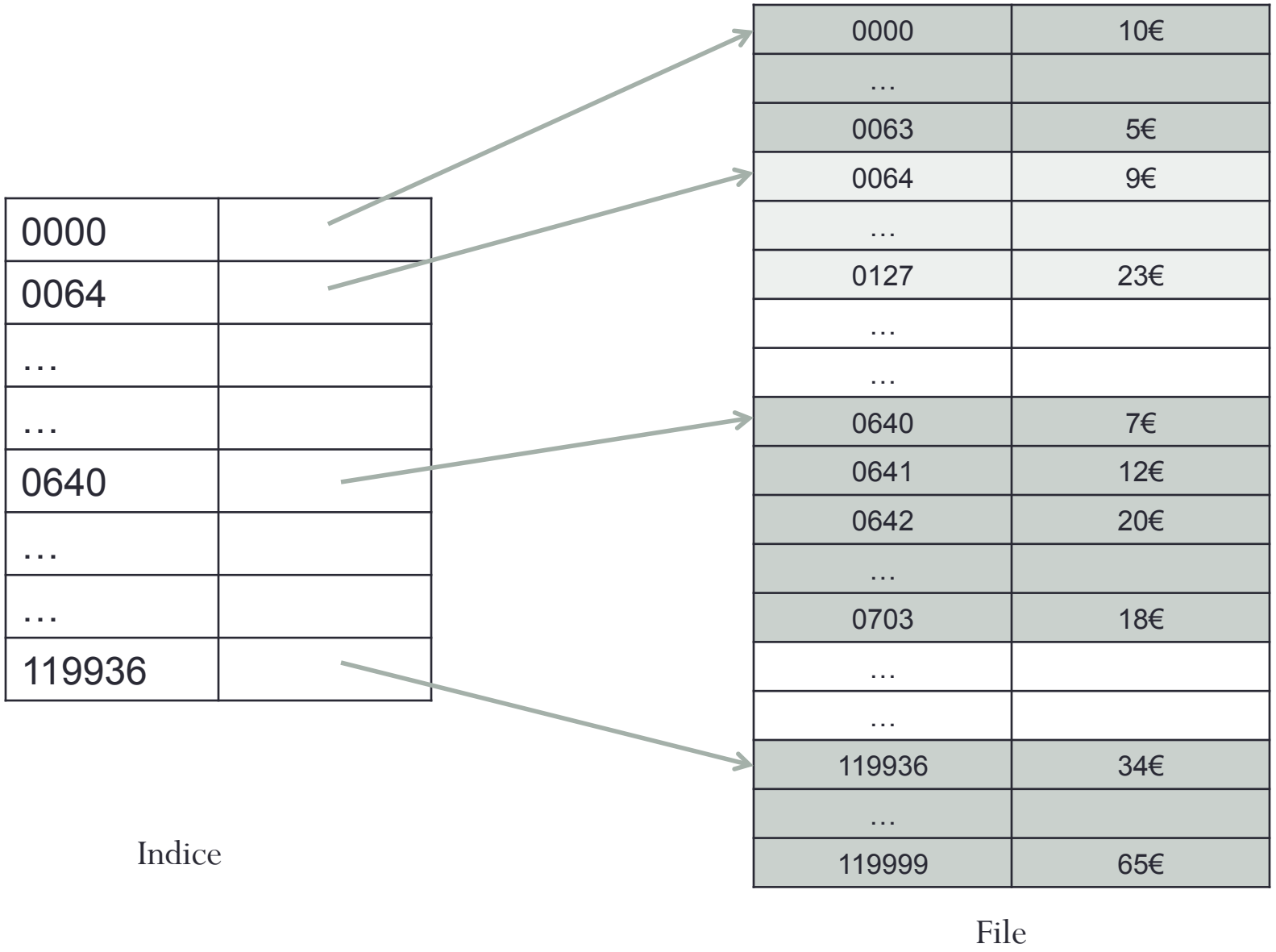
Accesso sequenziale	Implementazione dell'accesso diretto
reset	*cp* = 0;
lettura del successivo	*leggi cp*; *cp* = *cp* +1;
scrittura del successivo	*scrivi cp*; *cp* = *cp* +1;

- Non sarebbe efficiente implementare un accesso diretto in un sistema ad accesso sequenziale

Metodo di accesso tramite file indice (1)

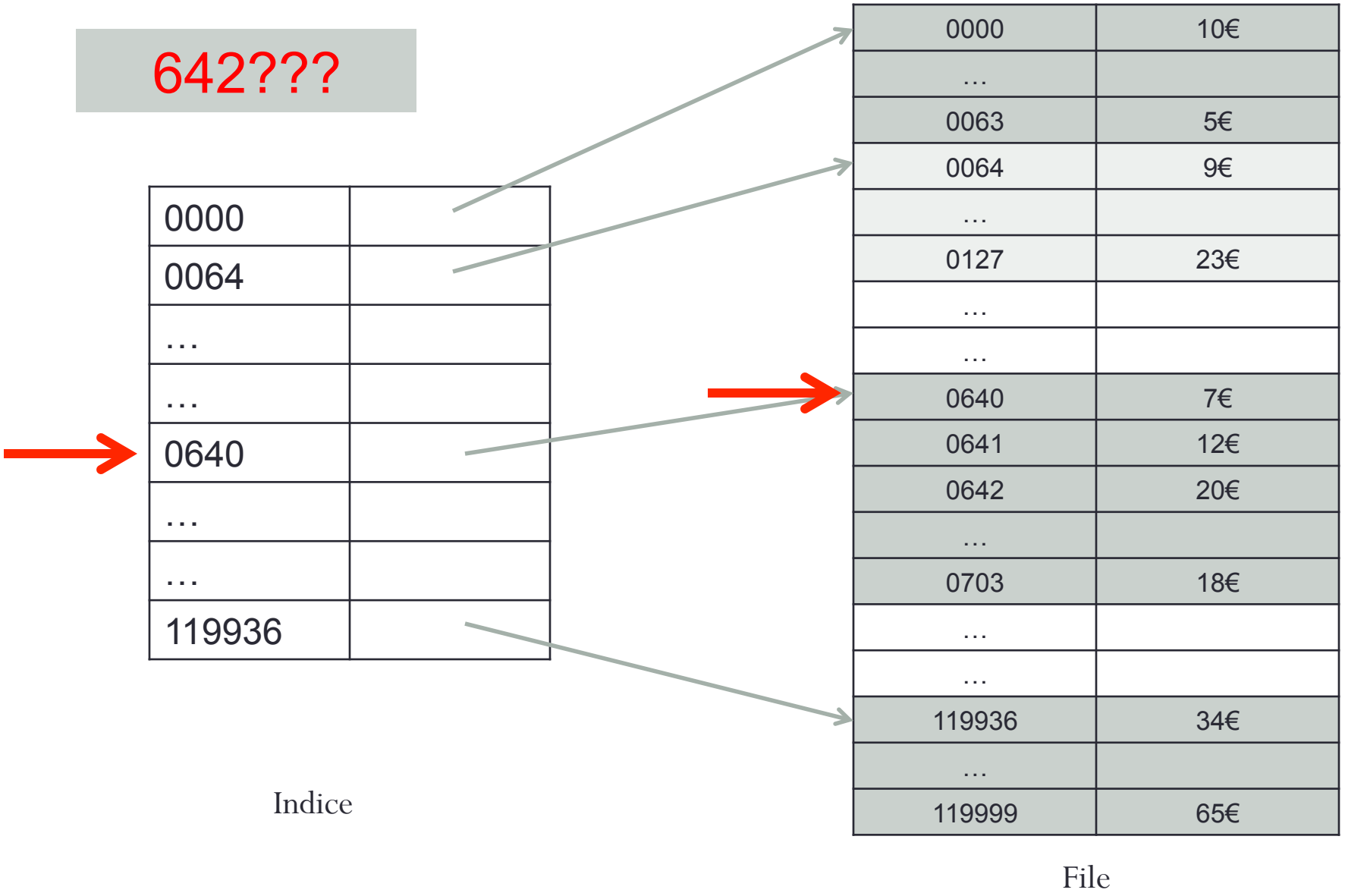
- La ricerca di un blocco nel file prevede
  - La ricerca del puntatore a tale blocco in un indice (attraverso una chiave)
  - L'uso del puntatore per accedere al blocco effettivo
- Esempio:
  - Dato un disco con blocchi da 1024 byte e elementi da 16 byte da memorizzare (ad esempio dei codici con associati dei prezzi)
  - Un blocco può contenere 64 elementi
  - Un file da 120000 elementi occupa 1875 blocchi
  - Avremo quindi un indice da 1875 elementi che viene mantenuto in memoria centrale
- Se il file è ordinato in base ai codici, l'accesso ad un determinato elemento richiede
  - Una ricerca binaria nell'indice (si cerca l'indice più vicino possibile, per difetto, al codice voluto)
  - Un accesso sequenziale al file per trovare il codice voluto

Metodo di accesso tramite file indice (2)



Metodo di accesso tramite file indice (2)

642???

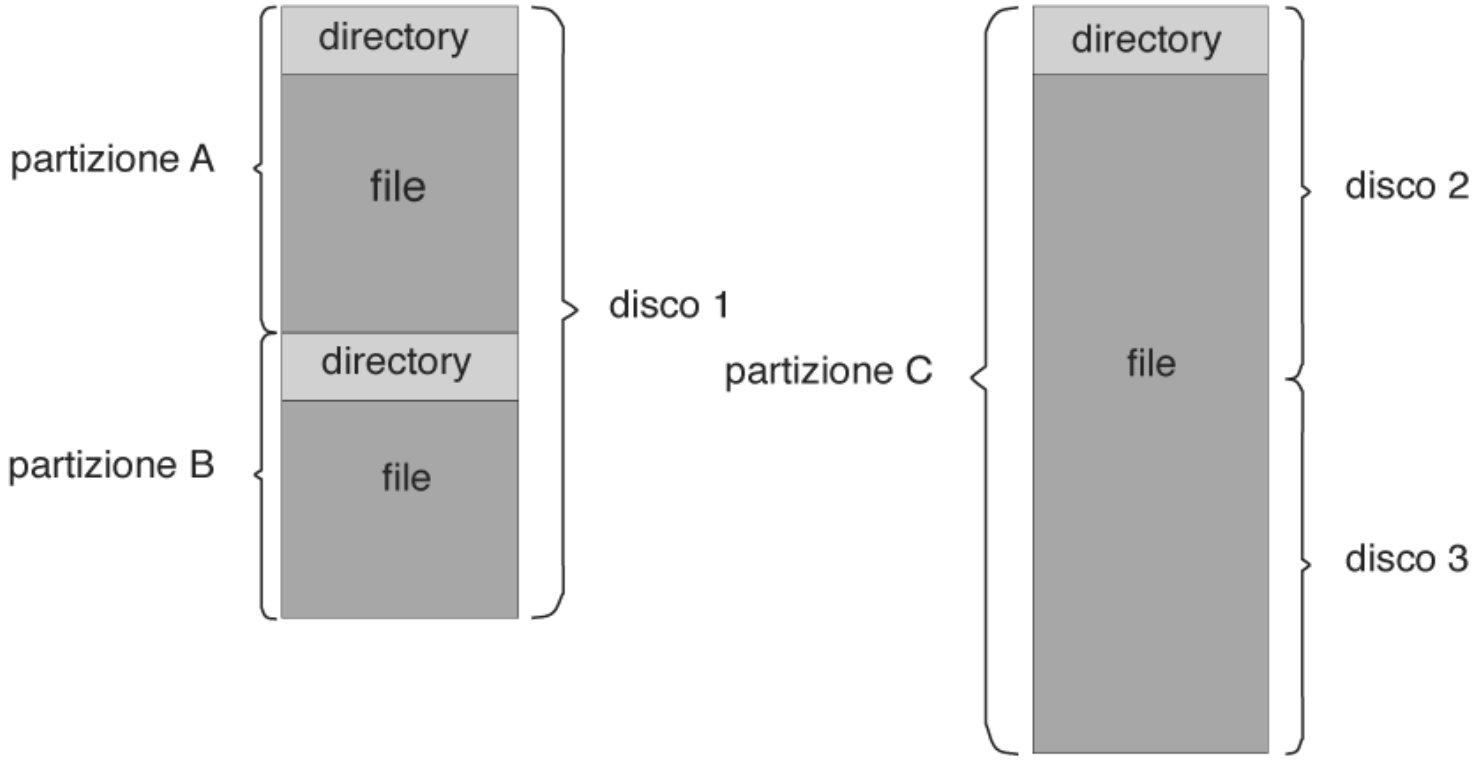


Struttura dei dispositivi di memorizzazione (1)

- Un dispositivo di memorizzazione può essere utilizzato
  - Per un unico file-system
  - Oppure può essere diviso in più parti, chiamate **partizioni**
- Utilità:
  - Limitare le dimensioni del file-system
  - Avere diversi tipi di file-system sullo stesso calcolatore
  - Liberare per altri scopi parte del disco (e.g. Swap)
- Ogni partizione contenente un file system è chiamata volume
  - Può essere vista come un **disco virtuale**
- Più dispositivi di memorizzazione possono essere raccolti in insiemi di RAID

Struttura dei dispositivi di memorizzazione (2)

- Ogni volume contiene una directory che memorizza gli attributi di tutti i file memorizzati



Generalità sulla directory

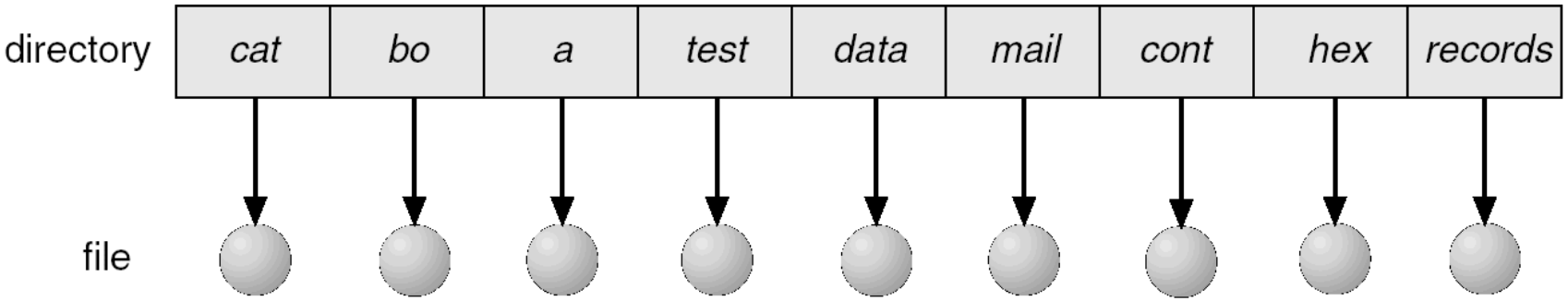
- La directory può essere vista come una tabella che dato un nome di file restituisce informazioni sul file stesso
  - Gli attributi
- La directory può essere organizzati in diversi modi che analizzeremo nelle seguenti slide:
  - Directory a singolo livello
  - Directory a due livelli
  - Directory con struttura ad albero
  - Directory con struttura a grafo aciclico
  - Directory con struttura a grafo generale

Operazioni offerte dalla directory

- **Ricerca di un file**
  - Non solo sul nome preciso ma anche in base a pattern (e.g. `*.exe`)
- **Creazione di un file**
- **Cancellazione di un file**
- **Elencazione di una directory**
  - Permette di visualizzare tutti i file presenti in una directory (o parte) e i relativi attributi
- **Ridenominazione di un file**
  - Può comportare lo spostamento del file nella directory
- **Attraversamento del file system**

Directory a singolo livello (1)

- Tutti i file sono contenuti in un'unica directory (un'unica tabella)
  - Facilmente gestibile e comprensibile



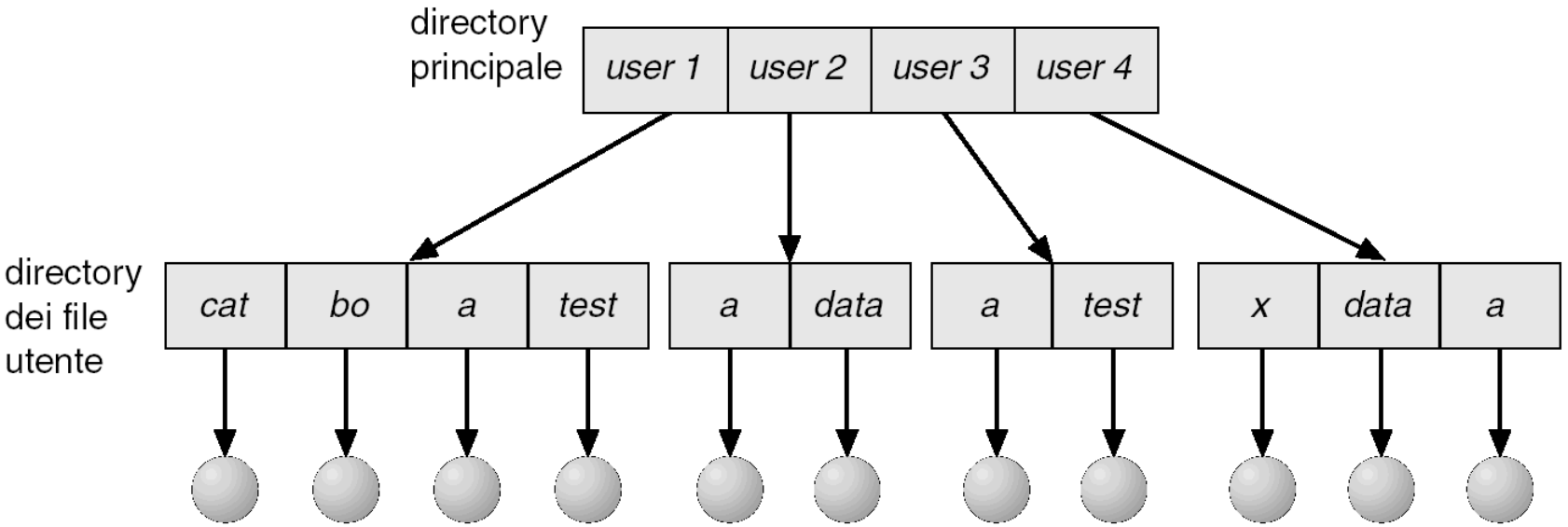
Directory a singolo livello (2)

- Presenta notevoli limiti che si manifestano all'aumentare del numero di file e utenti:
  - Tutti i file devono avere nomi unici
    - Due utenti non possono avere file con lo stesso nome
    - I sistemi operativi limitano la lunghezza del nome (e.g. MS-DOS 11 caratteri)
      - Di conseguenza questo limita il numero massimo di possibili nomi, quindi di file
  - Non potendo raggruppare i file in livelli (i.e. cartelle) è difficile organizzare i file
    - È necessario ricordare i nomi quando si vuole aprire un file

Directory a due livelli (1)

- È un evoluzione dello schema a singolo livello
- Ogni utente ha una sua directory personale (User File Directory UFD)
- Esiste una directory principale che contiene tanti elementi quanti sono gli utenti (Master File Directory MFD)
  - È indicizzata secondo i nomi degli utenti
  - Ogni elemento punta ad una UFD
- Ogni volta che viene create una nuova directory è necessario aggiornare la MFD
  - Solo l'amministratore del sistema può eseguire tali operazioni

Directory a due livelli (2)



Directory a due livelli (3)

- Quando un utente esegue una ricerca il SO esplora solo la UFD riservata a quell'utente
  - In questo modo più utenti possono avere file con lo stesso nome
    - ma sono ancora organizzati su un unico livello
  - Allo steso modo un'operazione di remove non può eliminare file di altri utenti
- Problema: non è possibile condividere file tra gli utenti in modo semplice
- Soluzione: i file di un altro utente possono essere acceduti specificando un percorso (path)
  - composto da nome utente e nome file, e.g. `/utenteA/file.doc`

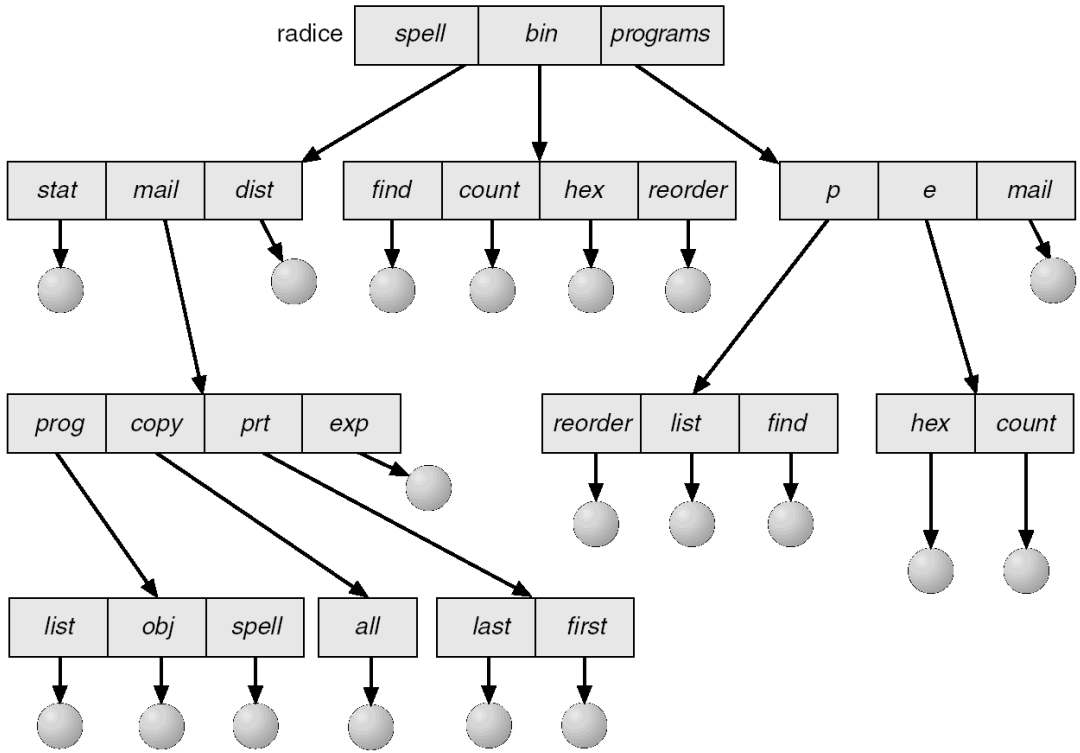
Directory a due livelli (4)

- Il problema della condivisione dei file si presenta anche per i comandi offerti dal sistema (e.g. ls, mv, cp)
- Tipicamente essi possono essere eseguiti lanciando un file che ha lo stesso nome del comando
- Soluzione A: i file di sistema possono essere copiati in ogni directory utente
  - Potrebbe essere molto costoso
- **Soluzione B**: si definisce una directory utente speciale che contiene i file di sistema
  - Quando un utente ricerca un file il SO esplora
    - La directory utente
    - Se non trova il file, la directory speciale contenente i file di sistema

Directory con struttura ad albero (1)

- L'idea della directory a due livelli viene estesa per creare un albero di **profondità arbitraria**

- La directory principale (radice) non è necessariamente organizzata in base agli utenti



Directory con struttura ad albero (2)

- Ogni file è caratterizzato da un **path**
  - Concatenazione delle directory partendo dalla radice
  - Nome del file
  - E.g. /spell/mail/exp
- In questa struttura **le directory sono degli speciali file**, tutti organizzati secondo la stessa struttura
  - Il SO distingue file a directory attraverso uno specifici bit
- Il SO introduce delle **chiamate di sistema** per operare sulle directory
  - In Ubuntu `mkdir`, `rmdir`

Directory con struttura ad albero (3)

- Ogni ricerca o operazione su file viene fatta partendo dalla **directory corrente**
  - È la directory di lavoro
  - Se un utente vuole operare su un file residente in un'altra directory deve
    - Specificare l'intero percorso, oppure
    - Cambiare directory corrente (comando `cd`)
- La directory corrente permette di distinguere tra path
  - Assoluti (partono dalla directory radice)
  - Relativi (partono dalla directory corrente)

Directory con struttura ad albero (4)

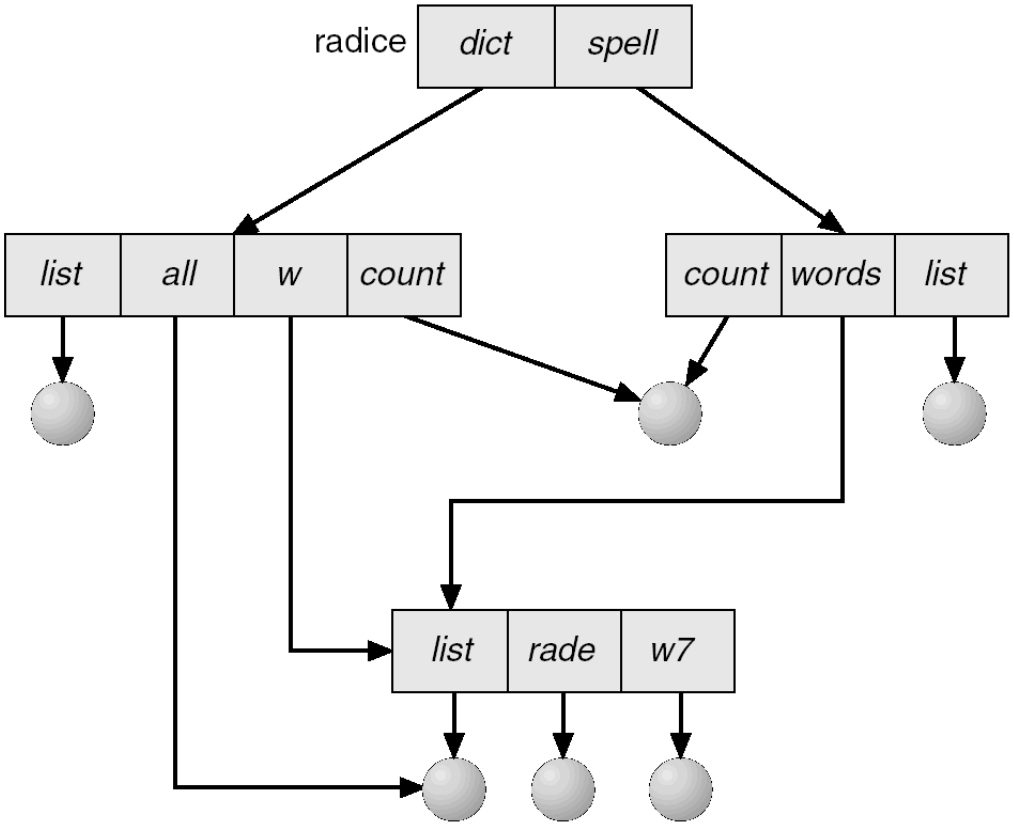
- L'eliminazione di una directory comporta
  - Eliminazione di tutti i file, e
  - Eliminazione ricorsiva delle sotto-directory
- Due politiche di gestione
  - Eliminare una directory con un singolo comando
  - Permettere di eliminare una directory solo se vuota, o su esplicita richiesta (e.g. `rm –r`)

Directory con struttura ad albero (5)

- Altri vantaggi:
  - Il file possono essere facilmente organizzati in directory gerarchiche in base al contenuto
  - Il file possono essere facilmente condivisi da più utenti
  - Ricerca efficiente: visita dell'albero in tempo O(h)

Directory con struttura a grafo aciclico (1)

- In alcune situazioni è necessario che lo stesso file (o directory) sia contenuto in più directory
  - E.g. file condivisi tra più utenti
- La struttura a grafo aciclico risponde a questo problema



Directory con struttura a grafo aciclico (2)

- La condivisione dei file può essere implementata secondo due modalità

- **Link**
  - È un terza tipologia di elemento che può essere contenuta in una directory
  - Associa un nome al percorso di un file presente nel sistema
  - Quando l'utente esegue un operazione sul link il SO
    - Ricava il percorso del file associata al link
    - Esegue le operazioni sul file linkato
  - Possono essere usati anche per altri scopi (e.g. gestire più versioni della stessa libreria dinamica)
- **Dupplicazione**
  - Esistono due copie dei file (directory) condivisi
  - In questo caso è necessario garantire la consistenza delle modifiche

Directory con struttura a grafo aciclico (3)

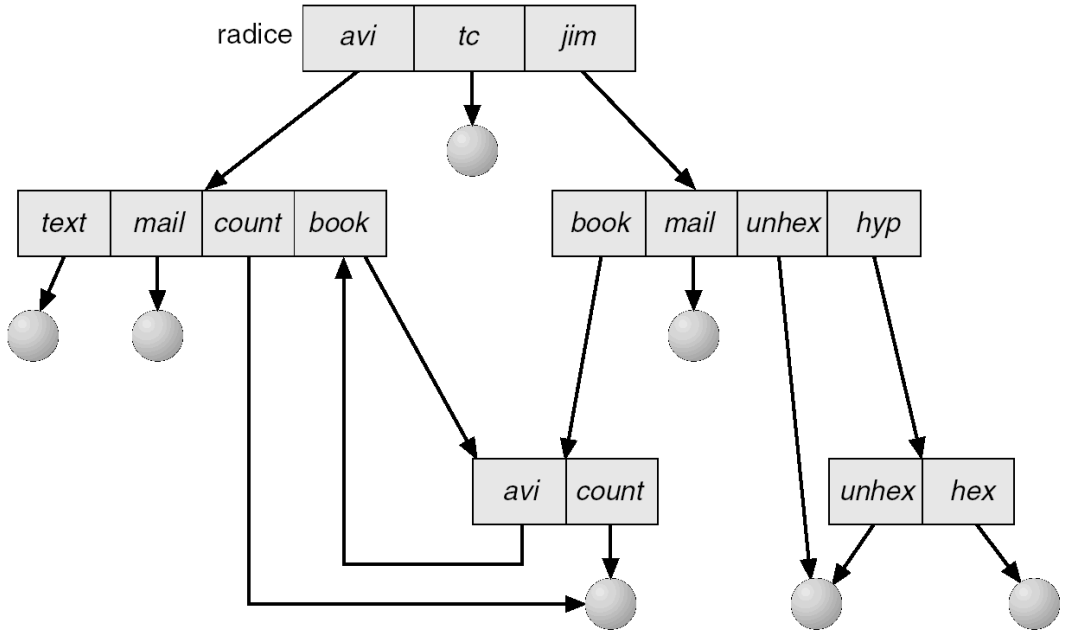
- La condivisione introduce delle problematiche
- **Complessità**
  - Path diversi possono riferire allo stesso file
  - L'esecuzione di operazioni non deve essere eseguita più volte sui file condivisi
    - Operazioni statistiche di conteggio

Directory con struttura a grafo aciclico (4)

- La condivisione introduce delle problematiche
- **Eliminazione**
  - Quando si elimina il file fisico si potrebbero avere dei **dangling link**
    - Si hanno dei puntatori che puntano a file non esistenti, o peggio
    - quando viene creato un file con lo stesso nome del precedente il puntatore punta ad un file diverso da quello voluto
  - Diverse soluzioni
    - Eliminare anche tutti i puntatori (richiede una lista dei puntatori)
    - Gestire il dangling link nel momento in cui viene referenziato per la prima volta
    - Eliminare il file solo quando l'ultimo link è stato rimosso (ogni file ha un contatore dei link)

Directory con struttura a grafo generale (1)

- Il vantaggio di un grafo aciclico è dato dalla semplicità degli algoritmi di attraversamento e ricerca
- In un grafo a struttura generale infatti si potrebbero avere diverse difficoltà
  - Cicli infiniti di ricerca
  - Difficile definire quando un file possa essere effettivamente rimosso



Directory con struttura a grafo generale (1)

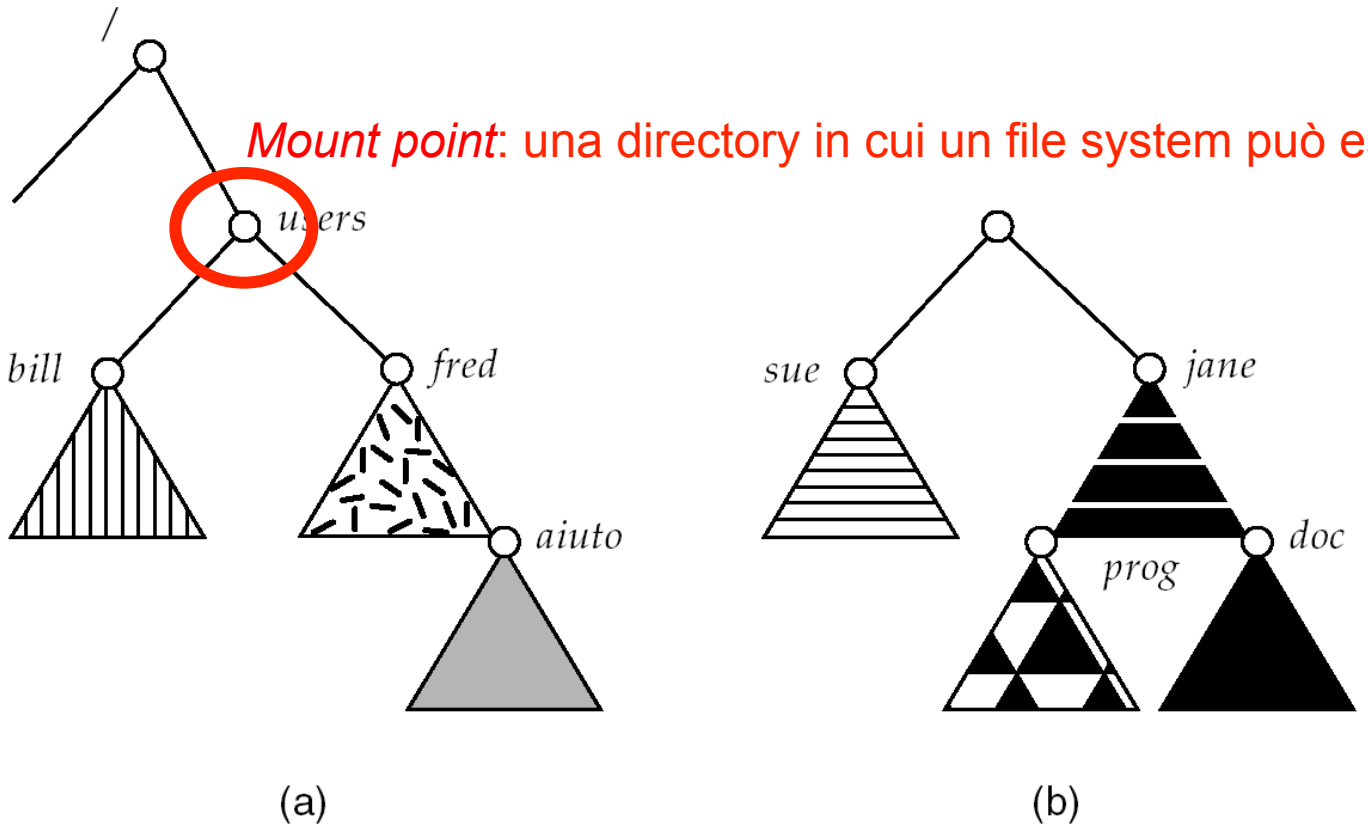
- È quindi preferibile evitare la presenza dei cicli
- Diverse soluzioni
  - Schema di **garbage collection**:
    - Elimina eventuali file con contatore dei link non nullo per via del ciclo, ma con link effettivamente non esistenti
  - Ogni volta che viene aggiunto un nuovo link usare un algoritmo per rilevare i cicli nei grafi
  - Permettere solo link ai file, non sotto-directory

Montaggio di un file system (1)

- Come è possibile "inserire" un nuovo dispositivo in una struttura di directory già esistente?
- Similmente alle operazioni di apertura dei file, esiste l'operazione di **montaggio del file system**
  - Dato un nuovo dispositivo l'utente specifica una **directory esistente** (punto di montaggio) nella quale la directory del dispositivo dovrà essere inserita
  - Successivamente il SO verifica la validità del file system da montare
    - Si verifica che la directory da inserire abbia il formato desiderato
  - Il SO annota nella struttura della directory l'inserimento del nuovo file system nel punto di montaggio desiderato

Montaggio di un file system (2)

a. Struttura della directory esistente
b. File system da montare

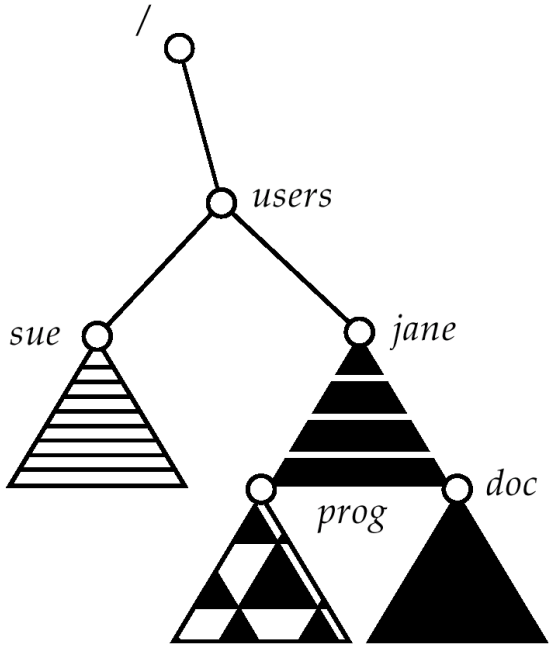


Mount point: una directory in cui un file system può essere montato

Montaggio di un file system (3)

- **Risultato del montaggio**

- Se il punto di montaggio è una directory non vuota, il SO può
  - Impedire il montaggio
  - Effettuare il montaggio e rendere invisibili i file precedentemente presenti fino allo smontaggio del dispositivo



Montaggio di un file system (4)

- Alcuni SO montano automaticamente il file system quando rilevano un nuovo dispositivo
- Altri prevedono una richiesta di montaggio da parte dell'utente
  - Mac Os X monta automaticamente i dispositivi nella directory `/Volumes` e offre all'utente un link sul desktop
  - Windows (fino a Vista) mantiene una struttura a due livelli estesa e associa ad ogni dispositivo una lettera di unità
    - E.g. `C:\directory_radice\directory_a\directory_b\documento.doc`
    - Si noti la differenza nel carattere di separazione tra sistemi Unix e Windows
  - In altri sistemi Unix esistono dei dispositivi noti che vengono montati automaticamente, mentre altri devono essere montati su richiesta

Condivisione dei file

- Nei moderni sistemi multiutente la condivisione di file è una funzionalità alla quale non è possibile rinunciare
- Essa introduce le problematiche di
  - Molteplicità degli utenti e metodi di condivisione
  - Condivisione di interi file system e file system remoti
  - Risoluzione di possibili conflitti su file condivisi

Utenti multipli (1)

- In un sistema multi-utente i file (e directory) possono essere condivisi secondo due modalità:
  - Il SO può concedere a **tutti** gli utenti l'accesso a **tutti** i file
  - Il SO può regolare l'accesso ai file tramite permessi
- Nel secondo caso il sistema deve memorizzare per ogni file una serie di attributi per la condivisione
- Nella maggior parte dei SO questi riguardano:
  - Proprietario del file, colui che può cambiare gli attributi
  - Gruppo di condivisione e permessi del gruppo
  - Permessi per il resto degli utenti

Utenti multipli (2)

- Definiti i permessi, ad ogni accesso al file il SO
  - Confronta l'ID del richiedente con gli attributi "proprietario" e "gruppo"
  - Decide di conseguenza che tipo di accesso al file concedere
- Queste problematiche sono gestite dagli algoritmi di controllo degli accessi e la protezione dei file

File system remoti (1)

- L'avvento delle reti a reso possibile la condivisione di dati tra sistemi distribuiti
- I metodi di condivisione dei file si sono migliorati con l'evoluzione dei protocolli di rete:
  1. Trasferimento dei file su richiesta tramite programmi FTP eseguiti dagli utenti
  2. File System Distribuito (DFS), permette al calcolatore locale di avere visibilità su directory remote e viceversa
  3. World Wide Web, esplorazione dei file tramite browser e operazioni per trasferirli (wrapper che nascondono all'utente l'uso di servizi FTP)

File system remoti (2)

- Un file system distribuito richiede una stretta integrazione tra il calcolatore che accede ai file e il calcolatore che li condivide
- Un modello di condivisione molto diffuso è il **client-server**
  - Client: chi richiede i file
  - Server: chi condivide i file, specificando eventuali permessi di accesso
- Richiede dei controlli di sicurezza
  - Il client potrebbe cercare di ingannare il server falsando il suo identificativo
- Tipicamente si usano autentificazioni reciproche basate su chiave di cifratura
  - Simmetrica
  - Asimmetrica

File system remoti (3)

- I sistemi moderni introducono dei file system di rete
  - Network File System NFS in Unix
  - Common Internet File System in Windows
- In questi file system sia il client che il server devono memorizzare lo stesso identificativo e la stessa password per un determinato utente
  - L'accesso viene concesso solo se ID e PSW corrispondono sia sul client che sul server
- Per rendere più facile la gestione dei sistemi distribuiti sul reti geografiche sono stati introdotti i DNS (Domain Name Systems)
  - Essi mappano il nome associato al calcolatore con il suo indirizzo di rete (IP)
  - In questo modo non è necessario conoscere gli IP dei server

File system remoti (4)

- I file system locali possono presentare malfunzionamenti dovuti
  - Al calcolatore: malfunzionamento dei dischi o dei bus
  - All'utente: errori di amministrazione dei file o rimozioni involontarie
- Che portano all'instabilità del sistema (crash)

- Nei file system remoti il problema si aggrava a causa della rete
  - Interruzione del canale di comunicazione
  - Crollo del server
- Questi problema portano all'interruzione dei comandi del DFS

File system remoti (5)

- Tipicamente i client gestiscono i guasti su file system remoti in modo diverso a quelli su file system locali:
  - Interrompono tutte le azioni sui file remoti, oppure
  - Posticipano le azioni al momento in cui il server tornerà disponibile (più diffusa)
- La seconda politica richiede che client e server mantengano un registro delle attività e dei file aperti
  - In modo da rendere semplice la ripresa delle attività interrotte dal guasto di rete

Semantica della coerenza (1)

- Specifica quando le modifiche operate da un utente su un file condiviso saranno visibili ad altri utenti

- Queste semantiche richiedono **algoritmi di sincronizzazione dei processi** simili a quelli già studiati
  - A causa della lunga latenza e bassa velocità di trasferimento delle connessioni di rete non è possibile usare gli stessi algoritmi

Semantica della coerenza (2)

- Semantica Unix
  - Esiste un'unica immagine del file e gli utenti condividono un puntatore alla sua locazione
  - Anche il puntatore di scrittura/lettura è unico
  - Le modifiche al file sono immediatamente visibili a tutti gli utenti
  - Gli utenti competono per eseguire le operazioni di lettura e scrittura
- Semantica delle sessioni (Andrew File System, AFS)
  - Le scritture di un file aperto non sono visibili immediatamente a tutti gli utenti
  - Vengono salvate solo al momento della chiusura e saranno visibili agli utenti che apriranno il file da quel momento in poi
  - Esistono quindi diversi immagini locali dello stesso file
  - È necessario gestire eventuali conflitti sulle scritture (sez.17.6.3)
- File condivisibili immutabili
  - Sono file che diventano di sola lettura al momento della condivisione

Protezione

- La protezione di un file system consiste in due grandi problematiche

- **L'affidabilità**
  - Si vuole che eventuali danni ai dispositivi di memorizzazione non comportino una perdita dei dati
  - È tipicamente gestita implementando delle politiche di backup
    - Automatiche e periodiche
    - Manuali

- **Il controllo degli accessi**
  - Si vuole determinare chi può accedere ai file e cosa può farne

Il controllo degli accessi (1)

- Gli accessi ai file possono essere controllati attraverso due politiche estreme
  - Accesso negato: protezione totale
  - Accesso completamente consentito: nessuna protezione
- Una politica più furba prevede di concedere dei permessi in base al tipo di accesso:
  - Lettura
  - Scrittura
  - Esecuzione
  - Aggiunta (di informazioni in coda al file)
  - Cancellazione
  - Elencazione (del nome e degli attributi di un file, o directory)
- Altri tipi di accessi possono essere garantiti come combinazione dei precedenti

Il controllo degli accessi (2)

- Definiti i tipi di accesso è necessario definire a quali utenti essi sono concessi e a quali sono negati
- Una prima soluzione è la **lista di controllo degli accessi** (ACL):
  - A ogni file si associa una ACL che definisce per ogni utente i diversi permessi di accesso
  - Problemi:
    - La compilazione di questa lista per ogni file può essere un compito oneroso
    - Questa lista ha una dimensione variabile (quindi la dimensioni degli attributi di ogni file non è più fissa)
      - Problemi di gestione dello spazio

Il controllo degli accessi (3)

- Una seconda soluzione è la definizione di tre **classi**:
  - Proprietario, Gruppo, Universo
- Ad ognuna di queste classi possono essere associati dei permessi diversi
  - In questo modo ogni file ha dei permessi di default (e.g. lettura per tutti, scrittura solo per il proprietario) e la lista non va compilata per ogni file
- Problema
  - Non è possibile definire accessi complessi
  - E.g. si supponga di avere una directory contenente i file sorgente di un progetto software avente i seguenti permessi
    - Il proprietario ha un controllo totale
    - Gli utenti del gruppo "developers" possono scrivere e leggere
    - Gli altri utenti possono solo leggere
    - Diventa difficile concedere ad un utente che non fa parte del gruppo "developers" l'accesso in scrittura ad uno specifico file sorgente

Il controllo degli accessi (4)

- Una terza soluzione, tipicamente adottata, combina le due soluzioni precedenti:
  - Ad ogni file sono associati i permessi relativi alle classi
  - In casi di necessità di controlli fini è possibile aggiungere delle ACL

- In caso di conflitti tipicamente la precedenza viene data alle ACL, essendo esse più dettagliate

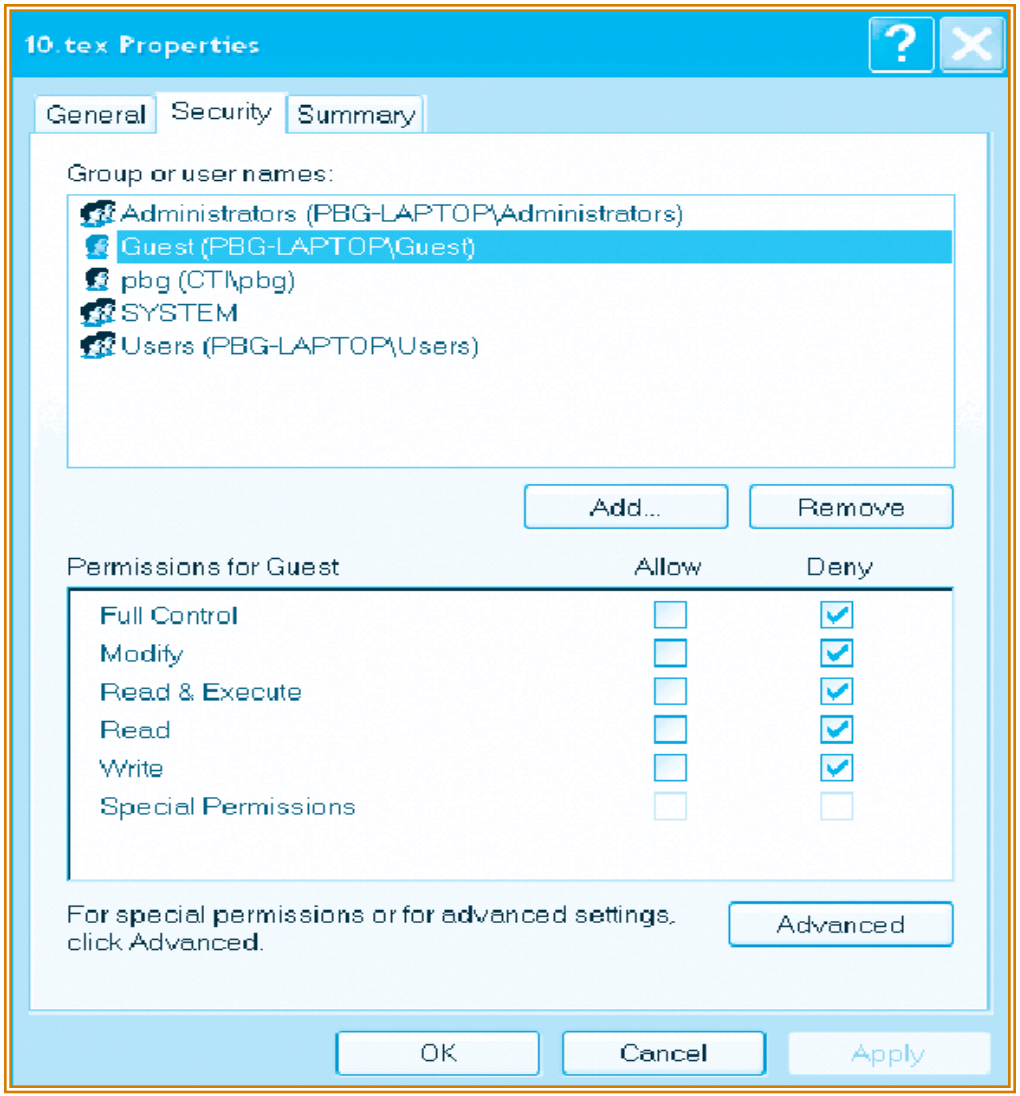
Il controllo degli accessi (5)

- In Unix
  - Si definiscono tre bit per ogni classe: `r, w, x`
  - I permessi possono essere associati con il comando `chmod`
    - E.g. `chmod 664 intro.ps` (per ogni gruppo 2^2`*r + `2^1`*w + `2^0`*x`)
  - Utente e gruppo di un file sono settati con i comandi `chown` e `chgrp`
    - E.g. `chown user1 intro.ps  chgrp staff intro.ps`

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

Il controllo degli accessi (6)

- In Windows
  - Un'interfaccia grafica permette di definire ACL
  - Per utenti e gruppi



Il controllo degli accessi (6)

- Altre problematiche di protezione
  - Protezione dei file tramite password
    - Richiede di ricordare un alto numero di password
    - Oppure di usare sempre quella, meno sicuro (scoperta una si ha accesso totale)
  - Protezione delle directory
    - Le directory richiedono controlli diversi rispetto ai file
    - Creazione di nuovi file, rimozione dei file, elencazione del contenuto
    - Possono essere usate tecniche simili a quelle adottate per i file