versione 1.0
20121204

Sistemi Operativi

II Facoltà di Ingegneria - Cesena
a.a 2012/2013
docente: Alessandro Ricci

[modulo 6a]
FILE SYSTEM

SOMMARIO

- Introduzione ai concetti di base
  - File, Directory, File System
- Struttura logica e struttura fisica di un file
  - record, blocchi
- Operazioni su file
  - Tabelle usate nella gestione dei file
  - metodi di accesso: sequenziale, diretto, ad indice
- Directory
  - struttura logica a livelli
- File System
  - mounting
  - esempi concreti

FILE SYSTEM: DEFINIZIONE

- Il file system è quella parte del S.O. che fornisce i meccanismi di accesso e memorizzazione delle informazioni (programmi e dati) allocate in memoria di massa
- Il file system realizza i concetti astratti di:
  - **file**, come unità *logica* di memorizzazione
  - **directory (direttorio)**, come insieme di file (e direttori)
  - **partizione**, come insieme di file associato ad un particolare dispositivo fisico (o porzione di esso)
- Le caratteristiche di file, directory e partizione sono del tutto indipendenti dalla natura e dal tipo di dispositivo utilizzato.
- Tra i tipi di file system noti sviluppati nel tempo:
  - FAT (Flat Allocation Table), file system di MSDOS
  - FAT32 e VFAT, versione moderna supportata da Windows 2000/XP
  - NTFS, file system introdotto con Windows NT
  - ext2, ext3, ext4 - extended file system introdotti con Linux
  - HFS+ (Mac OS X)
  - ZFS (Solaris)
  - Google File System
  - ...

L'ASTRAZIONE DI FILE

- Il file è un insieme di informazioni:
  - programmi
  - dati
  - testi
  - …

  rappresentati come insieme di **record logici** (bit, byte, linee, record, etc.)
- Ogni file è individuato da (almeno) un **nome simbolico** mediante il quale può essere riferito
  - ad esempio, nell'invocazione di comandi o system call
- Ogni file è caratterizzato da un insieme di **attributi**

ATTRIBUTI DI UN FILE

- A seconda del sistema operativo, i file possono avere **attributi** diversi
- Solitamente come attributi abbiamo:
  - tipo: stabilisce l'appartenenza a una classe (eseguibili, batch, testo, etc)
  - dimensione: numero di byte contenuti nel file data e ora (di creazione e/o di modifica)
  - data di creazione, di modifica,..
  - indirizzo: puntatore/i a memoria secondaria
- In S.O. multiutente (e.g. UNIX) fra gli attributi abbiamo anche:
  - utente proprietario
  - protezione: diritti di accesso al file per gli utenti del sistema
- Definiti anche *meta-dati*

TIPI DI FILE PIU' COMUNI

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

FILE CONTROL BLOCK E FILE DESCRIPTOR

- I meta-dati di un file sono memorizzati in apposite strutture in memoria secondaria e caricati in memoria centrale quando un file viene acceduto
  - in generale vengono chiamati **File Control Block (FBC)**
  - nei sistemi UNIX si chiamano **inode**
- Per **descrittore di file** *(file descriptor)* si intende invece generalmente un numero intero non negativo con cui a livello user è possibile identificare e manipolare un file
  - utilizzato a livello di API
  - in alcuni sistemi (es. Windows) prende il nome di **file handle**

ESEMPIO FCB: INODE IN UNIX

- In UNIX i FCB prendono il nome di **i-node**
  - tra gli attributi contenuti in un i-node: tipo di file (ordinario, direttorio, file speciale, per i dispositivi), proprietario, gruppo (user-id, group-id), dimensione, data, 12 bit di protezione...
  - con i-number si indica l'indice dell'i-node nella i-list
- Ogni **inode** è identificato da un identificatore intero chiamato **inumber** (inode number)
  - ogni file ha il proprio inumber
- E' possibile avere informazioni circa l'inode di un file via shell mediante alcuni comandi, tra cui `ls` con opzione `-i`

STRUTTURA INTERNA DI UN FILE

- Dal punto di vista fisico, ogni dispositivo di memorizzazione secondaria viene partizionato in **blocchi** (o **record fisici**)
  - Il blocco è l'unità di trasferimento nelle operazioni di I/O da/verso il dispositivo
  - la sua dimensione è fissa
- Dal punto di vista logico invece, l'utente vede il file come un insieme di **record logici**
  - il record logico è unità di trasferimento nelle operazioni accesso al file (es. lettura, scrittura di blocchi)
  - la sua dimensione può variare
- Uno dei compiti del S.O. per ciò che concerne il file system è stabilire una corrispondenza tra record logici e blocchi
  - di solito la dimensione del blocco è molto maggiore della dimensione del record logico, per cui il sistema operativo si occupa di "impaccare" più record logici all'interno di blocchi
  - problemi di frammentazione e spreco memoria secondaria
    - descrizione nel modulo 6b

OPERAZIONI SUI FILE

- Compito del S.O. (del file system) è consentire l'accesso ai file mediante un certo insieme di operazioni
- Le tipiche operazioni che caratterizzano tale accesso sono:
  - **creazione**
    - allocazione di un file in memoria secondaria ed inizializzazione dei suoi attributi.
  - **lettura**
    - leggere in memoria centrale record logici dal file
  - **scrittura**
    - inserimento di nuovi record logici all'interno di file
  - **cancellazione**
    - eliminazione del file dal file system

TABELLA FILE ATTIVI E FILE APERTI

- Ogni operazione di per sé richiederebbe la localizzazione di informazioni su disco
  - ad esempio gli indirizzi dei record logici a cui accedere, gli altri attributi del file, i record logici stessi
- Questo comporterebbe un costo molto elevato in caso di gruppi di operazioni
- Allo scopo, per migliorare l'efficienza il S.O. mantiene in memoria centrale due strutture (tabelle)
  - la **tabella dei file attivi**
    - contiene i FCB dei file che sono attualmente in uso
  - la **tabella dei file aperti**
    - tiene traccia delle informazioni relative ad un file aperto
      - lo stesso file può essere aperto da più processi, per cui ha uno slot nella tabella dei file attivi e più slot nella tabella dei file aperti
    - fra le informazioni in ogni entry della tabella c'è il puntatore alla entry nella tabella dei file attivi

APERTURA E CHIUSURA DI UN FILE

- Prima di poter accedere il contenuto di un file, è necessario *aprire* il file e poi *chiuderlo* esaurite tali operazioni:
  - **l'apertura** comporta l'introduzione di un nuovo elemento nella tabella del file aperti ed eventualmente dei file attivi nel caso in cui non sia la prima volta in cui viene aperto
  - **chiusura** comporta il salvataggio del file in memoria secondaria e eliminazione dell'elemento corrispondente dalla tabella dei file aperti
- Spesso il contenuto (parziale o totale) dei file aperti viene temporaneamente copiato in memoria centrale (**memory mapping**), in modo da ottenere accessi più veloci
  - nasce un problema di consistenza fra le informazioni relative al contenuto del file in memoria secondaria ed in memoria centrale
  - l'operazioni di **flushing** è l'operazione con cui si forza l'aggiornamento delle informazioni su memoria secondaria a partire da quelle in memoria centrale

STRUTTURE PER FILE APERTI

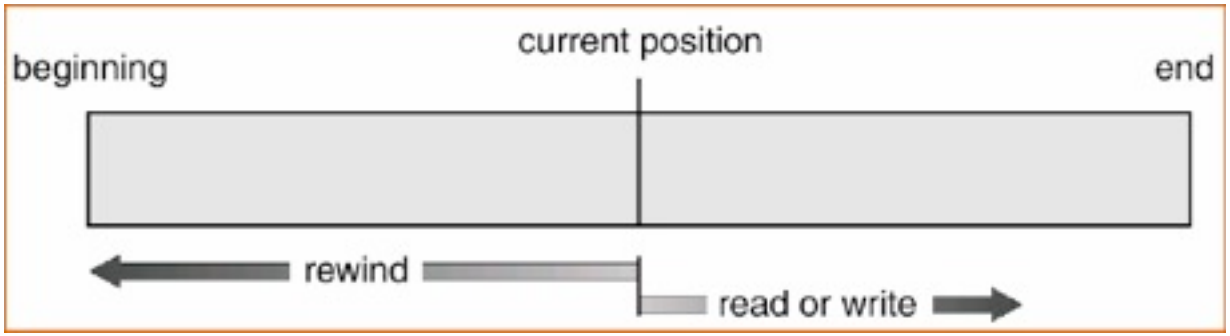
- Per ogni file aperto, tipicamente i S.O. mantengono come informazioni necessarie per la gestione:
  - **file pointer**
    - (per ogni processo) punta all'ultima locazione (indice del record logico) in c'è stata lettura / scrittura
    - ogni volta che avviene una lettura o scrittura il file pointer viene aggiornato
  - **file-open count**
    - contatore del numero di volte in cui è stato aperto un file, utile per rimuovere l'elemento dalla tabella dei file aperti quando l'ultimo chiude
  - **diritti di accesso**
    - per ogni processo, specifica la modalità di accesso
- Locking
  - in alcuni sistemi operativi è presente un meccanismo di locking che permette di avere mutua esclusione nell'accesso ai file
  - tipi di locking
    - **mandatory**
      - l'accesso è impedito / garantito a seconda di chi detiene il locking
    - **advisory**
      - lo stato di locking è solo esposto, e i processi possono decidere cosa fare

METODI DI ACCESSO

- L'accesso a file può avvenire secondo varie modalità:
  - accesso **sequenziale**
  - accesso **diretto** (random access)
  - accesso a **indice**
- Il metodo di accesso è indipendente sia dal tipo di dispositivo utilizzato, sia dalla tecnica di allocazione dei blocchi in memoria secondaria.

ACCESSO SEQUENZIALE

- In questa modalità di accesso, Il file è una **sequenza** [R1, R2,.. RN] di record logici e per accedere ad un particolare record logico Ri , è necessario accedere prima agli (i-1) record che lo precedono nella sequenza:
- in questo caso le operazioni di accesso sono del tipo:
  - *read next* lettura del prossimo record logico della sequenza
  - *write nex*t: scrittura del prossimo record logico
  - *rewind* per tornare al primo record logico
- Ogni operazione di accesso (lettura/scrittura) posiziona il puntatore al file sull'elemento successivo a quello letto/scritto



ACCESSO DIRETTO

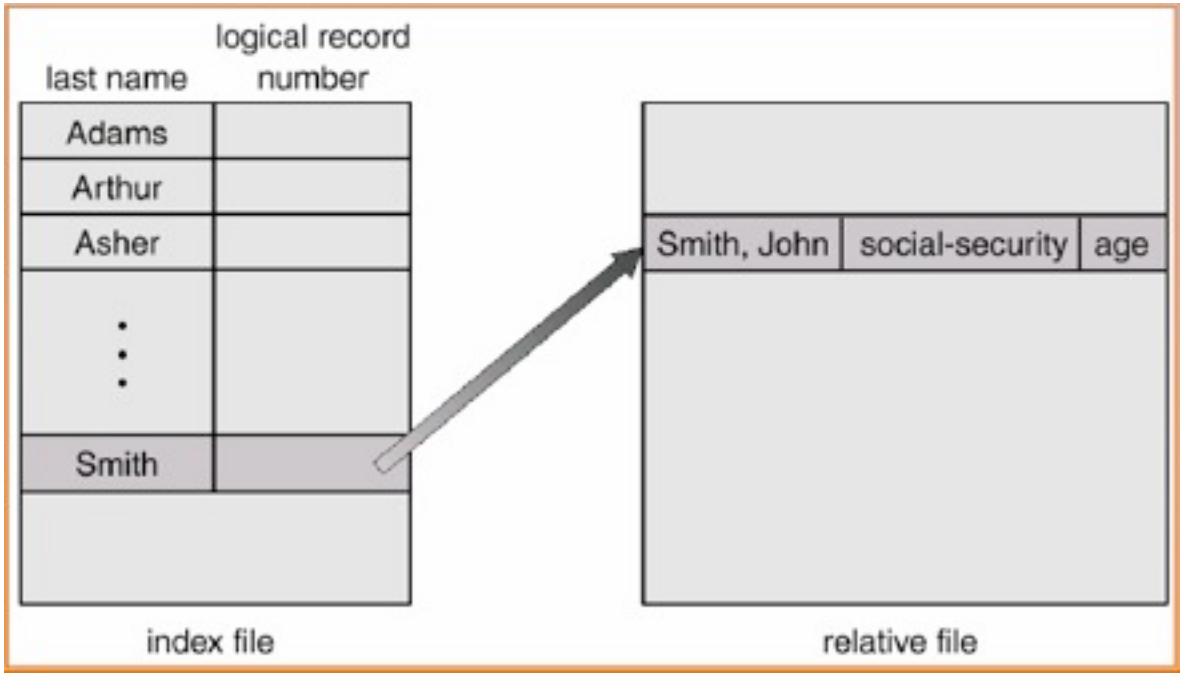
- In questa modalità di accesso il file è un insieme non ordinato {R1, R2,.. RN} di record logici numerati: si può accedere direttamente ad un particolare record logico specificandone il numero.
- In questo caso le operazioni di accesso sono del tipo:
  - *read i*: lettura del record logico i
  - *write i*: scrittura del record logico i
- Questa modalità di accesso è utile quando si vuole accedere a grossi file per estrarre/aggiornare poche informazioni (es. database)

sequential access	implementation for direct access
reset	*cp = 0;*
read next	*read cp;* *cp = cp+1;*
write next	*write cp;* *cp = cp+1;*

Implementazione dell'accesso sequenziale utilizzando l'accesso diretto

ACCESSO AD INDICE

- Infine, nella modalità accesso ad indice ad ogni file viene associata una struttura dati (ancora un file) contenente l'indice delle informazioni contenute nel file
- Per accedere a un record logico, si esegue una ricerca nell'indice (utilizzando una chiave):



DIRECTORY

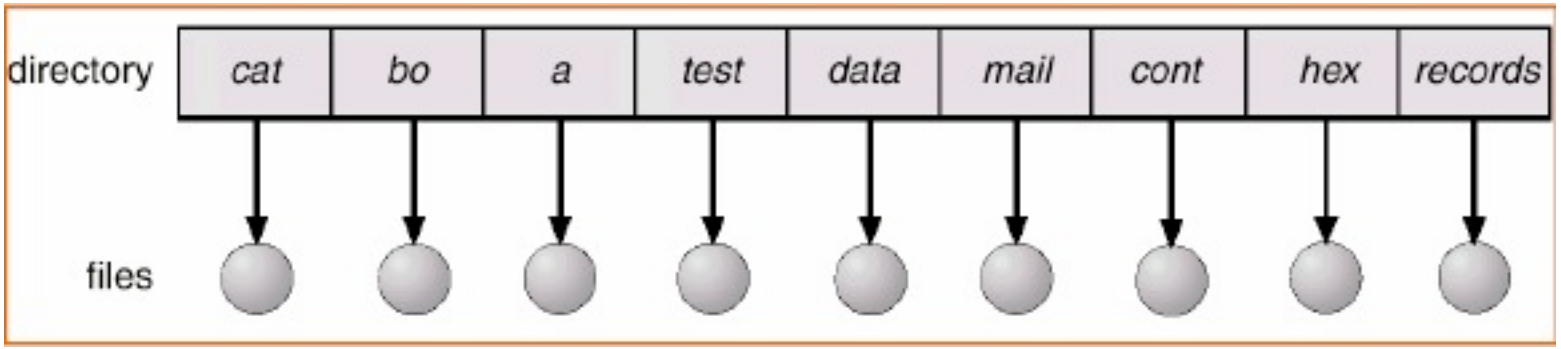
- La directory (o direttorio) è lo strumento per organizzare i file all'interno del file system:
  - una directory può contenere più file
  - è realizzato mediante una struttura dati che associa al nome di ogni file (in esso contenuto) la posizione del file nel disco.
- Obiettivi
  - localizzazione efficiente di file
  - raggruppamento logico dei file a partire dalle loro proprietà
- Le operazioni classiche sulle directory sono:
  - creazione e cancellazione di directory
  - aggiunta/cancellazione di file
  - listing: elenco di tutti i file contenuti nella directory
  - attraversamento della directory
  - ricerca di file nella directory

STRUTTURA LOGICA DELLE DIRECTORY

- La struttura logica del direttorio può variare a seconda del Sistema Operativo
- Gli schemi più comuni sono:
  - **a un livello**
  - **a due livelli**
  - **ad albero**
  - **a grafo aciclico**

STRUTTURA AD UN LIVELLO

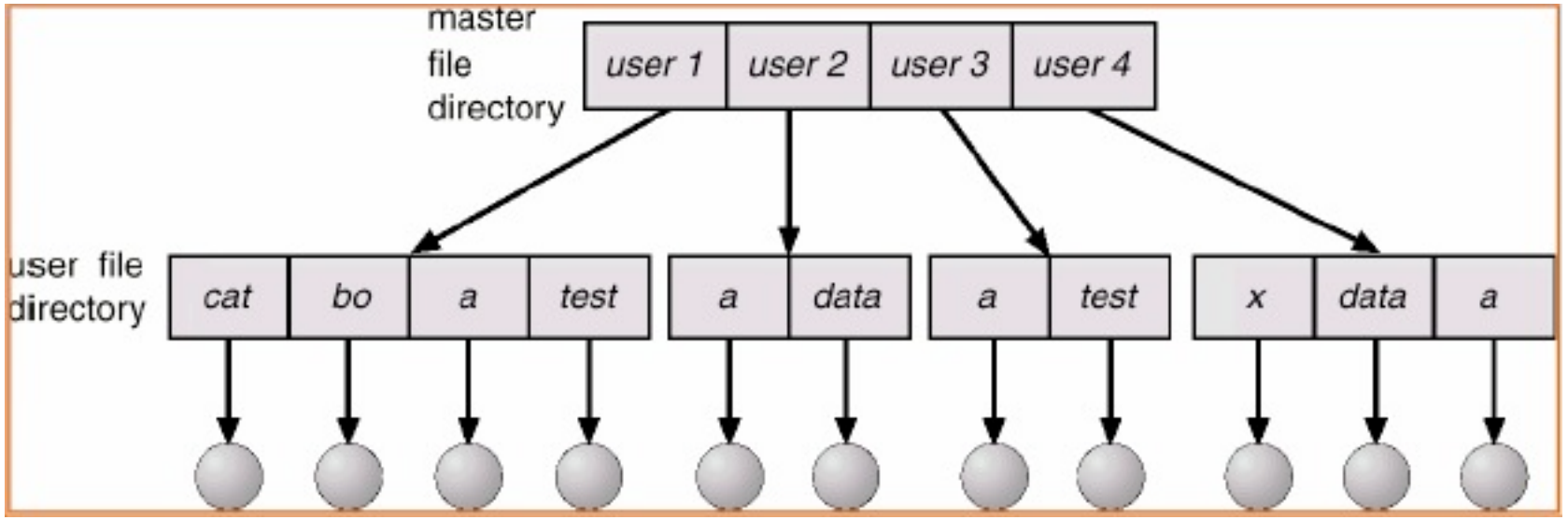
- Nella struttura ad un livello c'è una sola directory per ogni file system



- Problemi:
  - unicità dei nomi
  - multiutenza: come separare i file dei diversi utenti?

STRUTTURA A DUE LIVELLI

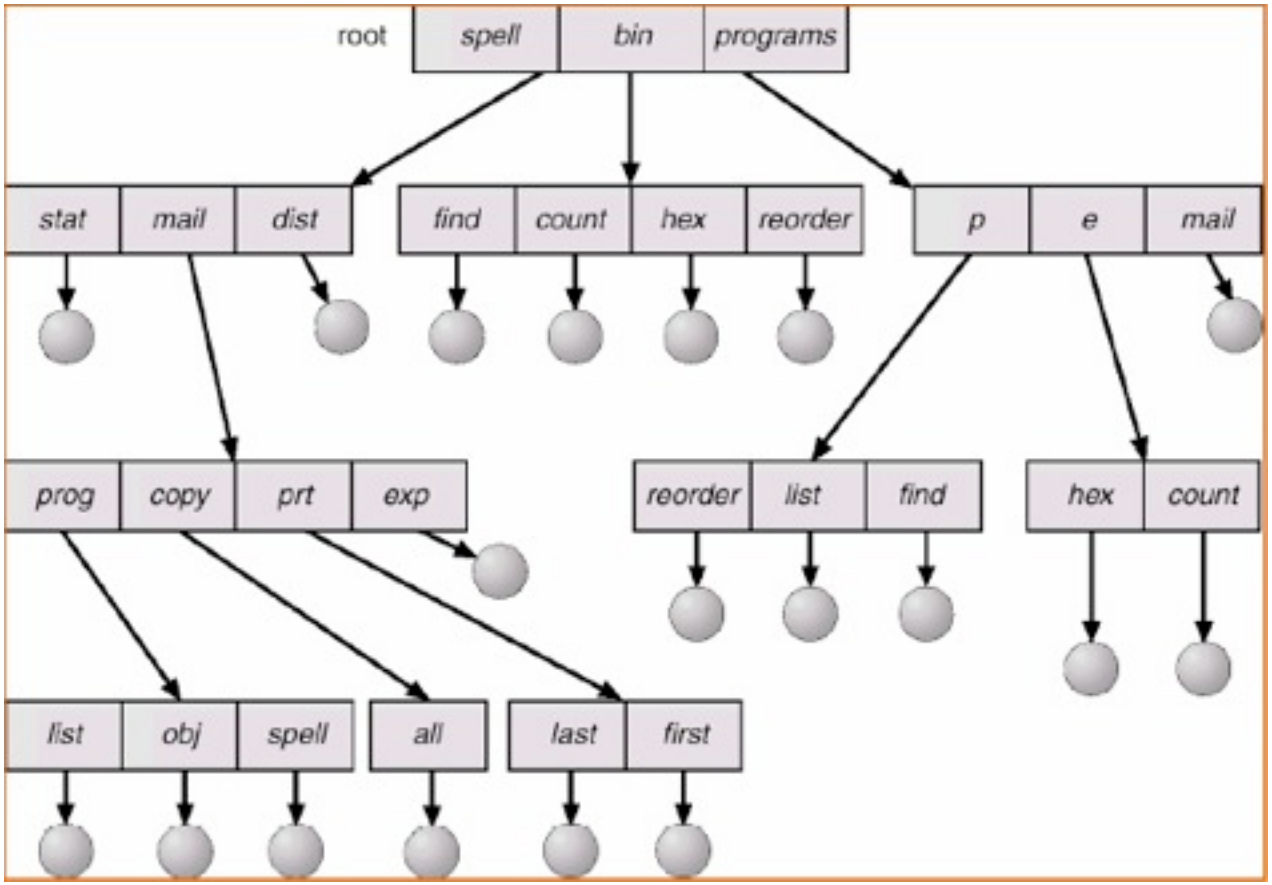
- In questo caso il file system è strutturato su due livelli:
  - il primo livello (directory principale) contiene una directory per ogni utente del sistema
  - il secondo livello contiene le directory degli utenti (a un livello)



- Risolto il problema multi-utenza, ma rimangono problemi relativamente alla mancanza di raggruppamento
- Inoltre, come condividere file fra utenti diversi?

STRUTTURA AD ALBERO (1/2)

- La struttura ad albero è una generalizzazione della precedente soluzione, consentendo una organizzazione gerarchica a N livelli
  - ogni direttorio può contenere file e altri direttori

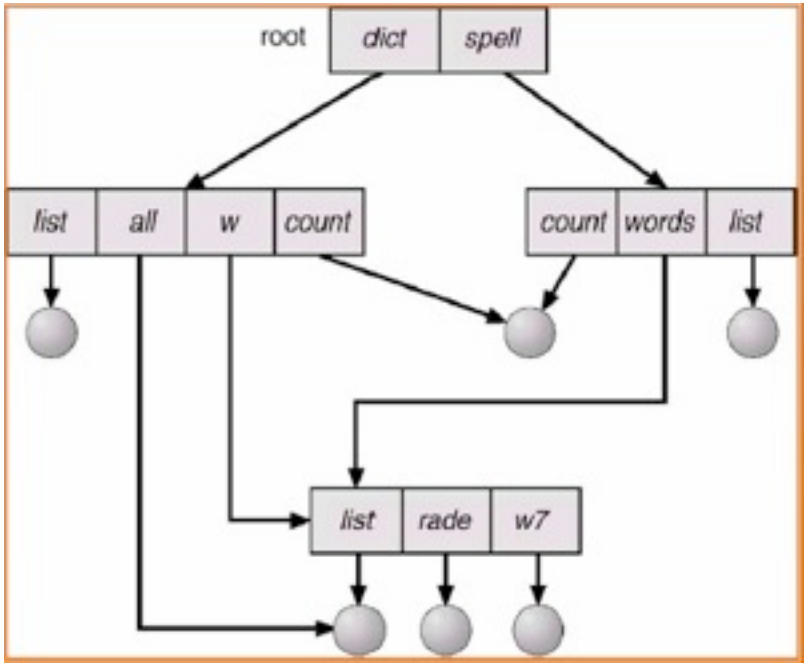


STRUTTURA AD ALBERO (2/2)

- La soluzione ad albero permette di avere ricerche efficienti e di poter raggruppare in modo flessibile i file
  - i file all'interno di strutture ad alberi sono riferiti mediante nomi simbolici detti **pathname**
    - un path-name **assoluto** riferisce il file specificando il percorso (insieme delle directory) a partire dalla radice e il nome del file
      - Ex: /usr/aricci/docs/article.pdf
    - un pathname **relativo** riferisce il file specificando il percorso (insieme delle directory) a partire dalla directory corrente e il nome del file
      - Ex: docs/article.pdf supponendo che la directory corrente sia /usr/aricci
- Tuttavia rimane il problema per cui due utenti non possono condividere direttamente un file

STRUTTURA A GRAFO

- La generalizzazione del caso precedente consiste nell'avere una struttura a grafo
  - quindi con nodi (directory) che possono condividere figli (file o directory)



- In questo modo è possibile avere la condivisione di file, riferiti con nomi (pathname) diversi
  - UNIX link file (in seguito)

FILE E DIRECTORY IN UFS

- Nello UNIX File System l'astrazione di file viene utilizzata - oltre all'accezione ordinaria di contenitore di informazioni - per rappresentare in modo uniforme ogni risorsa con cui può interagire l'utente, in particolare:
  - **directory** (*directory files)*
    - tengono traccia delle informazioni di una directory
    - attributo 'd'
    - file ".." = parent, "." corrente
  - **link** (*link files*)
    - rappresentano collegamenti ad file locati in un'altra posizione del file system
    - definiti anche **soft link** o **symbolic link**
    - attributo 'l'
  - **dispositivi** (*device files)*
    - tutti i dispositivi di I/O (stampanti, disk driver, periferiche usb,terminali,...) sono mappati in file presenti della directory ***/dev***
    - in questo modo vi si interagisce in modo uniforme mediante operazioni quali open, read, write, close...
    - attributi speciali del descrittore di file

ESEMPI DI DEVICE FILE

- Dischi: **/dev/hd??**
  - /dev/hda, /dev/hdb, /dev/hdc
  - partizioni: /dev/hda1, /dev/hda2, ...
- Floppy: **/dev/fd?**
  - /dev/fd0 (floppy disk a:), /dev/fd1,...
- CDROM: **/dev/cdrom**
- Dischi SCSI: **/dev/sd??**
  - /dev/sda, /dev/sdb, /dev/sdc
  - partizioni: /dev/sda1, /dev/sda2, ...
- Porte seriali: **/dev/ttyS?**
  - /dev/ttyS0 (COM1), /dev/ttyS1,...
- Parallele: **/dev/par?**
  - /dev/par0,...
- Stampante: **/dev/lp?**
  - /dev/par1,...

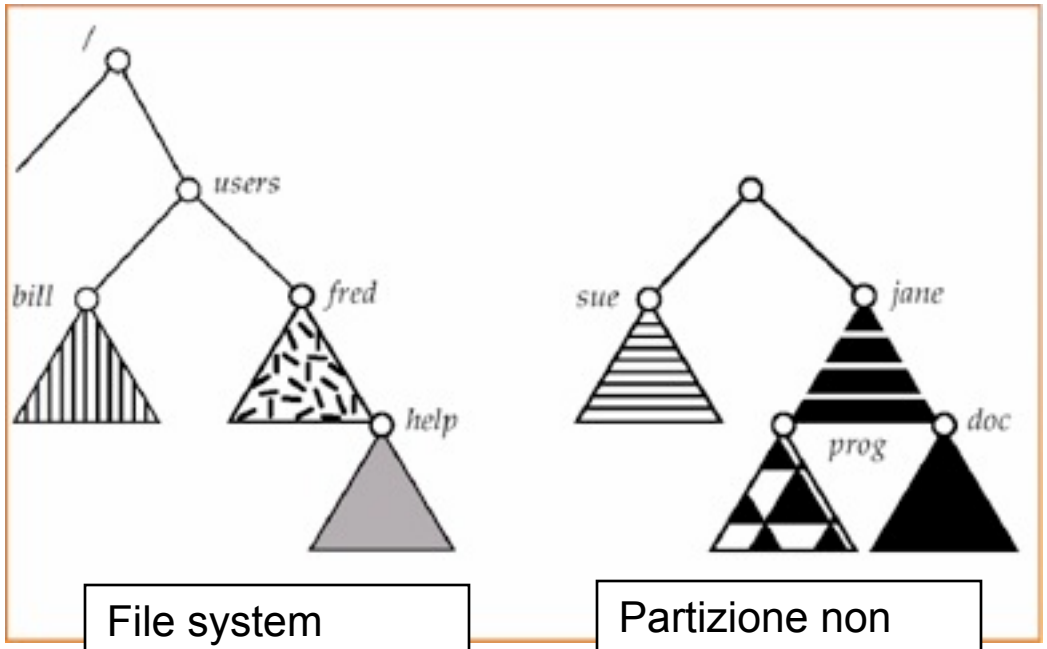
DEVICE FILE SPECIALI

- STANDARD INPUT, OUTPUT, ERROR: **/dev/stdin, /dev/stdout/ stderr**
  - rappresentano il corrente stdin, stdout, stderr
- NULL device: **/dev/null**
  - funge da ‘buco nero’, utile per raccogliere output che si vuole cartare
- RANDOM device: **/dev/random**
  - generatore di numeri casuali
- ZERO device file: **/dev/zero**
  - funge da generatore di byte uguali a 0
  - da usare in combinazione con il comando **dd**

MOUNTING DI UN FILE SYSTEM

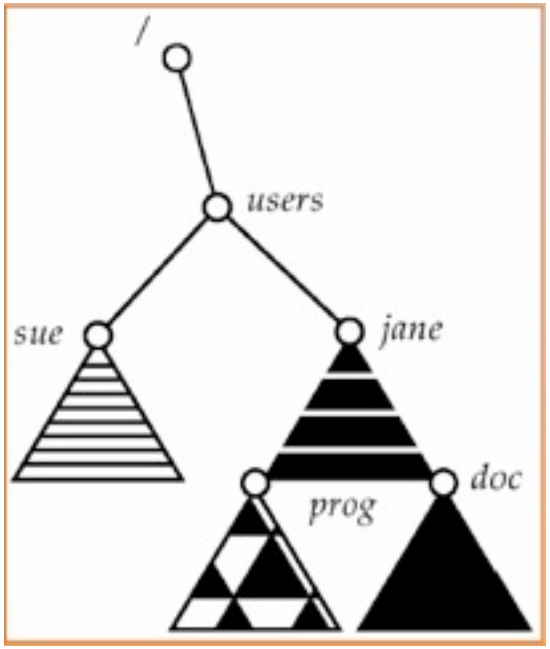
- Nei sistemi operativi moderni un file system deve essere “montato” (**mounted**) prima di poter essere disponibile ai processi del sistema
  - analogamente al fatto che un file debba essere aperto prima di essere usato
- Un file system non ancora montato viene montato in un cosiddetto punto di mount (**mount point**)
  - tipicamente una directory vuota
- Un file system montato può essere poi successivamente smontato, con una operazione di unmounting

MOUNTING DI UN FILE SYSTEM: UNIX



File system esistente

Partizione non montata, ad esempio residente in /devices/dsk



Punto di mount

MOUNTING NEI SISTEMI UNIX

- Per fare il mounting di un file system si utilizza il comando mount, specificando device / partizione da montare e punto di mounting
  - tra le opzioni anche il tipo di file system
  - es: mount di un floppy (device /dev/fd0 come file system floppy con punto di mount /mnt/floppy) e copia di file

    ```
    $ mount –t ext2 /dev/fd0 /mnt/floppy
    $ cp ~/src/* /mnt/floppy/*
    ```
  - Per l'unmounting esiste il comando unmount
- Durante il processo di boot i file system elencati nel file /etc/fstab sono automaticamente montati
  - il file contiene una lista di linee di testo in cui si specifica il device da montare, il punto di mounting, il tipo di file system e varie opzioni
  - I file system già montati si trovano invece nel file /etc/mtab.

ROOT FILE SYSTEM IN UNIX: /

- File system montato al boot, contiene utilities e files fondamentali, organizzate nelle seguenti directory standard:
  - **bin**
    - contiene i comandi e utility essenziali per amministratori e utenti, necessari prima ancora che sia montato qualsiasi altro file system
    - Esempi di utility: cat, cp, ls, mkdir, sh,...
  - **boot**
    - file necessari per il boot
  - **dev**
    - contiene i dispositivi del sistema rappresentati da opportuni file
  - **etc**
    - file di configurazione del sistema. Esempi
      - fstab file: informazioni sui file system presenti
      - profile file: file inizializzazione shell sh
  - **lib**
    - librerie condivise essenziali
  - **usr**
    - directory condivisa fra tutti gli utenti, con informazioni di sola lettura

ROOT FILE SYSTEM IN UNIX: /

- (directories - segue):
  - **var**
    - Contiene file dal contenuto dinamico, come informazioni circa le sessioni aperte, cache,...
      - /var/accounts: informazioni di log sugli utenti
      - /var/cache: informazioni cached delle applicazioni
      - ...
  - **sbin**
    - contiene utility di sistema, utilizzabili esclusivamente dall'amministratore
  - **proc**
    - file system che contiene in forma di file informazioni sui processi in esecuzione
  - **media**
    - punto di mount per i media removibili
  - **mnt**
    - punto di mount per file-systems temporanei
  - **opt**
    - add-ons e packages per applicazioni
  - **tmp**
    - contiene file temporanei creati dai programmi e dal sistema

/usr

- Contiene dati shareable, di sola lettura
- Directories
  - **/usr/bin**
    - contiene la maggior parte dei comandi utente
  - **/usr/include**
    - header file inclusi dai programmi C
  - **/usr/local**
    - utilizzata dagli amministratori per installare programmi in locale
  - **/usr/sbin**
    - utility non essenziali per amministratori
  - **/use/share**
    - contiene file dati di programmi o package

FILE SYSTEM IN WINDOWS

- I S.O. della famiglia Windows implementano una versione estesa della struttura a 2 livelli
  - device (dischi) e partizioni al primo livello, denotate con un identificatore di drive del tipo a: c: d: (drive letter)
  - partizioni strutturate in termini di grafi
- Il path per uno specifico file si ottiene specificando

  drive-lettera:\path\to\file
- Il S.O. scopre automaticamente tutti i device e monta tutti i file system localizzati al boot time
  - al contrario dei S.O. della famiglia Unix, ove invece il mounting è esplicito, mediante opportuni comandi (e chiamate di sistema)

FILE SHARING

- La condivisione di file è un aspetto importante per i sistemi multiutente
  - User ID identificano utenti, permettendo di specificare permessi e protezione a livello di utente
  - Group ID permette di definire permessi a livello di gruppi
- Nei sistemi distribuiti i file possono essere condivisi fra host remoti, mediante opportune architetture e protocolli
  - fra gli altri, Network File System (NFS) è un metodo molto diffuso per avere condivisione di file system distribuiti
- Modello client-server: clients possono montare file system remoti da server
  - **NFS** è il protocollo standard nel mondo Unix per condivisione file modello client server
  - **CIFS** è il protocollo standard lato Windows
  - le system call standard sono tradotte in chiamate remote
- I *distributed information system* (distributed naming services) sono infrastrutture che implementano un accesso unificato alle informazioni remote
  - **LDAP** (Light-weight directory-access protocol), DNS, NIS
  - **Active directory** è l'implementazione sotto Windows di LDAP

CONSISTENZA

- Nel caso di file sharing (eventualmente distribuito) deve essere definita una semantica che specifichi la politica per più utenti che accedono simultaneamente a file condivisi
  - relazione con algoritmi di sincronizzazione dei processi
- In generale si definisce **file session** una sessione di lavoro su un file da parte di un utente
  - l'insieme degli accessi di un utente su un file compresi fra l'apertura e la chiusura del file
- Esistono varie semantiche di consistenza. Esempi:
  - UNIX
    - la scrittura su un file condiviso da parte di un utente che ha aperto una sessione sul file è immediatamente visibile a tutti gli altri utenti che hanno anch'essi una sessione aperta
  - **Andrew File System (AFS):** *Semantica di sessione*
    - AFS è un file system distribuito, studiato nella ricerca
    - la scrittura su un file condiviso in una sessione non è immediatamente visibile a tutte le altre sessioni.
    - I cambiamenti sono visibili sono dopo la chiusura del file, per le nuove sessioni

PROTEZIONE

- Nei sistemi multi-utente / multi-processo, c'è la necessità di regolare / controllare l'accesso ai file da parte di utenti / processi distinti
- Possibili operazioni da controllare
  - read
  - write
  - execute
  - append
  - delete
  - list
- Metodo di controllo dell'accesso più utilizzato: **Access Control List** (ACL)
  - per ogni risorsa (file) si specifica una lista che descrive *chi* (user o gruppo) ha il permesso di fare *cosa* (operazione) sulla risorsa stessa

ACCESS CONTROL IN UNIX

- Nei sistemi UNIX si utilizza una strategia ACL semplificata, raggruppando l'insieme dei possibili utenti in tre categorie
  - proprietario (owner)
  - membri del gruppo del proprietario (group)
  - tutti gli altri utenti (others o universe)
- Per ogni file (incluse le dir) si tiene traccia di tre campi, che descrivono i permessi per ogni categoria di utenti relativamente all'esecuzione di tre tipi di operazioni
  - lettura (read)
  - scrittura (write)
  - esecuzione (execute)
- In particolare i permessi possono essere specificati da una tripletta RWX di valori ottali (prefisso "0"), ogni valore composto da tre bit che indicano i permessi relativi alle tre categorie (Owner, Group, Others)
  - bit a 1: permesso concesso, a 0: permesso non concesso
  - esempi
    - 0666 = 110110110 = permesso di lettura e scrittura per tutti
    - 0550 = 101101000 = permesso di lettura ed esecuzione per proprietario e gruppo

AFFIDABILITA’

- I file system più moderni forniscono supporti per aumentare l’affidabilità nella scrittura dei file e più in generale nella gestione del file system
  - al fine di evitare, ad esempio, la perdita di dati in caso di crash del sistema e più in generale preservare l’integrità dei dati
- Sono chiamati **log-based transaction-oriented** o **journaling file-system**
  - applicano tecniche proprie dei database
  - ogni scrittura su disco è interpretata dal file system come una transazione.
    - uso di log file
- Fra i file system dotati di journaling: NTFS, ext3, ext4, Journaled File System (JFS), HFS+

ESEMPI DI FILE SYSTEMS

- NTFS
- ext, ext2, ext3, ext4
- FUSE
- ZFS

NOTA: STANDARD BINARY PREFIXES

- Used to denote file or memory sizes
- In 1999, following recommendations by the International Union of Pure and Applied Chemistry (IUPAC) in 1995 and National Institute of Standards and Technology (NIST), the standards organization known as the International Electrotechnical Commission (**IEC**) adopted a set of distinct prefixes (cf. IEC 60027), e.g., kibi (symbol Ki, from kilobinary)[2] and mebi (symbol Mi, from megabinary), to indicate **binary multipliers.**
- This system uses the multiplier 1024, rather than 1000 as in the SI system, to arrive at successively larger prefixes.
  - within this recommendation, the SI prefixes should only be used in the decimal sense: kilobyte and megabyte denote one thousand bytes and one million bytes respectively, while kibibyte and mebibyte denote 1024 bytes and 1048576 bytes respectively.
  - this recommendation has since been adopted by leading national and international standards bodies.
  - despite the standardization of the new binary prefixes and the availability of unambiguous representations, they have seen limited adoption in practice;
    - the use of K (or k), M and G as binary multipliers when denoting the capacity of solid−state memory like random access memory (RAM) remains ubiquitous industry practice.

STANDARD BINARY PREFIXES

IEC prefix		Representations				SI prefix names in binary use	
Name	Symbol	Base 2	Base 1024	Value	Base 10	Name	Symbol
kibi	Ki	2^{10}	1024^{1}	1 024	$\sim 1.02 \times 10^{3}$	kilo	k, K
mebi	Mi	2^{20}	1024^{2}	1 048 576	$\sim 1.05 \times 10^{6}$	mega	M
gibi	Gi	2^{30}	1024^{3}	1 073 741 824	$\sim 1.07 \times 10^{9}$	giga	G
tebi	Ti	2^{40}	1024^{4}	1 099 511 627 776	$\sim 1.10 \times 10^{12}$	tera	T
pebi	Pi	2^{50}	1024^{5}	1 125 899 906 842 624	$\sim 1.13 \times 10^{15}$	peta	P
exbi	Ei	2^{60}	1024^{6}	1 152 921 504 606 846 976	$\sim 1.15 \times 10^{18}$	exa	E
zebi	Zi	2^{70}	1024^{7}	1 180 591 620 717 411 303 424	$\sim 1.18 \times 10^{21}$	zetta	Z
yobi	Yi	2^{80}	1024^{8}	1 208 925 819 614 629 174 706 176	$\sim 1.21 \times 10^{24}$	yotta	Y

NTFS

- New Technology File System (NTFS)
  - Standard file system of Windows NT, including its later versions Windows 2000, Windows XP, Windows Server 2003, Windows Server 2008, Windows Vista, and Windows 7
  - NTFS supersedes the FAT file system as the preferred file system for Microsoft's Windows operating systems.
- Several improvements over FAT and HPFS (High Performance File System)
  - improved support for metadata and the use of advanced data structures to improve performance, reliability, and disk space utilization, plus additional extensions such as security access control lists (ACL) and file system journaling.
- Recent versions
  - v3.1 from Windows XP (autumn 2001; "NTFS V5.1"), Windows Server 2003 (spring 2003; occasionally "NTFS V5.2"), Windows Vista (mid-2005) (occasionally "NTFS V6.0") and Windows Server 2008 V1.0 and V1.1 (and newer)
  - Windows Vista introduced **Transactional NTFS**, NTFS symbolic links, partition shrinking and *self-healing* functionality though these features owe more to additional functionality of the operating system than the file system itself.
- Maximum File Size: about **16 TiB**

EXT, EXT2

- **ext** (extended file system)
  - implemented in April 1992 as the first file system created specifically for the Linux operating system.
  - It has metadata structure inspired by the traditional Unix File System (UFS)
  - designed to overcome certain limitations of the Minix file system.
  - It is the first of the extended file systems
- **ext2** or second extended filesystem (1993)
  - replacement for the extended file system (ext).
  - the canonical implementation of ext2 is the ext2fs filesystem driver in the Linux kernel.
  - ext2 was the default filesystem in several Linux distributions, including Debian and Red Hat Linux, until supplanted more recently by ext3,
  - ext2 is still the filesystem of choice for flash-based storage media (such as SD cards, SSDs, and USB flash drives) since its lack of a journal minimizes the number of writes
  - limits
    - Block size: 1KiB =>Max file size: 16 GiB and Max filesystem size: 4 TiB
    - Block size: 2KiB =>Max file size: 256 GiB and Max filesystem size: 8 TiB
    - Block size: 4KiB =>Max file size: 4 TiB and Max filesystem size: 64 TiB
    - Block size: 8KiB =>Max file size: 64 TiB and Max filesystem size: 32 TiB

EXT3, EXT4

- The **ext3** or third extended filesystem (~1998..2001)
  - journaled file system that is commonly used by the Linux kernel.
  - It is the default file system for many popular Linux distributions.
  - the filesystem was merged with the mainline Linux kernel in November 2001 from 2.4.15 onward.
  - Its main advantage over ext2 is journaling which improves reliability and eliminates the need to check the file system after an unclean shutdown.
  - limits
    - Block size: 1KiB =>Max file size: 16 GiB and Max filesystem size: 2 TiB
    - Block size: 2KiB => Max file size: 2 TiB and Max filesystem size: 16 TiB
    - Block size: 8KiB => Max file size: 2 TiB and Max filesystem size: 32 TiB
- The **ext4** or fourth extended filesystem
  - journaling file system developed as the successor to ext3.
  - It was born as a series of backward compatible extensions to remove 64-bit storage limits and add other performance improvements to ext3.
  - Kernel 2.6.28, containing the ext4 filesystem, was finally released on December 2008

FUSE FILE SYSTEM

- **FUSE**
  - File system in User SpacE
- loadable kernel module for Unix-like computer operating systems, that allows non-privileged users to create *their own file systems* without editing the kernel code
  - this is achieved by running the file system code in user space, while the FUSE module only provides a "bridge" to the actual kernel interfaces
- Particularly useful for writing virtual file systems
  - unlike traditional filesystems, which essentially save data to and retrieve data from disk, virtual filesystems do not actually store data themselves.
  - they act as a view or translation of an existing filesystem or storage device.
  - in principle, any resource available to FUSE implementation can be exported as a file system.
- Available for Linux, FreeBSD, NetBSD (as PUFFS), OpenSolaris, and Mac OS X
  - officially merged into the mainstream Linux kernel tree in kernel version 2.6.14.

ZFS

- **Zetta-File System**
- A combined file system and logical volume manager designed by Sun Microsystems
- The features of ZFS include support for **high storage capacities**, integration of the concepts of filesystem and volume management, snapshots and copy-on-write clones, continuous integrity checking and automatic repair, RAID-Z and native NFSv4 ACLs
- ZFS is implemented as open-source software, licensed under the Common Development and Distribution License (CDDL).
- ZFS is a **128-bit file system**, so it can address 18 quintillion (1.84 × 10^19) times more data than current 64-bit systems.
- Limits
  - 16 EiB (2^64 bytes) — Maximum size of a file system
  - **16 EiB** — Maximum size of a single file