

Il file system

File system

- Fornisce il meccanismo per la memorizzazione e l'accesso di dati e programmi
- Consiste di due parti
 - Collezione di file
 - Struttura di cartelle (directory)

Sommario

- Interfaccia
- Implementazione
- Prestazioni

L'INTERFACCIA DEL FILE SYSTEM

Sommario

- Concetto di file
- Metodi di accesso
- Struttura delle directory
- Protezione

Concetto di file

- Il S.O. astrae dalle caratteristiche fisiche dei supporti di memorizzazione fornendone una visione logica
- File
 - spazio di indirizzamento logico e contiguo
 - Insieme di informazioni correlate identificate da un nome
- Tipi di file
 - Dati
 - Numerici
 - Caratteri
 - Binari
 - Programmi

Attributi di un file

- Sono memorizzati su disco nella struttura della directory
 - Nome
 - Unica informazione in formato “leggibile”
 - Tipo
 - Posizione
 - Puntatore allo spazio fisico sul dispositivo
 - Dimensione
 - Protezione
 - Controllo su chi può leggere, scrivere, eseguire
 - Tempo, data e identificazione dell’utente

Operazioni su file

- Creazione
 - Cercare spazio su disco, nuovo elemento su directory per attributi
- Scrittura
 - System call che specifica nome file e dati da scrivere
 - Necessario puntatore alla locazione della prossima scrittura
- Lettura
 - System call che specifica nome file e dove mettere dati letti in memoria
 - Necessario puntatore alla locazione della prossima lettura (lo stesso della scrittura)

Operazioni su file

- Riposizionamento all'interno di file
 - Aggiornamento puntatore posizione corrente
- Cancellazione
 - Libera spazio associato al file e l'elemento corrispondente nella directory
- Troncamento
 - Mantiene inalterati gli attributi ma cancella contenuto del file

Operazioni su file

- Apertura
 - Ricerca del file nella struttura della directory su disco, copia del file in memoria e inserimento di un riferimento nella tabella dei file aperti
 - In sistemi multiutente ci sono 2 tabelle per gestire i file aperti
 - Una tabella per ogni processo contiene riferimenti per file aperti relativi al processo (es.: puntatore alla locazione di lettura/scrittura)
 - Una tabella per tutti i file aperti da tutti i processi contiene i dati indipendenti dal processo (es.: posizione sul disco, dimensione file, data accessi, n° di processi che hanno aperto il file)
- Chiusura
 - Copia del file in memoria su disco

Tipi di file – nomi ed estensioni

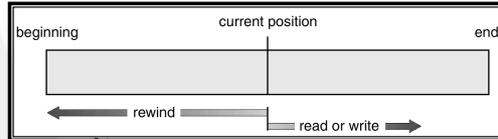
file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, 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	ps, dvi, gif	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

Struttura di un file

- I tipi di file possono essere adoperati per indicare la struttura interna del file
 1. Nessuna (sequenza di parole, bytes, ...)
 - Come Unix
 2. Struttura a “record” semplice
 - Record = riga
 - Lunghezza fissa o variabile
 3. Strutture complesse
 - Documenti formattati
 - Formati ricaricabili (load module)
- Possibilità di “emulare” (ma non sempre) 2 e 3 usando 1, tramite specifici caratteri di controllo

Metodi di accesso

- Sequenziale
 - Ad esempio usato da editor e compilatori
- Operazioni permesse
 - read next
 - write next
 - reset (rewind)
- Non è permesso il rewrite
 - Rischio di inconsistenza se scrivo qualcosa a metà del file perché potrei cancellare ciò che sta dopo

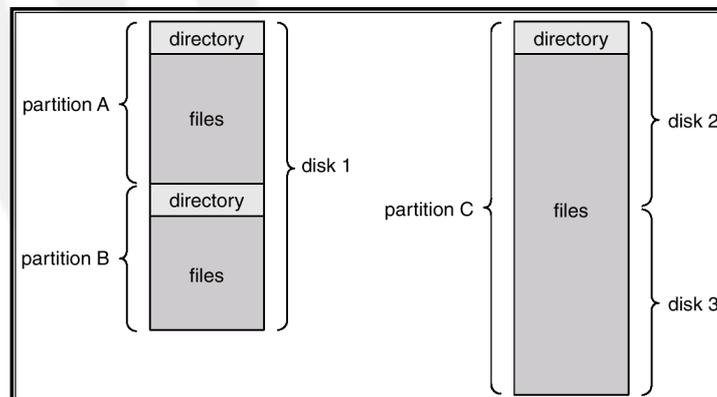


Metodi di accesso

- Diretto
 - Ad esempio usato da database
 - file=sequenza numerata di blocchi (record)
- Operazioni permesse
 - read n
 - write n
 - position to n
 - read next
 - write next
 - rewrite n

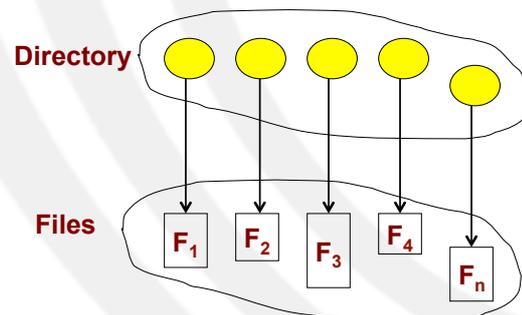
STRUTTURA DELLE DIRECTORY

Organizzazione del file system



Struttura delle directory

- Una collezione di nodi contenenti informazioni sui file
 - Entrambi residenti su disco
- Determinano la struttura del file system



Informazioni in una directory

- Per ogni file
 - Nome
 - Tipo
 - Indirizzo
 - Lunghezza attuale
 - Massima lunghezza
 - Data di ultimo accesso
 - Data di ultima modifica
 - Possessore
 - Info di protezione

Operazioni su directory

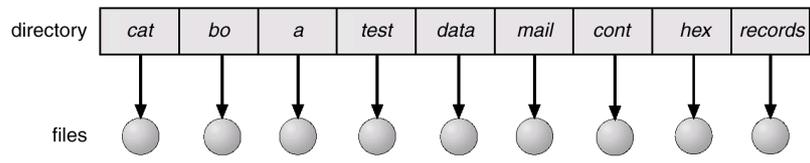
- Aggiungere un file
- Cancellare un file
- Visualizzare il contenuto della directory
- Rinominare un file
- Ricercare un file
 - Es.: cercare tutti i file il cui nome soddisfa una espressione
- Attraversare il file system

Directory - organizzazione logica

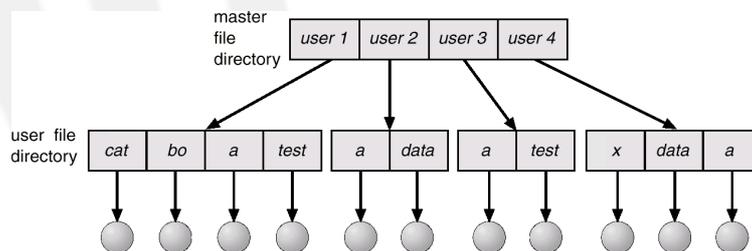
- Obiettivi
 - Efficienza
 - Rapido accesso ad un file
 - Nomenclatura (naming)
 - Conveniente agli utenti
 - Stessi nomi per file diversi e utenti diversi
 - Nomi diversi per lo stesso file
 - Raggruppamento
 - Classificazione logica dei file per criterio (tipo, protezione, etc.)

Directory a un livello

- Singola directory per tutti gli utenti
 - Problemi di nomenclatura
 - Difficile ricordare se un nome esiste già
 - Difficile inventare sempre nuovi nomi
 - Problemi di raggruppamento



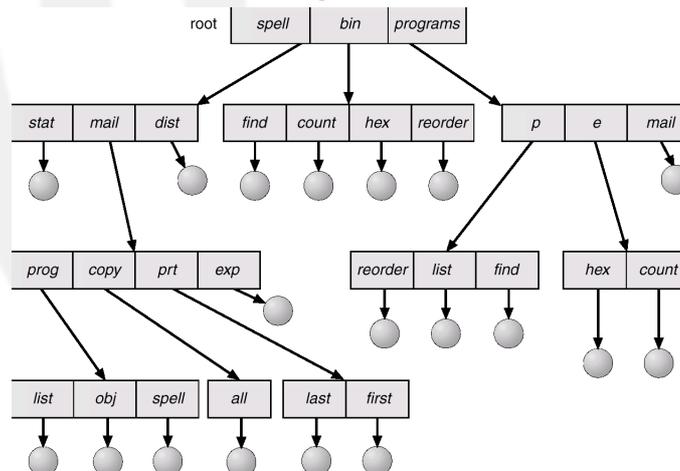
Directory a due livelli



Directory a due livelli

- Directory separata per ogni utente
 - Concetto di percorso (path)
 - Possibilità di usare lo stesso nome di file per utenti diversi
 - Ricerca efficiente
 - No raggruppamento
 - Dove mettiamo i programmi di sistema condivisi dagli utenti?
 - Es.: In Unix si usa la variabile d'ambiente PATH

Directory ad albero

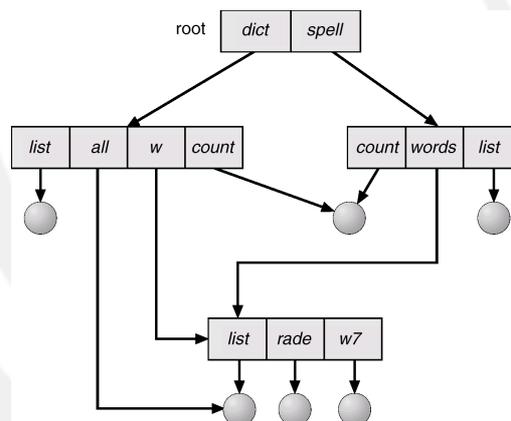


Directory ad albero

- Ricerca efficiente
- Possibilità di raggruppamento
- Concetto di directory corrente (working directory)
 - `cd /spell/mail/prog`
 - `pwd`
- Nomi di percorso assoluti o relativi

Directory a grafo aciclico

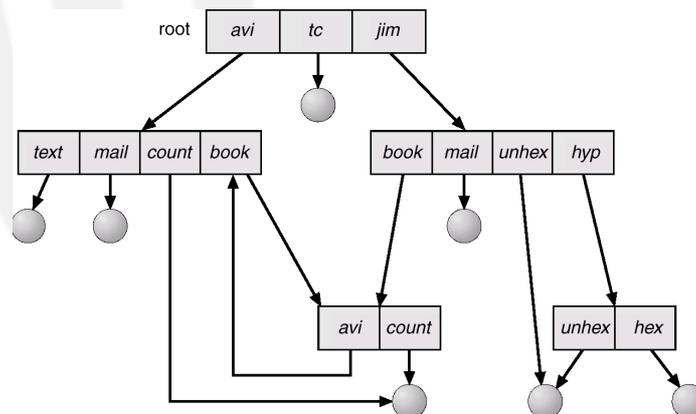
- Struttura ad albero non permette condivisione di file e directory
→ estensione a grafo aciclico



Directory a grafo aciclico

- Implementazione della condivisione
 - Windows → collegamenti
 - Unix → link
 - Link simbolico (come collegamenti di Windows)
 - Contiene il pathname del file/directory reale
 - Se cancello il file reale il link rimane pendente
 - Unix: In -s source destination
 - Hard link
 - Contatore che mantiene il # di riferimenti
 - Decrementato per ogni cancellazione di un riferimento
 - Cancellazione di file solo se il contatore vale 0
 - Unix: In source destination

Directory a grafo generico



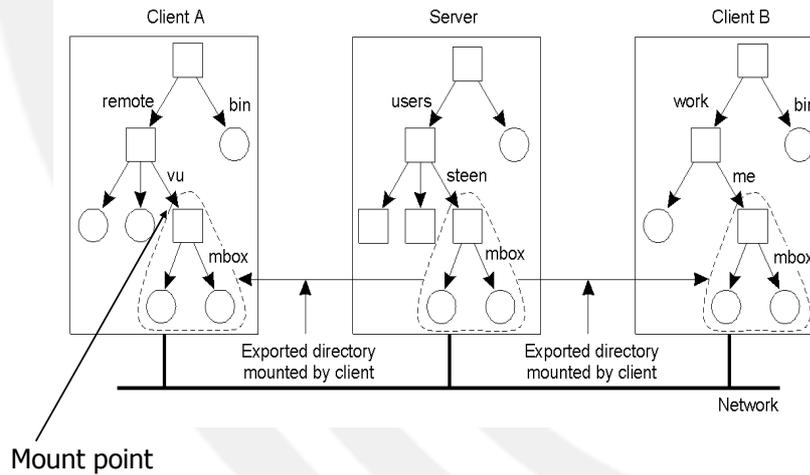
Directory a grafo generico

- Necessario garantire che non esistano cicli!
 - Per evitare loop infiniti nell'attraversamento del grafo
 - Soluzioni
 - Permettere di collegare solo file non directory
 - Usare garbage collection
 - Usare un algoritmo di controllo di esistenza di un ciclo ogni volta che si crea un nuovo collegamento
- } costosi

Mount di file system

- Realizzazione di file system modulari
- Possibilità di “attaccare” e “staccare” interi file system a file system preesistenti
- Mount/unmount = attaccare/staccare
- In generale un file system deve essere “montato” prima di potervi accedere
 - Punto in cui viene “montato” = mount point

Mount di file system – esempio



Condivisione di file

- Condivisione importante in sistemi multiutente
- Realizzabile tramite uno schema di protezione
- In sistemi distribuiti, i file possono essere condivisi attraverso una rete
- Esempio:
 - Network File System (NFS) è un tipico schema di condivisione di file via rete

Protezione

- Il possessore di un file deve poter controllare
 - Cosa è possibile fare su un file
 - Da parte di chi
- Tipi di operazioni controllabili
 - Lettura
 - Scrittura
 - Esecuzione
 - Append (aggiunta in coda)
 - Cancellazione
 - ...

Protezione

- Lista d'accesso per file/directory
 - Elenco di chi può fare che cosa per ogni file/directory
 - Può essere lungo
 - Utenti raggruppati in 3 classi (Unix)
 - Proprietario
 - Gruppo
 - Utenti appartenenti allo stesso gruppo del proprietario
 - Altri
 - Per ogni classe i permessi possono essere
 - r (read)
 - w (write)
 - x (execute)
 - Es.: `chmod 754 nome_file`
 - `rwx r-x r--`
 - tutto per proprietario, no write per gruppo, solo read per altri
- } Meno generale rispetto alla lista d'accesso

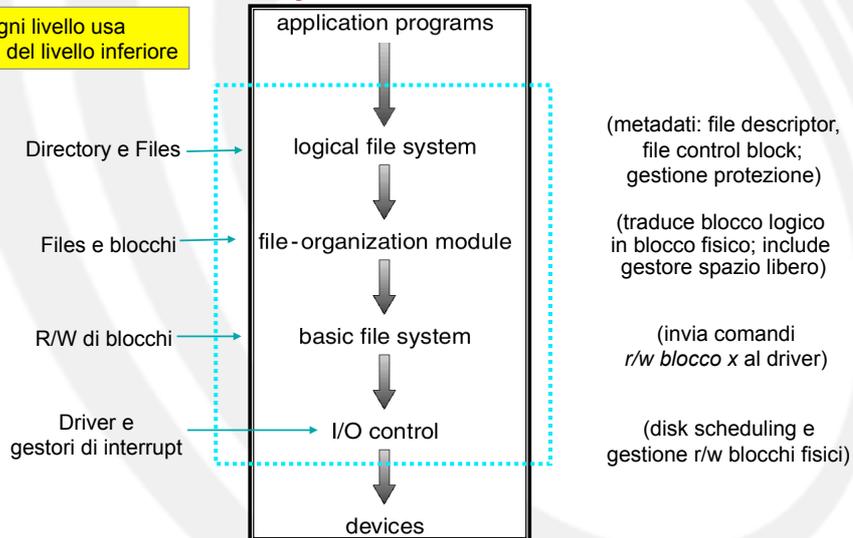
IMPLEMENTAZIONE DEL FILE SYSTEM

Sommario

- Implementazione del file system
- Metodi allocazione dello spazio su disco
- Gestione dello spazio libero
- Efficienza e prestazioni
- Recupero delle informazioni

File System a livelli

Ogni livello usa
funzioni del livello inferiore



Implementazione del file system

- Per gestire un file system si usano diverse strutture dati
 - Parte su disco
 - Parte in memoria
- Caratteristiche fortemente dipendenti dal sistema operativo e dal tipo di file system
 - Esistono caratteristiche comuni a tutte le tipologie

Implementazione del file system

- Strutture su disco
 - Blocco di boot
 - Informazioni necessarie per l'avviamento del S.O.
 - Blocco di controllo delle partizioni
 - Dettagli riguardanti la partizione
 - Numero e dimensione dei blocchi, lista blocchi liberi, lista descrittori liberi, ...
 - Strutture di directory
 - Descrivono l'organizzazione dei file
 - Descrittori di file (inode)
 - Vari dettagli sui file e puntatori ai blocchi dati

Tipicamente in questo ordine sul disco

Implementazione del file system

- Strutture in memoria
 - Tabella delle partizioni
 - Informazioni sulle partizioni montate
 - Strutture di directory
 - Copia in memoria delle directory a cui si è fatto accesso di recente
 - Tabella globale dei file aperti
 - Copie dei descrittori di file
 - Tabella dei file aperti per ogni processo
 - Puntatore alla tabella precedente ed informazioni di accesso

Allocazione dello spazio su disco

- Come i blocchi su disco sono allocati ai file (o alle directory)
- Alternative
 - Allocazione contigua
 - Allocazione a lista concatenata (linked)
 - Allocazione indicizzata
- Obiettivi
 - Minimizzare tempi di accesso
 - Massimizzare utilizzo dello spazio

Allocazione contigua

- Ogni file occupa un insieme di blocchi contigui su disco
 - Entry della directory semplice
 - Contiene indirizzo blocco di partenza e lunghezza (numero di blocchi)

Allocazione contigua

- Vantaggi
 - **Accesso semplice**
 - Accesso al blocco $b+1$ non richiede spostamento testina rispetto al blocco b
 - A meno che b non sia l'ultimo blocco di un cilindro (raro)
 - Supporta accesso sequenziale e casuale
 - Per leggere blocco i di un file che inizia al blocco b basta fare $b+i$

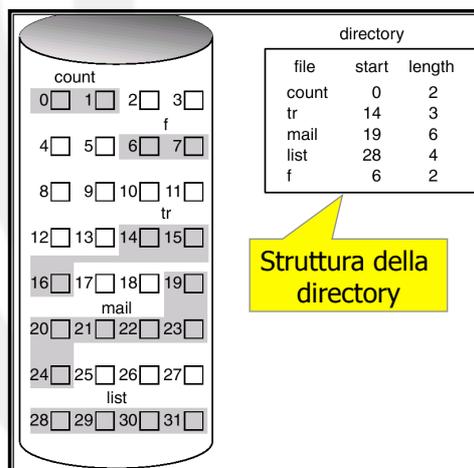
Allocazione contigua

- Svantaggi
 - Allocazione contigua dei blocchi presenta problemi simili a quelli dell'allocazione dinamica della memoria
 - Algoritmi best-fit, first-fit, worst-fit
 - Spreco di spazio (frammentazione esterna)
 - Richiede compattazione periodica dello spazio

Allocazione contigua

- Quanto spazio allocare quando viene creato un file dato che i file non possono crescere dinamicamente di dimensione?
 - Soluzioni:
 - Se il file deve crescere e non c'è spazio → errore e terminazione del programma → riesecuzione
 - Si tende a sovrastimare lo spazio necessario (spreco)
 - Trovare un buco + grande e ricopiare tutto il file (compresa la parte nuova) in quest'ultimo
 - Trasparente x l'utente, ma rallenta il sistema

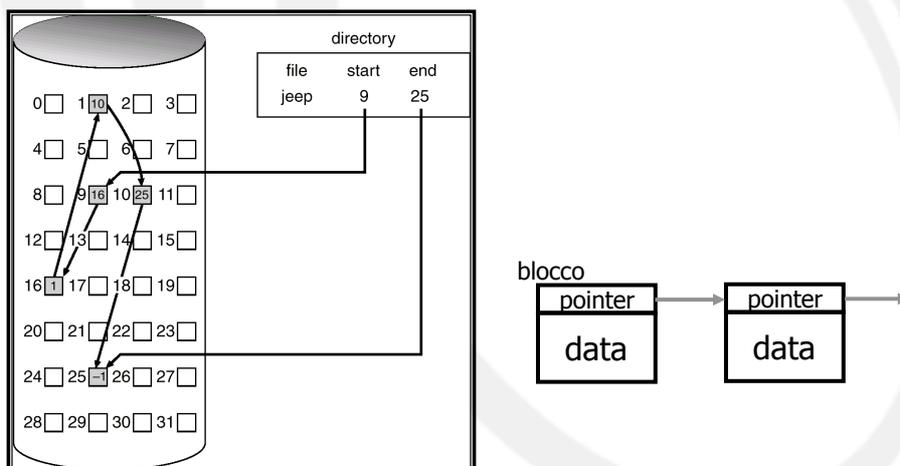
Allocazione contigua



Allocazione a lista

- Ogni file è una lista di blocchi
- I blocchi possono essere sparsi ovunque nel disco
 - Directory contiene puntatori al primo e all'ultimo blocco
 - Ogni blocco contiene puntatore al blocco successivo
- Indirizzamento
 - X = indirizzo logico
 - N = dimensione del blocco
 - $X / (N-1)$ = numero del blocco nella lista
 - $X \% (N-1)$ = offset all'interno del blocco

Allocazione a lista – esempio



Allocazione a lista

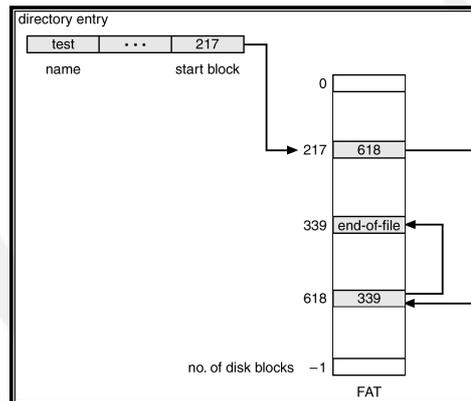
- Vantaggi
 - Creazione nuovo file semplice
 - Basta cercare un blocco libero e creare una nuova entry nella directory che punta al blocco
 - Estensione del file semplice
 - Basta cercare un blocco libero e concatenarlo alla fine del file
 - Nessuno spreco (eccetto lo spazio per il puntatore)
 - Posso usare qualunque blocco → No frammentazione esterna

Allocazione a lista

- Svantaggi
 - No accesso casuale
 - Bisogna scorrere tutti i blocchi a partire dal primo
 - Richiede tanti riposizionamenti sparsi → scarsa efficienza
 - Scarsa affidabilità
 - Se si perde un puntatore, oppure se un errore (SW/ HW) causa il prelevamento del puntatore sbagliato?
 - Soluzioni (con overhead)
 - Liste doppiamente concatenate
 - Memorizzare nome file e n° di blocco in ogni blocco del file

Esempio – FAT (File-Allocation Table)

- Tipica dei sistemi MS-DOS e OS/2
- Una FAT per ogni partizione contiene un elemento per ogni blocco del disco
- Usata come lista concatenata
- Migliora accesso casuale, ma l'efficienza rimane scarsa

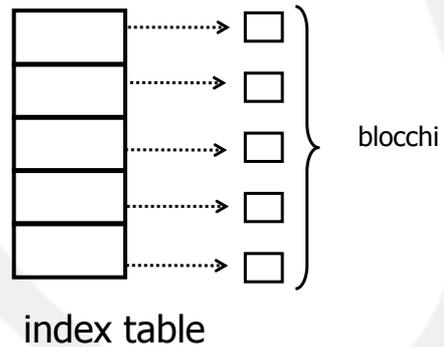


Allocazione contigua – variante

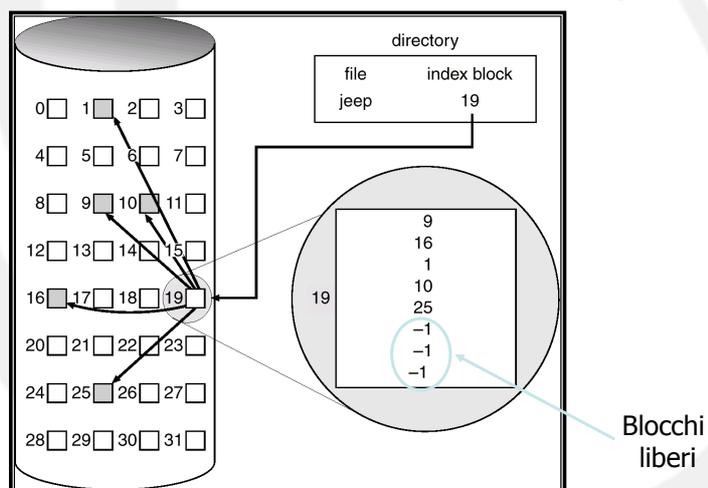
- Alcuni file system moderni usano uno schema modificato di allocazione contigua
- Basato sul concetto di extent
 - Extent = serie di blocchi contigui su disco (extent-based file system)
- File system alloca extent anziché singoli blocchi
 - File = serie di extent
 - I vari extent non sono in generale contigui
 - Adatto per allocazione a lista

Allocazione indicizzata

- Ogni file ha un blocco indice (index block) contenente la tabella degli indirizzi (index table) dei blocchi fisici
- La directory contiene l'indirizzo del blocco indice



Allocazione indicizzata – esempio



Allocazione indicizzata

- Accesso casuale efficiente
- Accesso dinamico senza frammentazione esterna
 - ma con overhead del blocco indice per la index table
 - maggiore di quello richiesto per allocazione concatenata
- Indirizzamento:
 - X = indirizzo logico
 - N = dimensione del blocco
 - X / N = offset nella index table
 - $X \% N$ = offset all'interno del blocco dati

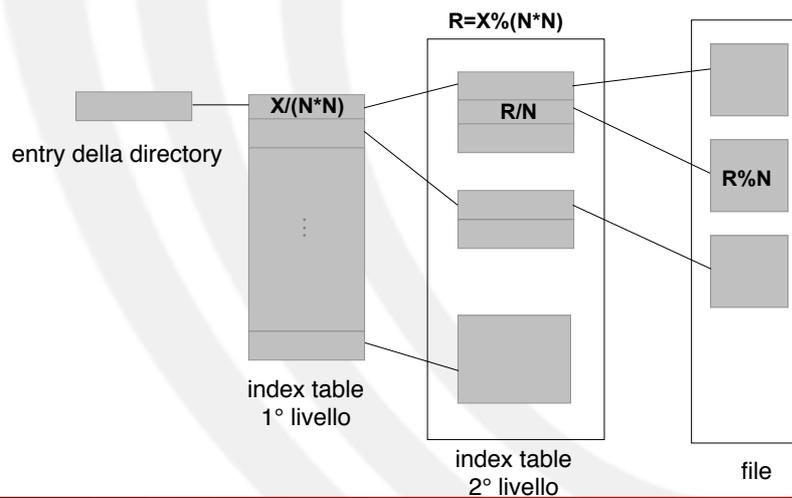
Allocazione indicizzata

- La dimensione del blocco limita la dimensione del file!
 - Es.: dimensione blocco = 512 parole ➔
massima dimensione del file = 5122 parole =
256K parole
- Per file di dimensione senza limiti si usa uno schema a più livelli:
 - Indici multilivello
 - Schema concatenato
 - Schema combinato (stile Unix BSD)

Indici multilivello

- Una tabella più esterna contiene puntatori alle index table
 - X = indirizzo logico
 - N = dimensione del blocco (in parole)
 - $X / (N*N) =$ blocco della index table di 1° livello
 - $X \% (N*N) = R$
 - R usato come segue
 - $R / N =$ offset nel blocco della index table di 2° livello
 - $R \% N =$ offset nel blocco dati
- Es.: blocchi da 4KB consentono 1K indici da 4 byte → due livelli di indici consentono file da 4GB

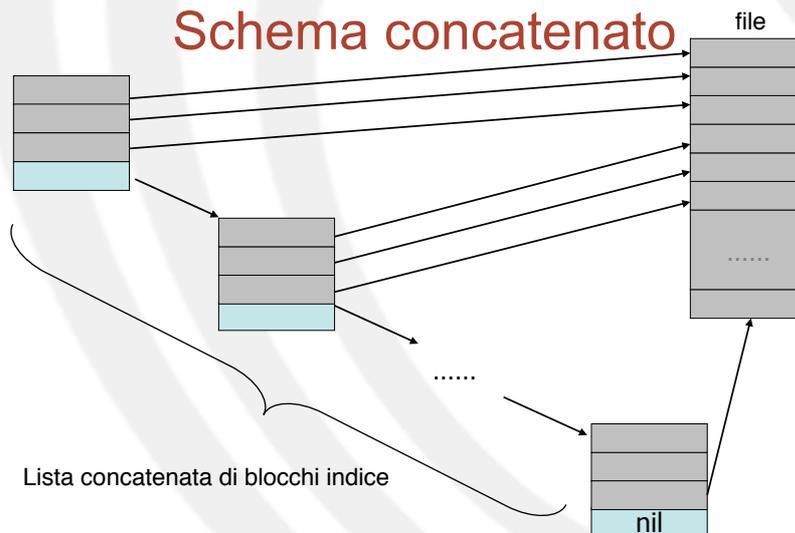
Indici multilivello



Schema concatenato

- Lista concatenata di blocchi indice
 - L'ultimo degli indici di un blocco indice punta a un'altro blocco indice
 - X = indirizzo logico
 - N = dimensione del blocco (in parole)
 - $X / (N(N-1)) =$ numero del blocco indice all'interno della lista dei blocchi indice
 - $X \% (N(N-1)) = R$
 - R usato come segue
 - $R / N =$ offset nel blocco indice
 - $R \% N =$ offset nel blocco dati

Schema concatenato

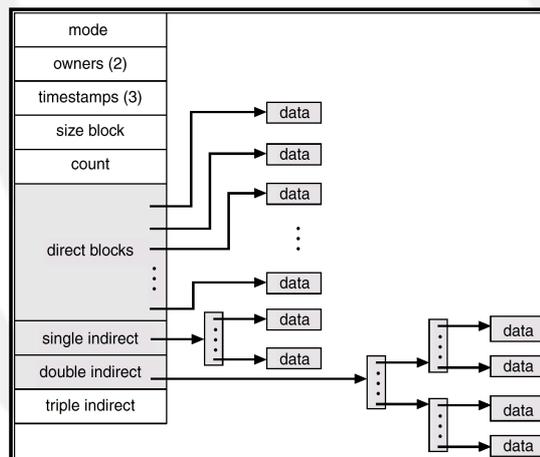


Schema concatenato – esempio

- $N = 1\text{KB} = 256$ parole
 - 4 byte per parola
- Max file = 256 blocchi da 1KB = 256KB
 - $X = 12200$
 - $X/N = 12200/256 = 47$ (blocco 47)
 - $X\%N = 168$ (parola 168 all'interno del blocco 47)
 - $X = 644000$ Non basta un primo livello
 - $X / (N(N-1)) = 9$ (blocco della lista concatenata)
 - $X \% (N(N-1)) = R = 56480$
 - $R / N = 220$ (offset nel blocco index)
 - $R \% N = 160$ (offset nel blocco dati)

Allocazione indicizzata – esempio

- Schema combinato (Unix)
 - blocco di 4KB



Implementazione delle directory

- Lo stesso meccanismo usato per memorizzare file si applica alle directory
- Le directory non contengono dati
 - Tipicamente la lista dei file (e delle directory) che essa contiene
- Problematiche:
 - Come questo contenuto viene memorizzato
 - Come si accede al contenuto delle directory

Implementazione delle directory

- Lista lineare di nomi di file con puntatori ai blocchi dati
 - Implementazione facile
 - Poco efficiente
 - Lettura, scrittura e rimozione del file richiedono ricerca per trovare il file
 - scansione lineare della lista (complessità $O(n)$)
 - ricerca binaria su lista ordinata (complessità $O(\log n)$) ma bisogna ordinare la lista...
- Tabella hash
 - Tempo di ricerca migliore
 - Possibilità di collisioni
 - Situazioni in cui due nomi di file collidono sulla stessa posizione

GESTIONE DELLO SPAZIO LIBERO

Gestione dello spazio libero

- Per tenere traccia dello spazio libero su disco si mantiene una lista dei blocchi liberi
 - Per creare un file si cercano blocchi liberi nella lista
 - Per rimuovere un file si aggiungono i suoi blocchi alla lista
- Alternative
 - Vettore di bit
 - Lista concatenata
 - Raggruppamento
 - Conteggio

Vettore di bit

- Vettore di bit, uno per blocco
 - Bit[i] = 1 ⇒ blocco i libero
 - Bit[i] = 0 ⇒ blocco i occupato
- Esempio: n blocchi

0	1	2				...		n-1
---	---	---	--	--	--	-----	--	-----
- Calcolo del numero del primo blocco libero:
 - Cerca la prima parola non 0
 - (# di bit per parola) * (# di parole a 0) + (offset del primo bit a 1)
 - Es: 000000000000000000000000000010000100000111110000000000000

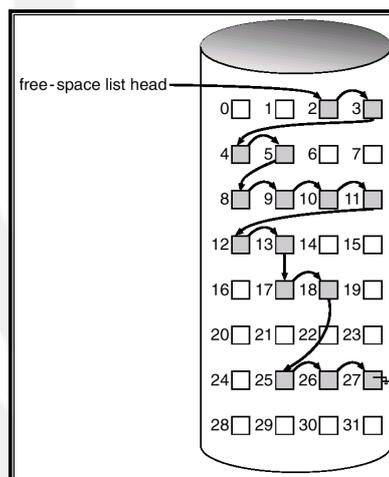
Vettore di bit

- La mappa di bit richiede extra spazio
 - Esempio:
 - |blocco| = 2^{12} byte (4KB)
 - Dimensione del disco = 2^{38} byte (256 GB)
 - $n = 2^{38}/2^{12} = 2^{26}$ bit (8MB) solo per il vettore di bit
- Efficiente solo se il vettore è mantenibile tutto in memoria
- Facile ottenere file contigui

Alternative

- Lista concatenata di blocchi liberi (free list)
 - Spreco minimo (solo per la testa della lista)
 - Spazio contiguo non ottenibile
- Raggruppamento
 - Modifica della lista linkata (simile ad allocazione indicizzata)
 - Primo blocco libero = indirizzi di n-1 blocchi liberi
 - Ultima entry del blocco = indirizzo del primo blocco del gruppo successivo di n blocchi liberi
 - Fornisce rapidamente un gran numero di blocchi liberi
- Conteggio
 - Mantiene il conteggio di quanti blocchi liberi seguono il primo in una zona di blocchi liberi contigui
 - Generalmente la lista risulta + corta (se il contatore è > 1 per ogni gruppo di blocchi liberi)

Gestione spazio libero con lista



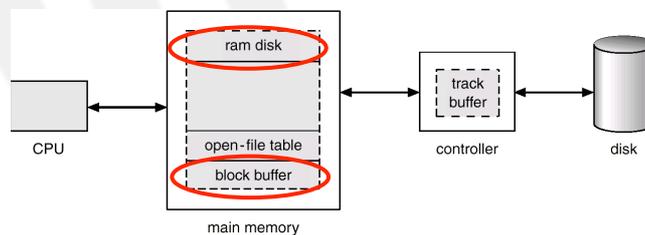
EFFICIENZA E PRESTAZIONI

Efficienza

- Disco è collo di bottiglia
- Efficienza dipende da:
 - **Algoritmo di allocazione dello spazio su disco**
 - Es.: in Unix si cerca di tenere i blocchi di un file vicini al suo i-node
 - Richiesta preallocazione i-node distribuiti sulla partizione
 - **Tipo di dati contenuti nella directory**
 - Es.: data di ultimo accesso di un file → lettura di un file richiede lettura e scrittura anche del blocco della directory

Prestazioni

- Il controller del disco possiede una piccola cache che è in grado di contenere un'intera traccia ma...
- ... non basta per garantire prestazioni elevate, quindi
 - Dischi virtuali (RAM disk)
 - Cache del disco (detta anche buffer cache)



Dischi virtuali

- Parte della memoria gestita come se fosse un disco
 - Il driver di un RAM disk accetta tutte le operazioni standard dei dischi eseguendole però in memoria
 - Veloce
 - Supporto solo per file temporanei
 - se spengo perdo tutto
 - Gestito dall'utente che scrive sul RAM disk invece che sul disco vero e proprio (non è una cache!)

Cache del disco

- Porzione di memoria che memorizza blocchi usati di frequente
- Simile alla cache tra memoria e CPU
- Gestita dal S.O.
- Sfrutta principio della località
 - Spaziale
 - uso di dati “vicini” a quelli attualmente usati
 - Temporale
 - uso successivo degli stessi dati
- Trasferimento dati nella memoria del processo utente non richiede spostamento di byte

Cache del disco

- Problematiche
 - Dimensione della cache
 - $O(\text{MB})$
 - Politica di rimpiazzamento
 - Cosa fare in caso di necessità di eliminare un settore?
 - LRU, LFU, RANDOM,
 - LRU poco efficiente per accesso sequenziale, meglio
 - rilascio indietro
 - lettura anticipata
 - Politica di scrittura
 - Se l'operazione è una scrittura, come aggiornare il contenuto su disco?
 - Write-back: scrivo solo quando devo rimuovere il blocco dalla cache
 - Problemi di affidabilità in caso di crash
 - Write-through: scrivo sempre
 - Meno efficiente, la cache è solo in lettura

Recupero

- Possibili problemi di consistenza tra disco e cache
- Controllo di consistenza
 - Confrontare i dati nella directory con i dati su disco, e sistemare le inconsistenze
 - Specie in caso di crash: per es. ScanDisk
- Utilizzo di programmi di sistema per fare il back up del disco su memoria di massa (nastri)
 - Recupero di file persi tramite restore dei dati dai backup

File system log structured

- Registrano ogni cambiamento del file system come una transazione
 - Tutte le transazioni sono scritte su un log
 - Una transazione è considerata avvenuta quando viene scritta sul log
 - Anche se il file system può non essere aggiornato
 - Le transazioni sul log sono scritte in modo asincrono nel file system
 - Quando il file system è modificato, la transazione viene cancellata dal log
 - Se il sistema va in crash, le transazioni non avvenute sono quelle presenti sul log
- Vantaggio
 - Ottimizzazione del numero di seek