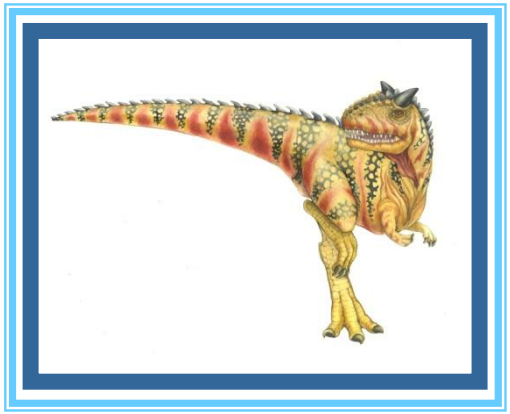
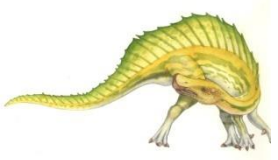
Realizzazione del file system



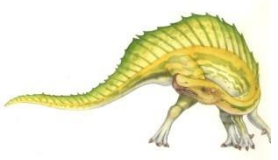
Obiettivi

- Descrivere i dettagli implementativi dei file system e delle strutture di directory
- Descrivere le tecniche di allocazione di blocchi/pagine e i trade–off degli algoritmi per la gestione dello spazio libero
- Esplorare i problemi relativi all'efficienza e alle prestazioni del file system
- Analizzare il ripristino del file system



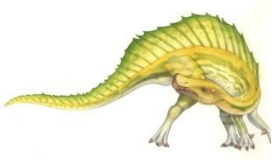
Sommario

- Struttura del file system
- Realizzazione del file system
- Realizzazione delle directory
- Metodi di allocazione dei file
- Gestione dello spazio libero
- Efficienza e prestazioni
- Ripristino
- Appendice: Il Network File System



Introduzione

- Un sistema operativo ad uso generale fornisce diversi file system e, frequentemente, permette agli utenti di aggiungerne ulteriori
- I file system variano in base a diversi aspetti, quali obiettivi di progettazione, funzionalità, prestazioni e affidabilità
  - Per la memorizzazione ed il recupero rapido di file non persistenti si usano file system temporanei realizzati in RAM
  - Il file system standard di archiviazione secondaria in Linux sacrifica (parzialmente) le prestazioni per gantire affidabilità e semplicità d'uso

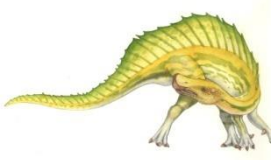


Struttura del file system – 1

- Struttura del file:
  - Unità di memorizzazione logica
  - Collezione di informazioni correlate
  - File control block: struttura dati del kernel che "descrive" il file
- Il file system risiede nella memoria secondaria
  - Fornisce una semplice interfaccia per la memorizzazione non volatile, realizzando il mapping fra indirizzi logici e fisici
  - Fornisce la possibilità di accedere alla memoria in maniera efficiente, per memorizzare e recuperare rapidamente le informazioni

Struttura del file system – 2

- I dischi magnetici rimangono un mezzo conveniente per la memorizzazione di file perché:
  - Si possono riscrivere localmente; si può leggere un blocco dal disco, modificarlo e quindi scriverlo nella stessa posizione
  - È possibile accedere direttamente a qualsiasi blocco di informazioni del disco, quindi a qualsiasi file, sia in modo sequenziale che diretto (spostando le testine di lettura/ scrittura ed attendendo la rotazione del disco)
- I dispositivi NVM sono sempre più diffusi per l'archiviazione di file e su di essi vengono creati file system
  - Non possono essere riscritti direttamente e presentano performance "variabili" per l'operazione di scrittura

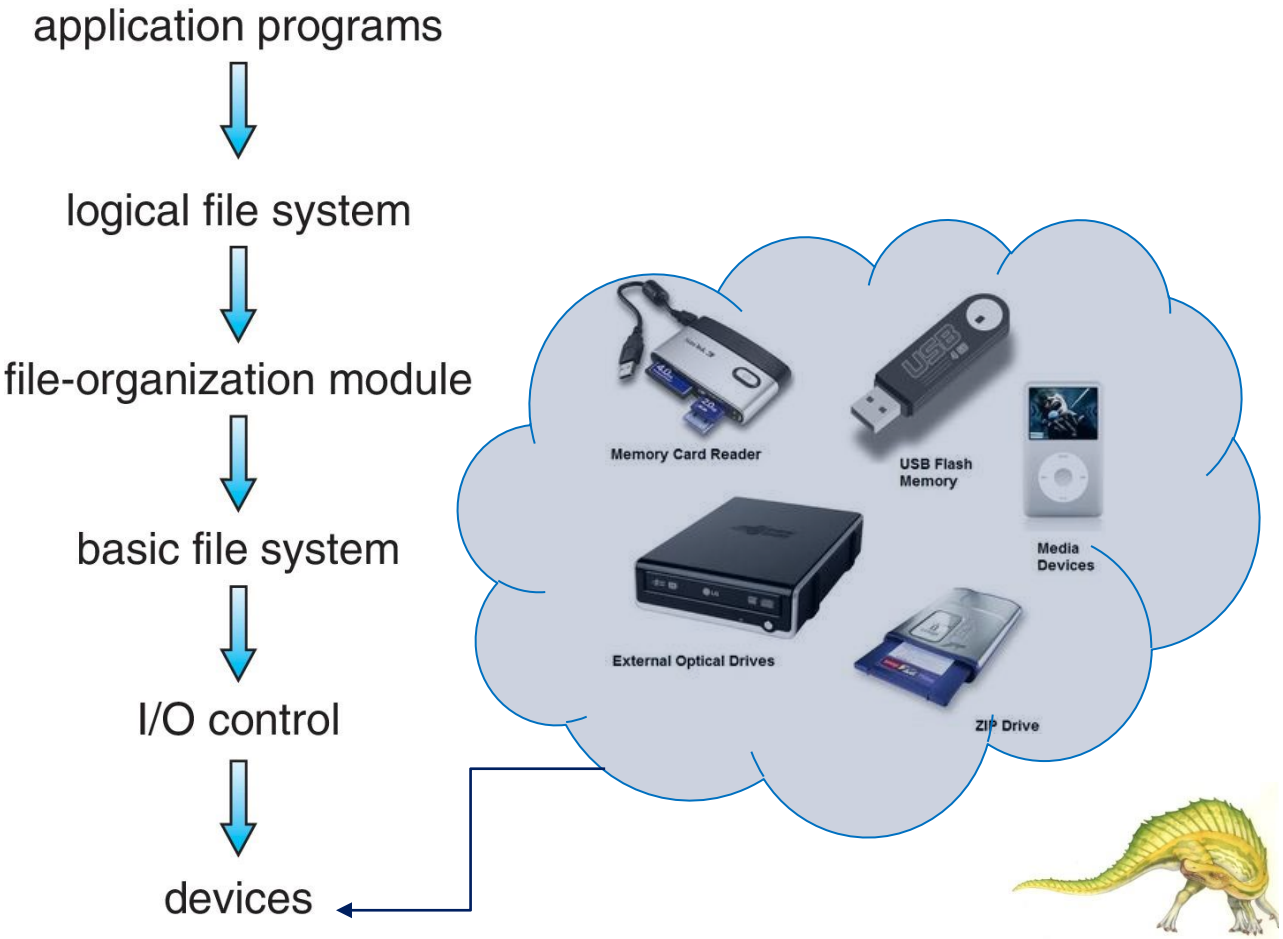


Struttura del file system – 3

- Le operazioni di I/O su disco avvengono con "granularità" di blocco
  - Ciascun blocco è composto da uno o più settori (512–4096 byte per settore)
  - Dimensione media attuale dei blocchi 4KB
  - I dispositivi fisici vengono controllati da device driver
- Anche i dispositivi NVM hanno normalmente pagine di 4KB e utilizzano metodi di trasferimento simili a quelli delle unità a disco
- File system per l'utente
  - Definizione (dell'aspetto) di file e directory e loro operazioni
- File system per il SO
  - Scelta di algoritmi e strutture dati che mettano in corrispondenza il file system logico con i dispositivi fisici di memorizzazione

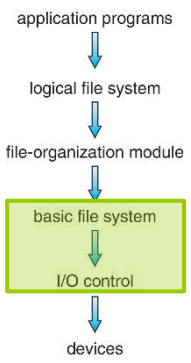
Struttura del file system – 4

- Il file system è **stratificato**, cioè organizzato in livelli
  - Ogni livello si serve delle funzioni dei livelli inferiori per crearne di nuove, impiegate dai livelli superiori



Strati del file system – 1

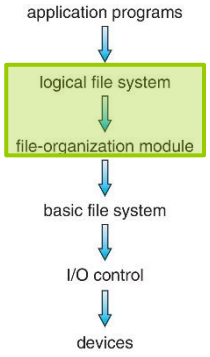
- Controllo dell'I/O: Driver e gestori interrupt
  - Traducono comandi di alto livello ("leggi il blocco fisico 123") in sequenze di bit che guidano l'hardware di I/O a compiere una specifica operazione in una data locazione
  - In altre parole… scrivono specifiche configurazioni di bit in specifiche locazioni della memoria del controllore (microcodice del controllore) per indicare al dispositivo che operazioni compiere e dove



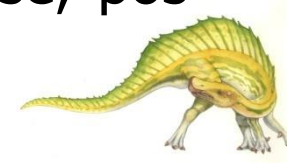
- File system di base
  - Invia comandi generici al driver di dispositivo utilizzando indirizzi LBA per leggere/scrivere blocchi fisici su disco
  - Si occupa della pianificazione delle richieste di I/O
  - Gestisce il buffer del dispositivo e la cache che conserva i blocchi del file e i metadati (in RAM)

Strati del file system – 2

- Modulo di organizzazione dei file
  - Traduce gli indirizzi logici di blocco (nel range $0–N$) in LBA
  - Contiene il modulo per la gestione dello spazio libero
- File system logico
  - Gestisce i metadati, cioè tutte le strutture del file system eccetto i dati veri e propri memorizzati nei file
    - Mantiene le strutture di file tramite i file control block, FCB (inode in UNIX), che contengono le informazioni sui file, quali proprietario, permessi, posizione del contenuto
    - Gestisce le directory
    - Gestisce protezione e sicurezza

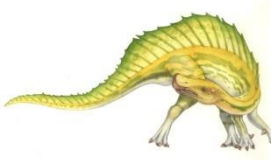


- La struttura a strati è utile per ridurre la complessità e la ridondanza, ma aggiunge overhead e può diminuire le performance
  - Il controllo dell'I/O e, talvolta, il file system di base, possono essere comuni a più file system



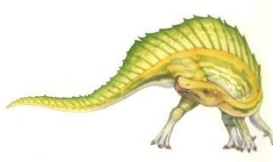
Tipi di file system – 1

- Esistono diversi tipi di file system e non è raro che i sistemi operativi ne prevedano più di uno
  - CD–ROM: ISO 9660
  - UNIX: UFS (UNIX File System) che si fonda sul Berkeley Fast File System (FFS)
  - Windows: FAT, FAT32, exFAT, NTFS
  - Linux: ext3, ext4 (extended file system), xfs, FAT, XFS e Btrfs, ma ne supporta più di centotrenta tipi
  - FUSE (File system in USErspace): è un progetto open–source, rilasciato sotto la licenza GPL e LGPL, volto alla realizzazione di un modulo integrabile al kernel Linux che permetta agli utenti non privilegiati di creare un proprio file system senza scrivere codice a livello kernel



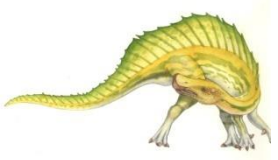
Tipi di file system – 2

- ZFS, implementato per la prima volta nella release 10 di Solaris (2006) sotto licenza CDDL (Common Development and Distribution License), è open source
  - Nasce come file system in grado di superare per dimensioni qualsiasi limite pratico di storage: caratteristica garantita dai 128 bit su cui è strutturato
  - Il nome originario era infatti "Zetta File System" a indicare la capacità di memorizzazione dell'ordine dei triliardi di bit, notevolmente superiore a quella dei classici sistemi a 64 bit
  - La minima massa di un dispositivo attuale in grado di archiviare 2^{128} byte è di circa 136 miliardi di kg!



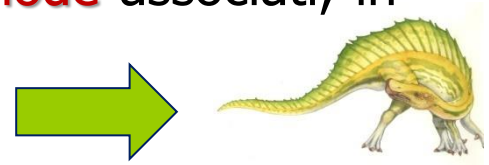
Tipi di file system – 3

- Google ha progettato un file system (proprietario) per soddisfare le esigenze di memorizzazione e recupero dati specifiche dell'azienda
  - GoogleFS, detto anche BigFiles, conserva i dati raccolti da Google utilizzando tecnologie non convenzionali, data la grandezza dei file
  - Dati immagazzinati in maniera permanente in file di circa 100GB ciascuno, solo raramente eliminati, sovrascritti o compressi
  - File con accesso per sola lettura
- Oracle ASM (Automatic Storage Management) permette di gestire file, directory, volumi per mezzo di direttive SQL in ambiente DBMS



Realizzazione del file system – 1

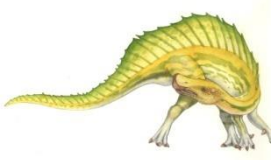
- Le funzioni del file system vengono realizzate tramite chiamate di sistema invocate attraverso la API, che utilizzano dati, gestiti dal kernel, residenti sia su disco che in memoria
- Strutture dati del file system residenti su disco
  - Blocco di controllo di avviamento: contiene le informazioni per l'avviamento di un SO da quel volume — boot block nell'UFS, nei sistemi Windows è il partition boot sector
    - Necessario se il volume contiene un SO (vuoto altrimenti)
    - Normalmente è il primo blocco del volume
  - Blocchi di controllo dei volumi: contengono dettagli riguardanti la partizione, quali numero totale dei blocchi e loro dimensione, contatore dei blocchi liberi e relativi puntatori, contatore degli FCB liberi e relativi puntatori — superblocco in UFS, master file table, MFT, in NTFS
  - Strutture delle directory: usate per organizzare i file (in UFS comprendono i nomi dei file ed i numeri di inode associati, in NTFS memorizzate nella master file table)



Realizzazione del file system – 2

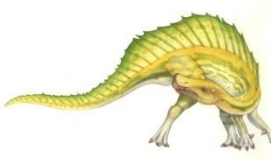
- Blocchi di controllo dei file, FCB (inode nell'UFS), contengono dettagli sul file
  - Identificativo del file, permessi, dimensione, date di creazione/ultimo accesso/ultima modifica, puntatori ai blocchi di dati
  - NTFS memorizza i metadati nella master file table utilizzando una struttura stile DB relazionale

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

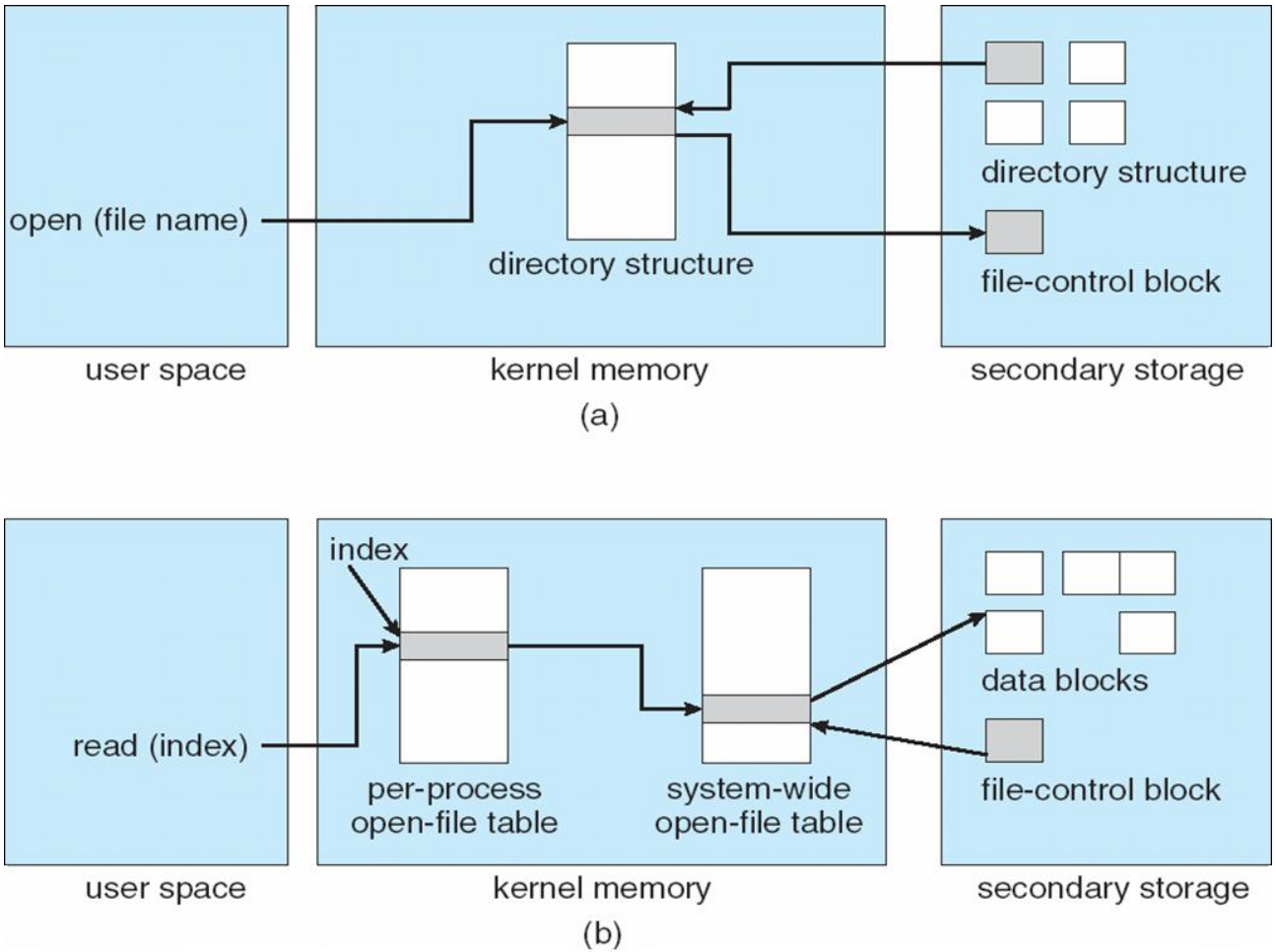


Realizzazione del file system – 3

- Strutture dati del file system residenti in memoria
  - Tabella di montaggio: contiene le informazioni relative a ciascun volume montato
  - Struttura delle directory: contiene informazioni relative a tutte le directory cui i processi hanno avuto accesso di recente (cache)
  - Tabella dei file aperti: contiene una copia dell'FCB per ciascun file aperto nel sistema
  - Tabella dei file aperti per ciascun processo: contiene un puntatore all'elemento corrispondente nella tabella generale, più informazioni di accesso specifiche del processo



Strutture del file system mantenute in memoria



(a) Apertura di un file, (b) lettura da file

Tutto è un file

- Alcuni sistemi operativi usano il file system come interfaccia per gestire altri «aspetti» del sistema
  - Nell'UFS la tabella di sistema dei file aperti contiene gli inode…
  - ma contiene anche informazioni simili per le connessioni di rete e i dispositivi

Realizzazione delle directory – 1

- La selezione degli algoritmi di allocazione e di gestione delle directory influenza in modo significativo l'efficienza, le prestazioni e l'affidabilità del file system
- Lista lineare di nomi di file con puntatori ai blocchi di dati
  - Semplice da implementare
  - Esecuzione onerosa dal punto di vista del tempo di ricerca (complessità lineare nel numero di elementi contenuti nella directory)
  - Liste concatenate per implementare facilmente l'operazione di cancellazione
  - Lista ordinata (o B–albero): migliora il tempo di ricerca, ma l'ordinamento deve essere mantenuto a fronte di ogni inserimento/cancellazione
    - Utile per produrre l'elenco ordinato (eventualmente parziale) dei file contenuti nella directory

Realizzazione delle directory

- Tabella hash: lista lineare con struttura hash
  - Migliora il tempo di ricerca nella directory
  - Inserimento e cancellazione costano $\mathcal{O}(1)$, se non si verificano collisioni
  - Collisione: situazione in cui due nomi di file generano lo stesso indirizzo hash nella tabella
  - Dimensione fissa e necessità di rehash o liste di trabocco

Metodi di allocazione

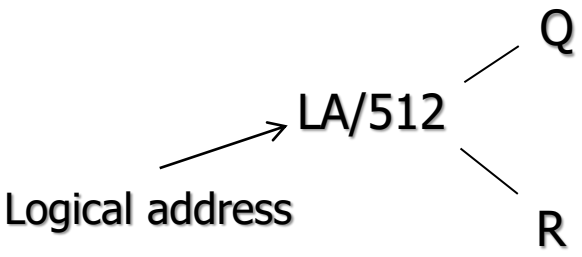
- La natura ad accesso diretto di dischi/NVM garantisce flessibilità nell'implementazione dei file
- Problema: allocare lo spazio ai file in modo da avere spreco minimo di memoria e rapidità di accesso
- Il metodo di allocazione dello spazio su memoria di massa descrive come i blocchi fisici vengono assegnati ai file; su disco:
  - Allocazione contigua
  - Allocazione concatenata
  - Allocazione indicizzata
- Anche se alcuni SO dispongono di tutti i metodi, più spesso un sistema usa un unico metodo per tutti i file di un dato file system

Allocazione contigua – 1

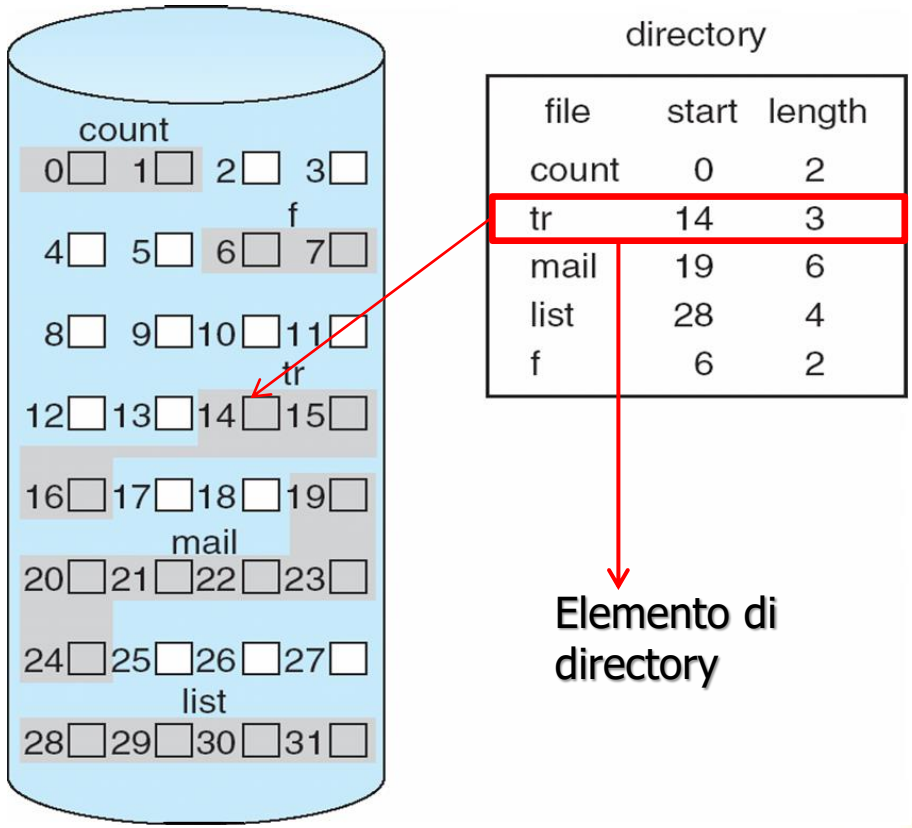
- Ciascun file occupa un insieme di blocchi contigui sul disco
- Per reperire il file occorrono solo la locazione iniziale (# blocco iniziale) e la lunghezza (numero di blocchi)
  - Accesso sequenziale ⇨ Nessuno o, al più, un solo spostamento della testina su un cilindro attiguo
  - Accesso casuale ⇨ Performance ottimali
- Problemi
  - Allocazione dinamica dello spazio disco
  - Frammentazione esterna
  - Necessità di conoscere a priori la dimensione dei file (i file non possono crescere, soprattutto per *best–fit*)
  - Compattazione dello spazio disco

Allocazione contigua – 2

- Mapping da blocchi logici a blocchi fisici (hp: dim. blocco pari a 512 byte/word)

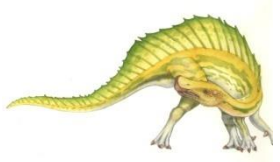


Il blocco da accedere è il Q–esimo a partire dall'indirizzo del blocco iniziale; lo spostamento all'interno del blocco è pari ad R



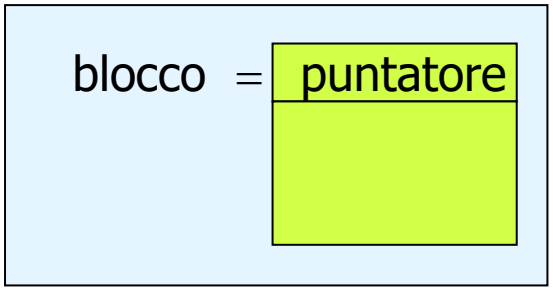
Allocazione contigua – 3

- Nei sistemi operativi di nuova generazione (es., nel file system Veritas), il disco viene allocato con granularità maggiore della dimensione del blocco fisico
- Ciascun file consiste di uno o più extent (di dim. variabile ed eventualmente definita dall'utente)
- Un extent è una "porzione di spazio contiguo"
- Inizialmente, per ciascun file viene allocato un extent
  - Se questo non è sufficientemente grande, si aggiunge un'ulteriore estensione
  - Problema: frammentazione interna per extent grandi
  - L'elemento di directory contiene l'indirizzo iniziale dell'extent e la sua dimensione ed un puntatore al primo blocco dell'estensione successiva



Allocazione concatenata – 1

- Ciascun file è una lista concatenata di blocchi: i blocchi possono essere sparsi ovunque nel disco
  - Ciascun blocco contiene un puntatore al blocco successivo
  - Il file termina quando si incontra un blocco con puntatore vuoto



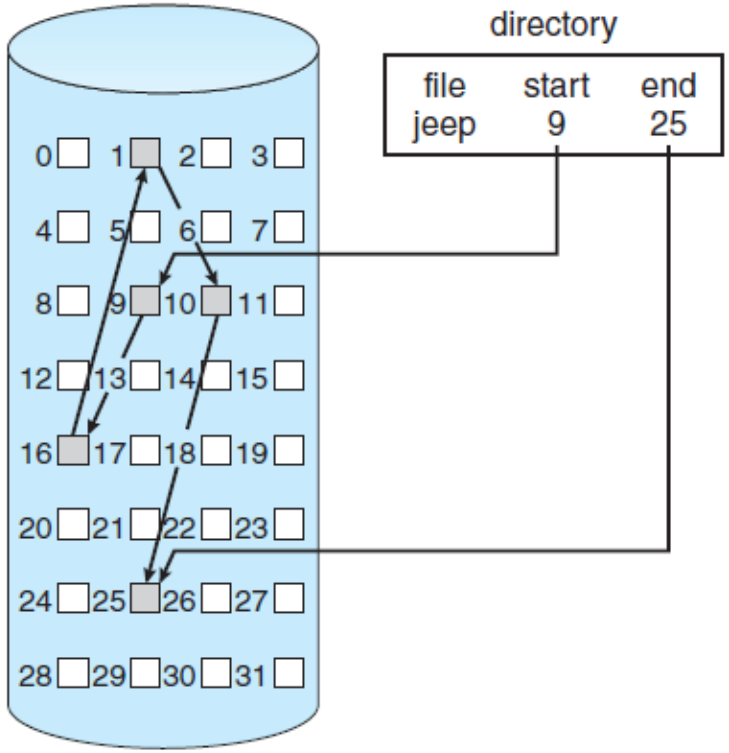
  - Non si ha spreco di spazio (no frammentazione esterna)
  - Quando necessita di un nuovo blocco da allocare ad un file (il file può crescere), il SO invoca il sottosistema per la gestione dello spazio libero
  - L'efficienza può essere migliorata raccogliendo i blocchi in cluster (aumenta, però, la frammentazione interna)
- Problemi
  - Accesso casuale impossibile: reperire un blocco può richiedere molte operazioni di I/O ed operazioni di seek
  - Affidabilità legata ai puntatori

Allocazione concatenata – 2

- Nella directory, si mantiene l'indirizzo dei blocchi iniziale e finale
- Mappatura da indirizzi logici ad indirizzi fisici

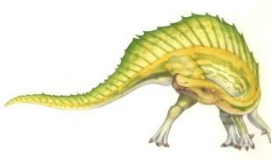
LA/511 → Q, R

Il blocco da accedere è il Q–esimo nella catena di blocchi che costituiscono il file; lo spostamento all'interno del blocco è pari ad R+1

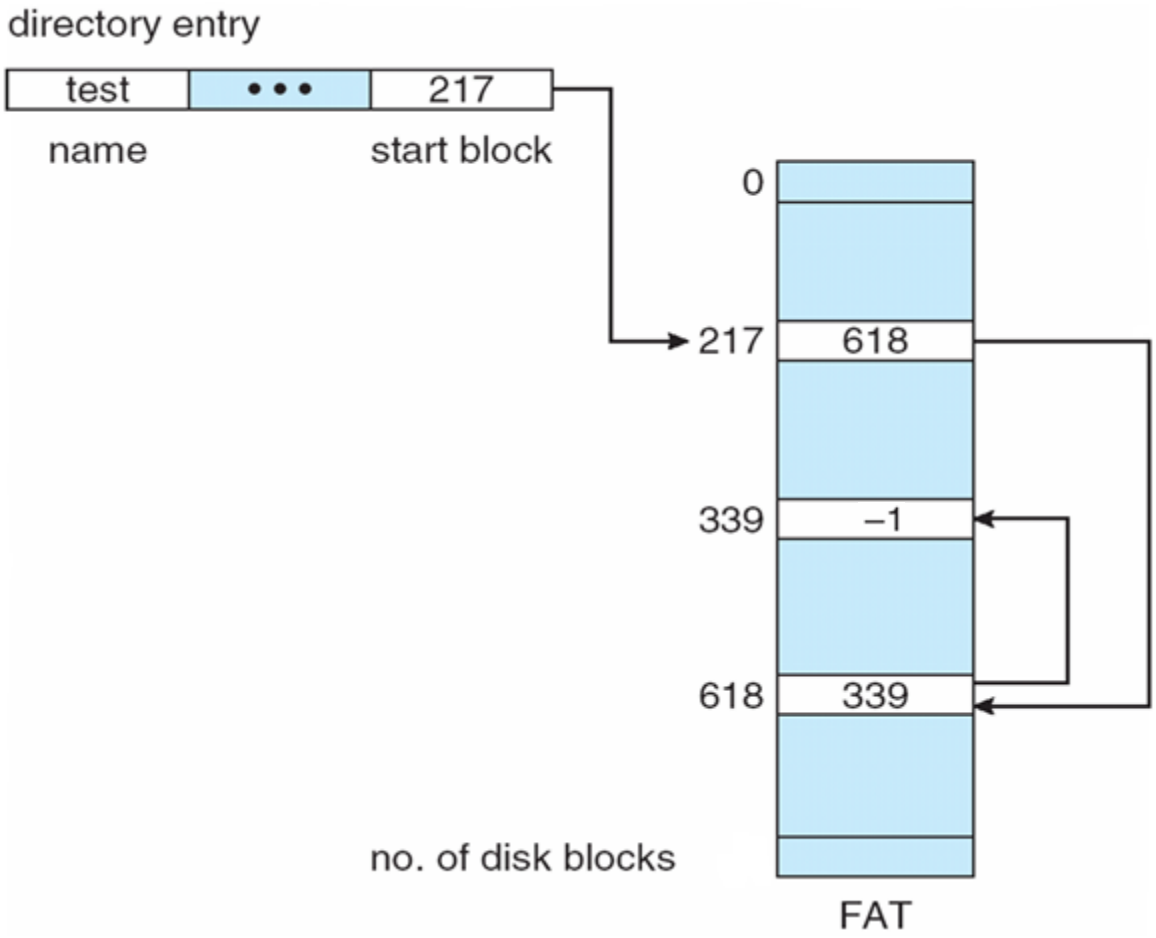


File Allocation Table (FAT) – 1

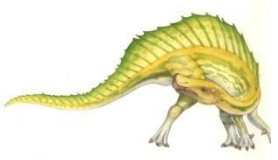
- Variante del metodo di allocazione concatenata implementata in MS–DOS e OS/2
  - Per contenere la FAT si riserva una sezione del disco all'inizio di ciascun volume
  - La FAT ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco
  - L'elemento di directory contiene il numero del primo blocco del file
    - L'elemento della FAT indicizzato da quel blocco contiene a sua volta il numero del blocco successivo del file
    - L'ultimo blocco ha come elemento della tabella un valore speciale di fine file
  - I blocchi inutilizzati sono contrassegnati dal valore 0
  - Facilita l'accesso casuale: informazione relativa alla locazione di ogni blocco "concentrata" nella FAT
    - Può essere soggetta a caching



FAT – 2

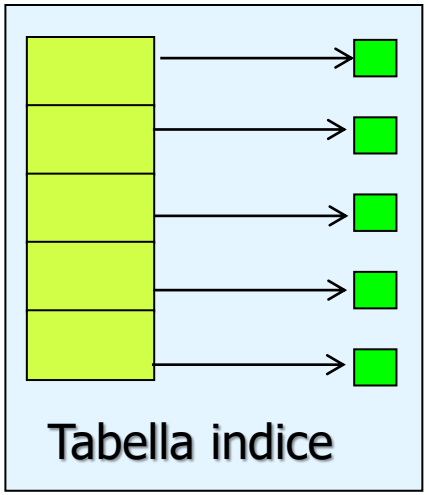


La FAT consente la memorizzazione "localizzata" dei puntatori

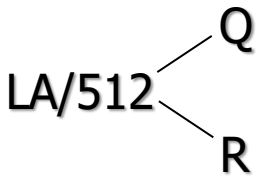


Allocazione indicizzata – 1

- Per ogni file, si collezionano tutti i puntatori in un unico blocco indice
- Richiede una tabella indice
- Accesso casuale
- Permette l'accesso dinamico senza frammentazione esterna; tuttavia c'è il sovraccarico temporale di accesso al blocco indice



- Nella directory si memorizza l'indirizzo del blocco indice
- Mappatura da indirizzi logici a indirizzi fisici per file di dim. max 256K parole e con dimensione di blocco di 512 parole: occorre un solo blocco indice

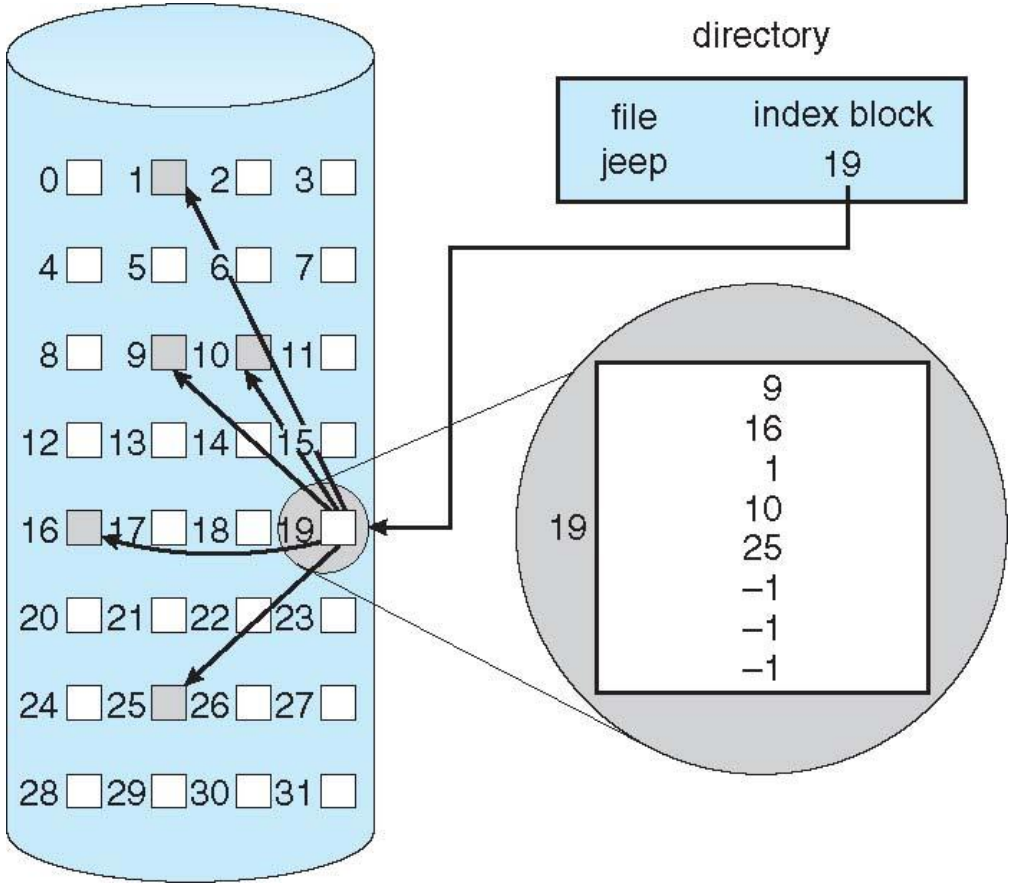


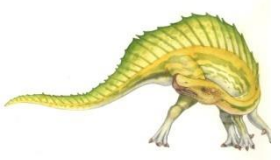
Q spostamento nella tabella indice

R spostamento all'interno del blocco

Allocazione indicizzata – 2

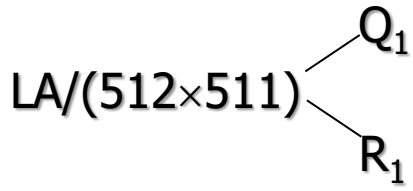
Nota: L'allocazione indicizzata soffre di alcuni degli stessi problemi dell'allocazione concatenata; infatti, i blocchi indice possono essere memorizzati in RAM, ma i blocchi dei dati possono essere distribuiti in tutto il volume





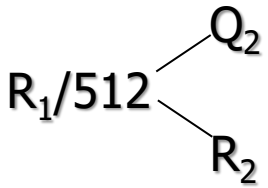
Allocazione indicizzata – 3

- Mapping fra indirizzi logici e fisici per un file di lunghezza qualunque (dim. blocco 512 byte/word)
- Schema concatenato — Si collegano blocchi della tabella indice
  - Il primo blocco indice contiene l'insieme degli indirizzi dei primi 511 blocchi del file, più un puntatore al blocco indice successivo



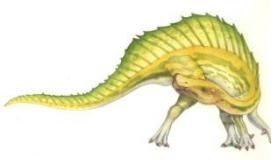
Q_1 spostamento nelle tabelle indice

R_1 si utilizza come segue…



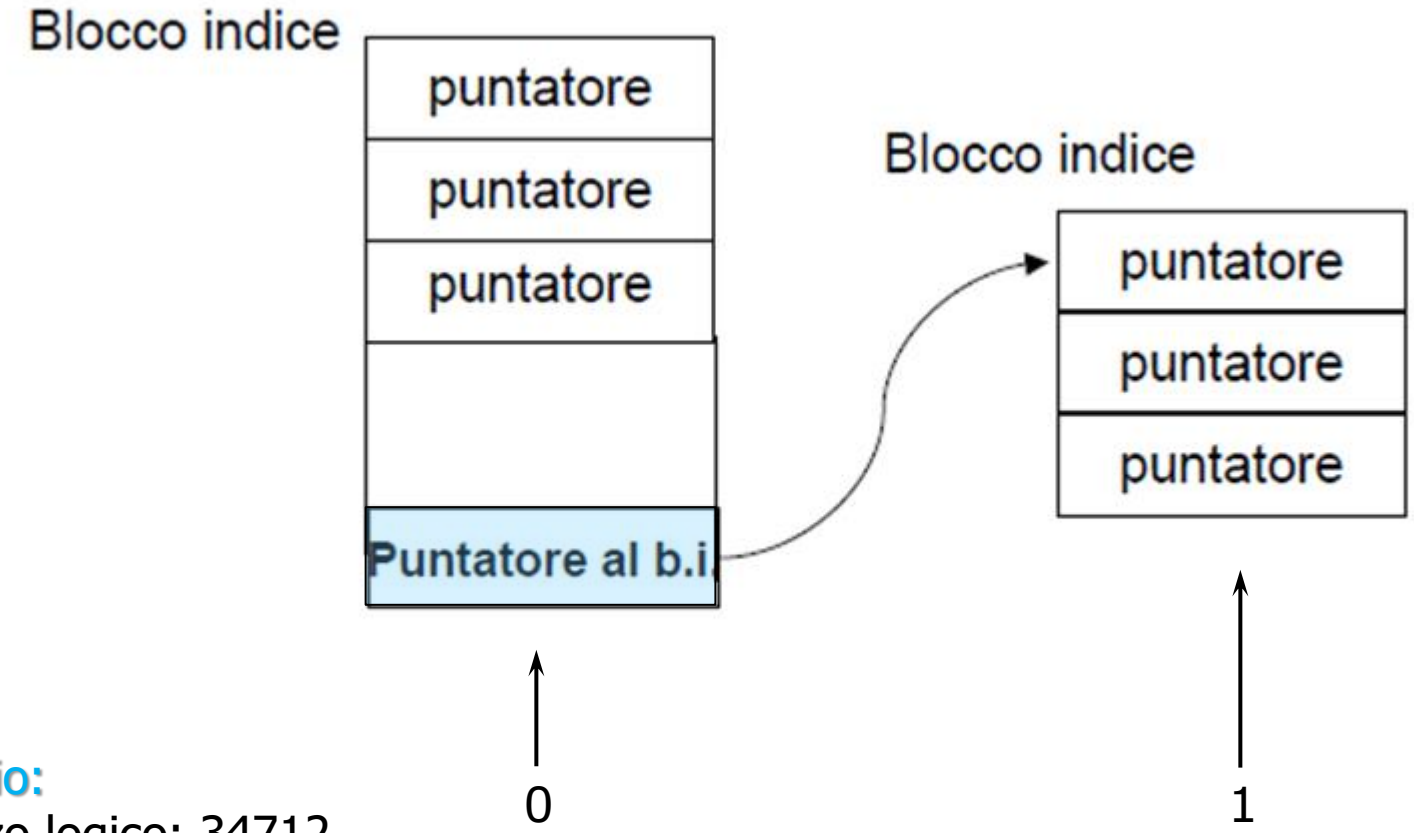
Q_2 spostamento nel blocco della tabella indice

R_2 spostamento all'interno del blocco del file

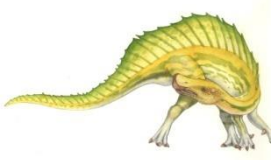


Allocazione indicizzata – 4

Indice concatenato



Esempio:
Indirizzo logico: 34712
34712/(512×511) = 0 con resto 34712
34712/512 → Blocco logico 67, offset 408



Allocazione indicizzata – 5

- Indice a più livelli
  - Indice a due livelli

$LA/(512 \times 512)$ → Q_1, R_1

Q_1 spostamento nell'indice esterno

R_1 utilizzato come segue…

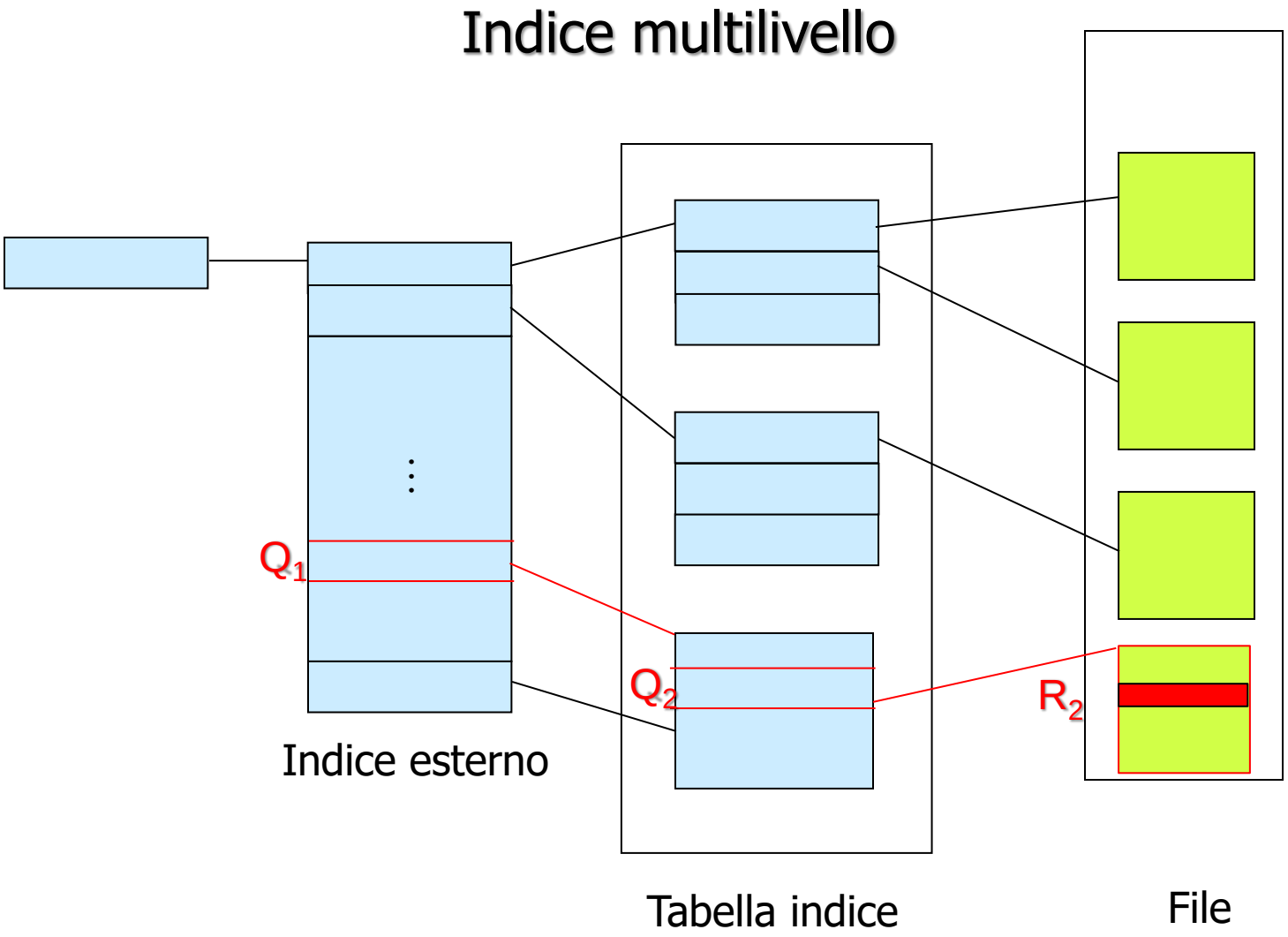
$R_1/512$ → Q_2, R_2

Q_2 spostamento nel blocco della tabella indice

R_2 spostamento nel blocco del file

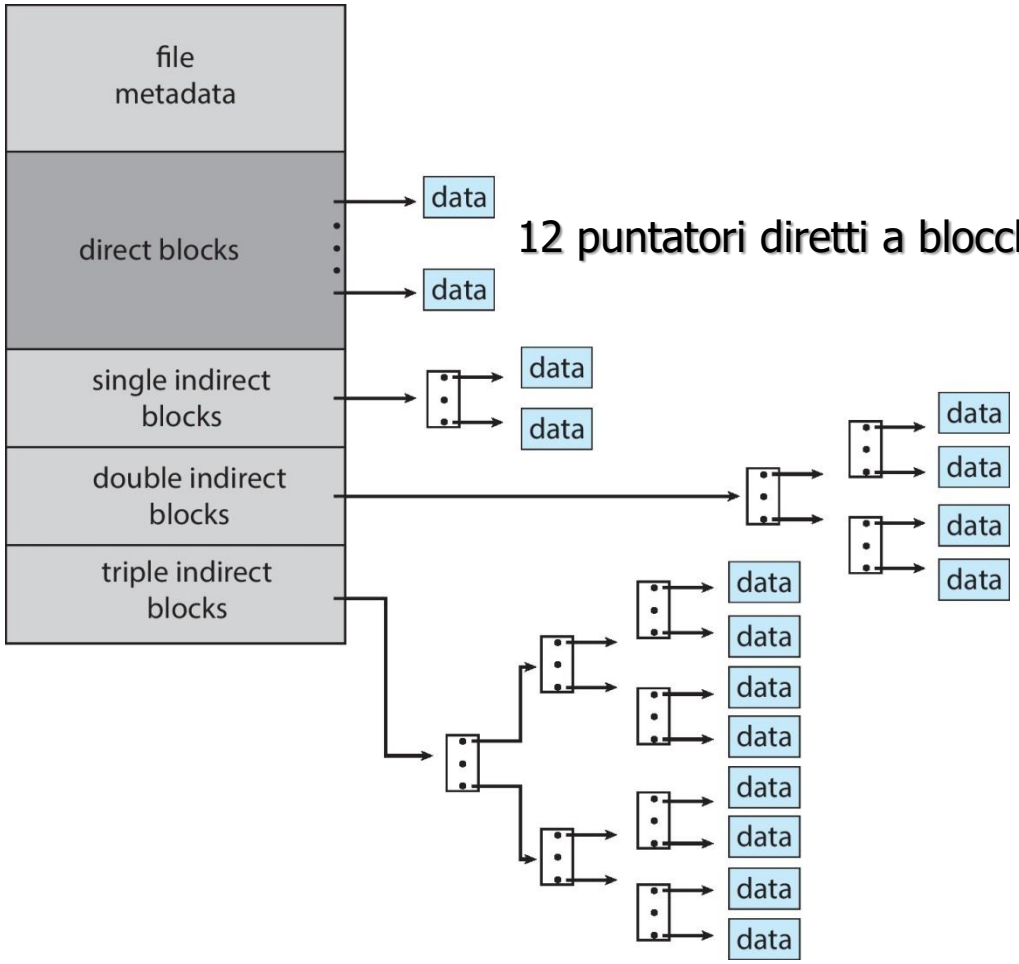
  - Blocchi da 4K possono contenere 1024 puntatori da 4 byte nell'indice esterno → per un totale di 1048567 blocchi di dati e file di dimensione massima pari a 4GB

Allocazione indicizzata – 6



Schema combinato: UNIX UFS

4Kbyte per blocco, indirizzi a 32 bit



12 puntatori diretti a blocchi di dati

Inode in UNIX

Scelta del metodo di allocazione – 1

- Il miglior metodo per l'allocazione di file dipende dal tipo di accesso
  - L'allocazione contigua ha ottime prestazioni sia per accesso sequenziale che casuale
  - L'allocazione concatenata si presta naturalmente all'accesso sequenziale
  - ⇨ Dichiarando il tipo di accesso all'atto della creazione del file, si può selezionare il metodo di allocazione più adatto
- L'allocazione indicizzata è "più complessa"
  - L'accesso ai dati del file può richiedere più accessi a disco (tre, nel caso di un indice a due livelli)
  - Tecniche di clustering possono migliorare il throughput, riducendo l'overhead di CPU

Scelta del metodo di allocazione – 2

- Aggiungere istruzioni per evitare anche un unico I/O da disco è "ragionevole"
  - L'Intel Core i7 6950x (2016) a 3 Ghz ⇨317900 MIPS
    - http://en.wikipedia.org/wiki/Instructions_per_second
  - La velocità tipica dei dischi odierni è pari a 250 I/O al secondo
    - 317900 MIPS/250 ≅ 1270 milioni di istruzioni durante un unico accesso a disco
- I dispositivi NVM non sono dotati di testine
  - Necessari algoritmi di ottimizzazione diversi
    - Cambia la struttura intrinseca del file system nell'ottica di ridurre il percorso complessivo tra il dispositivo di archiviazione e l'accesso dell'applicazione ai dati
  - SSD veloci permettono 60000 IOPS
    - 317900 MIPS/60000 = 5,3 milioni di istruzioni durante un I/O da SSD

Esempio 1

- Si consideri un file formato da 70 record e le sue possibili allocazioni su disco di tipo sequenziale, mediante lista concatenata e con tabella indice. In ognuno di questi casi i record del file sono memorizzati uno per blocco. Le informazioni che riguardano il file sono già in memoria centrale e la tabella indice è contenuta in unico blocco. Si dica quanti accessi al disco sono necessari, in ognuna delle seguenti situazioni, per cancellare:
  - Il primo record;
  - Il quarantesimo record;
  - L'ultimo record.

Esempio 1 (cont.)

- Soluzione
  - Allocazione sequenziale
    - Per cancellare il primo record serve solo aggiornare l'elemento di directory (nessun accesso a disco);
    - Per cancellare il quarantesimo record servono 30 letture e 30 scritture di blocchi: si legge il 41-esimo blocco e lo si copia nel 40-esimo, si legge il 42-esimo e lo scrive nel 41-esimo, etc.;
    - Per cancellare l'ultimo record serve solo aggiornare l'elemento di directory (nessun accesso a disco).
  - Allocazione concatenata (semplice)
    - Per cancellare il primo record serve leggerlo per aggiornare l'elemento di directory (un accesso a disco);
    - Per cancellare il quarantesimo record servono 40 letture e 1 scrittura per aggiornare il puntatore del blocco 39;
    - Per cancellare l'ultimo record servono 69 letture e 1 scrittura per aggiornare il puntatore del blocco 69 (che diventa **`null`**).
  - Allocazione indicizzata
    - In tutti i casi deve essere letto e riscritto il solo blocco indice.

Esempio 2

- Si consideri un file formato da 80 blocchi, allocato su disco in modo sequenziale. Il blocco precedente a quelli occupati dal file è occupato, mentre sono liberi i due blocchi successivi. Si dica quante operazioni di I/O su disco sono necessarie per aggiungere un blocco all'inizio del file.

- Soluzione
  Sono necessarie 161 operazioni di I/O su disco, 160 per spostare (leggere/scrivere) gli 80 blocchi del file, partendo dall'ultimo, più una per scrivere il nuovo blocco in testa al file.

Esempio 3

- Si consideri un file system con la dimensione di blocco logico di 2KB e con indirizzi di blocchi a 16 bit. Determinare la dimensione massima di un file in caso di
  - allocazione concatenata;
  - allocazione indicizzata con due livelli di blocco.
- Soluzione
  - Nel caso di allocazione concatenata, con puntatori a 16 bit si possono puntare 2^{16} blocchi, ognuno da 2KB (-2B, per il puntatore), per cui la dimensione massima del file è $2^{16} \times (2^{11} - 2)\text{B} = (2^{27} - 2^{17})\text{B} = 127\text{MB} - 127\text{KB}$.
  - Nel caso di allocazione indicizzata, in ogni blocco indice sono contenuti 1024 puntatori, per cui la dimensione massima del file è $(2^{10} \times 2^{10} \times 2^{11})\text{B} = 2^{31}\text{B} = 2\text{GB}$.

Esempio 4

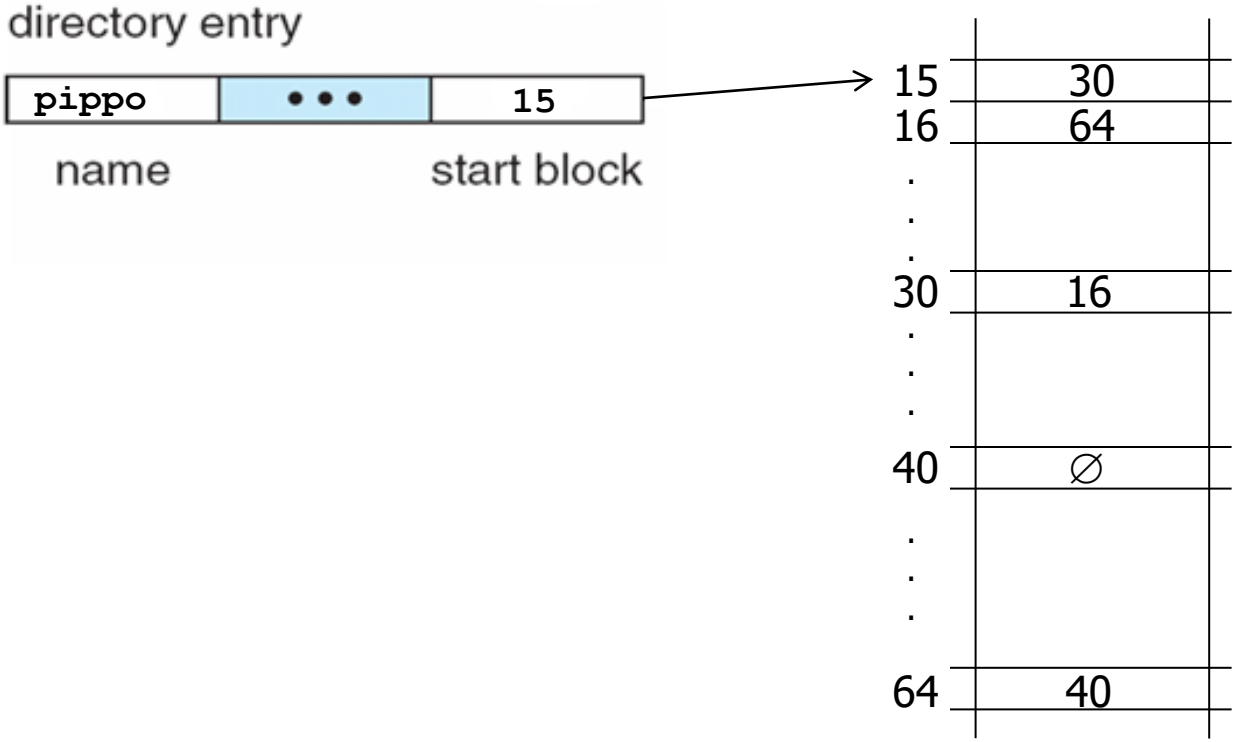
- Sia dato un file system Unix; sia 4096B la dimensione del blocco e p=64 bit la dimensione dell'indirizzo di blocco. Sia dato un file nel file system descritto. Il byte 312582 del suddetto file si trova in un blocco dati diretto, indiretto, doppiamente indiretto o triplamente indiretto?
- Soluzione
  - Con i 12 puntatori diretti si possono indirizzare file di al massimo 48KB;
  - Con il puntatore a singola indirezione si possono indirizzare ulteriori $(2^{12}/2^{3})=512$ blocchi da 4KB per un totale di $2^{9}\times 2^{12}$ B=2MB:
    - Il byte richiesto si trova in un blocco dati raggiungibile dall'indice singolo (nel 77-esimo blocco del file, ovvero in posizione 64 nell'indice).

Esempio 5

- In un disco con blocchi di 1 Kbyte è definito un file system FAT. Gli elementi della FAT sono in corrispondenza bi-univoca con i blocchi fisici del disco. Ogni elemento ha lunghezza di 3 byte e indirizza un blocco del disco. Ogni file è descritto da una lista concatenata di indirizzi di blocchi, realizzata sulla FAT. Il primo blocco di ogni file è identificato dalla coppia (*nomefile, indiceblocco)* contenuto nella rispettiva directory.
  - Qual è la massima capacità del disco, espressa in blocchi e in byte?
  - Quanti byte occupa la FAT?
  - Supponendo che il file **`pippo`** occupi i blocchi fisici 15, 30, 16, 64 e 40, quali sono gli elementi della FAT che descrivono il file e quale è il loro contenuto?

Esempio 5 (cont.)

- Soluzione
  - Poiché gli indirizzi sono a 3B si possono indirizzare al più 2^{24} blocchi da 1KB ciascuno, per un totale di 2^{34}B=16GB
  - La FAT occupa $2^{24}\times$3B=48MB.
  - Elementi "significativi" della FAT:

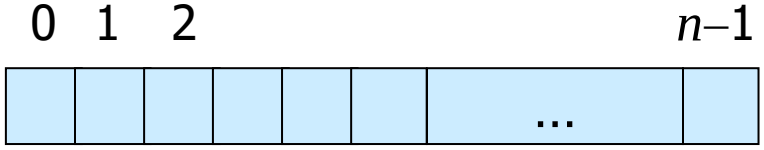


Gestione dello spazio libero – 1

- Per tenere traccia dello spazio libero in un disco, il sistema conserva una lista dello spazio libero
  - Per creare un file occorre…
    - cercare nella lista dello spazio libero la quantità di spazio necessaria ed allocarla al nuovo file
    - aggiornare il contenuto della lista
  - Quando si cancella un file, si aggiungono alla lista dello spazio libero i blocchi di disco precedentemente assegnati al file
- La lista dello spazio libero può non essere realizzata come una lista

Gestione dello spazio libero – 2

- Vettore di bit o bitmap (n blocchi)



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{blocco}[i] \text{ libero} \\ 0 \Rightarrow \text{blocco}[i] \text{ occupato} \end{cases}$$

- Calcolo del numero del primo blocco libero: si scorre il vettore, cercando il primo byte/parola diverso/a da 0

  (numero di bit per parola) $*$
  (numero di parole con valore 0) +
  offset del primo bit a 1

- Buone prestazioni se il vettore è conservato in memoria centrale

Gestione dello spazio libero – 3

- Difficoltà nel mantenere la **bitmap** in RAM per i dischi attuali
  - Esempio

    dim. blocco $= 2^{12}$ byte (4 kilobyte)

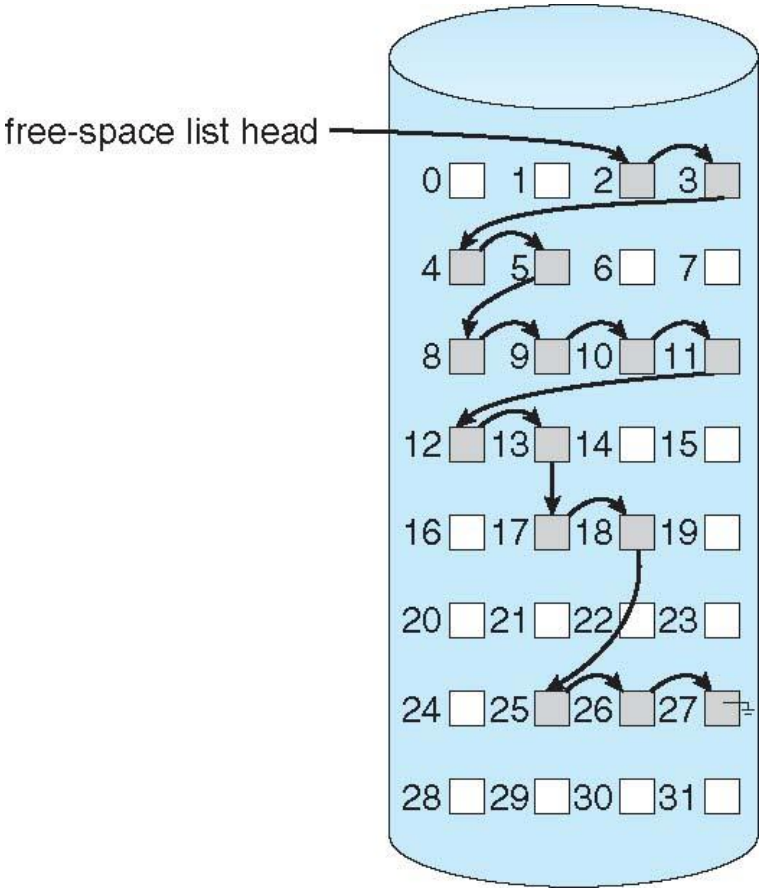
    dim. disco $= 2^{40}$ byte (1 terabyte)

    $n = 2^{40}/2^{12} = 2^{28}$ blocchi ⇨ 2^{28} bit (o 32 Mbyte)

    per cluster da 4 blocchi ⇨ 8 MB di memoria
- È adatta per gestire file contigui

Gestione dello spazio libero – 4

- Lista concatenata
  - Si collegano tutti i blocchi liberi mediante puntatori e si mantiene un puntatore alla testa della lista in memoria centrale
    - Ogni blocco contiene un puntatore al successivo blocco libero
  - Non si spreca spazio (solo un puntatore)
  - Non è necessario attraversare tutta la lista, poiché di solito la richiesta è relativa ad un singolo blocco
  - Non facile da usare per ottenere spazio contiguo



- Nella FAT, il conteggio dei blocchi liberi è incluso nella struttura dati per l'allocazione (blocchi contrassegnati con 0) e non richiede quindi un metodo di gestione separato

Gestione dello spazio libero – 5

- Grouping
  - Realizzazione di una lista di blocchi
  - Sul primo blocco: memorizzazione degli indirizzi di n blocchi liberi; $n-1$ blocchi sono effettivamente liberi, l'n–esimo contiene gli indirizzi di altri n blocchi, etc.
- Conteggio
  - Poiché lo spazio viene spesso allocato e liberato in modo contiguo (nell'allocazione contigua, per gli extent o nel caso di cluster)
  - ⇨ si mantiene una lista contenente un indirizzo del disco ed un contatore, che indica il numero di blocchi liberi contigui (più informazione su ogni elemento della lista, ma lista più breve)

TRIM dei blocchi non utilizzati – 1

- TRIM è un comando ATA (Advanced Technology Attachment) che consente al SO di informare il controller di un SSD su quali blocchi di dati può cancellare, perché non sono più in uso
- Dalla prospettiva dell'utente, i dati sono stati cancellati; tuttavia, dato il modo in cui gli SSD scrivono informazioni, i dati non vengono cancellati dall'unità al comando dell'utente, ma l'area che contiene i dati viene contrassegnata come non più in uso
- Senza TRIM, l'SSD non saprebbe che alcuni blocchi nell'unità contengono informazioni non valide finché il computer non comunica all'unità di scrivere nuove informazioni in quel punto
  - L'unità dovrebbe cancellare le informazioni esistenti quindi scrivere le nuove informazioni

TRIM dei blocchi non utilizzati – 2

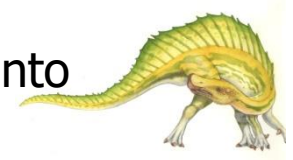
- TRIM è complementare alla garbage collection
  - Elimina la copiatura di pagine di dati scartate o non valide durante il processo di garbage collection per risparmiare tempo e migliorare le prestazioni dell'unità SSD
  - L'SSD ha un minor numero di pagine da spostare durante la garbage collection, il che riduce il numero totale di cicli di programmazione/cancellazione sul supporto flash NAND e prolunga la vita dell'SSD
  - In altre parole: TRIM suggerisce quali celle possono essere cancellate durante il tempo di inattività, il che consente all'unità di organizzare le celle rimanenti (valide e vuote) per un uso bilanciato (livellamento dell'usura)

Efficienza e prestazioni – 1

- L'efficienza del file system dipende da:
  - Tecniche di allocazione del disco e algoritmi di realizzazione/gestione delle directory
  - Preallocazione delle strutture necessarie a mantenere i metadati
  - Tipi di dati conservati nell'elemento della directory corrispondente al file
  - Strutture dati a lunghezza fissa o variabile

Efficienza e prestazioni – 2

- Esempi
  - In UNIX, gli inode sono preallocati e distribuiti nel disco, per mantenere dati e metadati vicini e diminuire il tempo di seek
  - Se, nell'elemento di directory, si mantiene la data di ultimo accesso ad un file per consentire all'utente di risalire all'ultima volta che un file è stato letto…
    - ⇨Ogni volta che si apre un file per la lettura, si deve leggere e scrivere anche l'elemento della directory ad esso associato
  - Se si aumenta la dimensione dei puntatori, si aumenta la dimensione della memoria gestibile via file system, ma si aumenta anche la dimensione delle strutture dati necessarie per l'allocazione dei blocchi e la gestione dello spazio libero
  - In Solaris, originariamente, la dimensione delle tabelle dei processi e dei file aperti era fissa
    - ⇨Raggiunto il limite massimo, il SO non poteva supportare più processi o aprire nuovi file
    - ⇨Allocazione dinamica, algoritmi più complessi e SO più lento

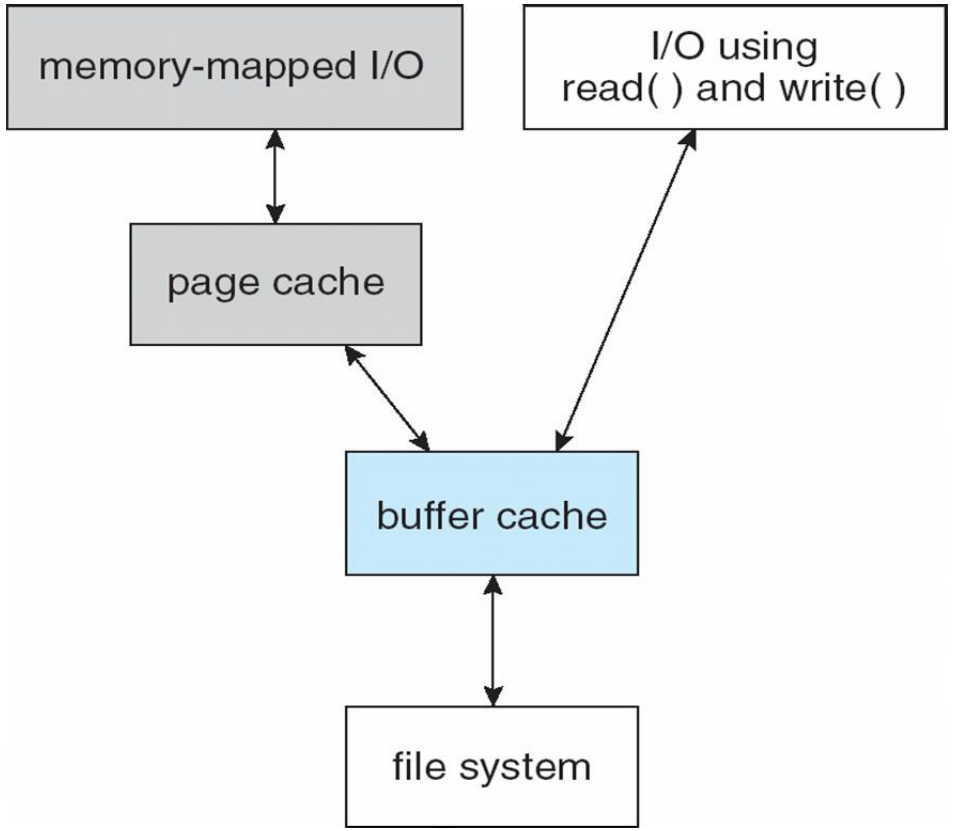


Efficienza e prestazioni – 3

- Le prestazioni dipendono da:
  - Disporre di buffer cache, cioè sezioni dedicate della memoria in cui si conservano i blocchi usati di frequente
  - Scritture sincrone talvolta richieste dalle applicazioni o necessarie al sistema operativo
    - Impossibilità di buffering/caching: l'operazione di scrittura su disco deve essere completata prima di proseguire l'esecuzione
    - Le scritture asincrone, che sono le più comuni, sono invece bufferizzabili (e più veloci)
  - Le operazioni di lettura risultano talvolta più lente delle scritture
  - Utilizzo di tecniche di svuotamento/riempimento delle cache per ottimizzare l'accesso sequenziale

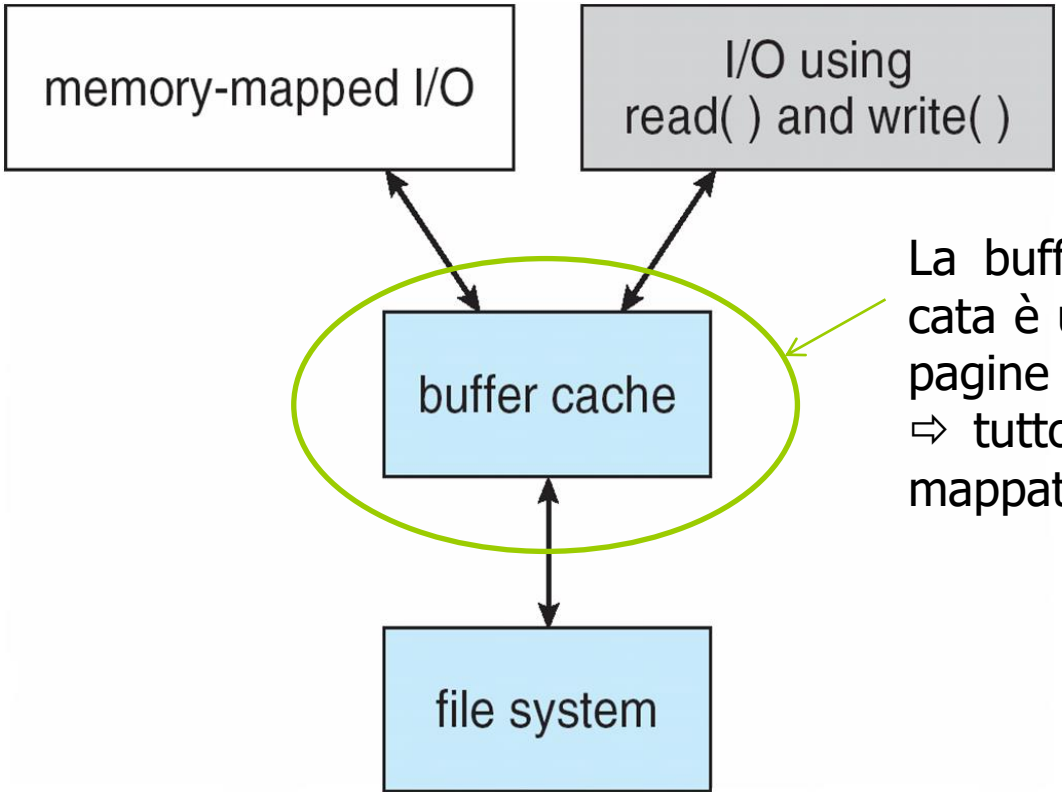
Cache delle pagine

- Una cache delle pagine impiega tecniche di memoria virtuale per la gestione dei dati dei file, trattandoli alla stregua di pagine, anziché come blocchi del disco
- L'I/O mappato in memoria impiega una cache delle pagine, mentre l'I/O da file system utilizza la buffer cache riservata al disco (in RAM)



Buffer cache unificata

- Una buffer cache unificata, invece, prevede l'utilizzo di un'unica cache per memorizzare sia i file mappati in memoria che i blocchi trasferiti per operazioni di I/O ordinario da file system, evitando il double caching



La buffer cache unificata è una cache delle pagine
⇨ tutto l'I/O da disco mappato in memoria

Ancora sulle prestazioni

- L'algoritmo LRU è in generale ragionevole per la sostituzione delle pagine e dei blocchi in cache; tuttavia…
  - Le pagine relative ad un file da leggere o scrivere in modo sequenziale non si dovrebbero sostituire nell'ordine LRU, dato che la pagina usata più di recente non verrà probabilmente più utilizzata
  - Il rilascio indietro o *free–behind* rimuove una pagina dalla cache non appena si verifica una richiesta della pagina successiva
  - Con la lettura anticipata o *read–ahead* si leggono e si copiano nella cache la pagina richiesta e diverse pagine successive, che verranno probabilmente accedute in sequenza

Ripristino – 1

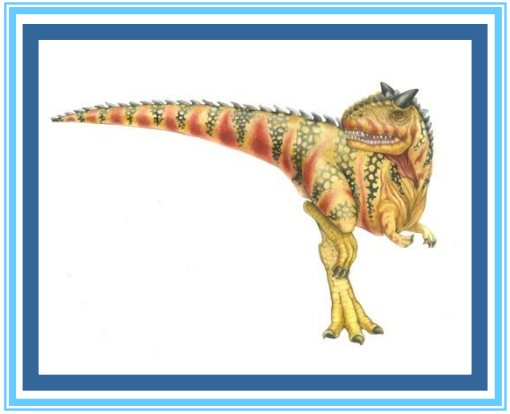
- Poiché i file e le directory sono mantenuti sia in memoria RAM (parzialmente) sia nei dischi, è necessario assicurarsi che malfunzionamenti del sistema non comportino la perdita di dati o la loro incoerenza
- Esempio: all'atto della creazione di un file
  - Modifica dell'elemento di directory
  - Allocazione blocchi di dati e FCB
  - Aggiornamento delle informazioni (puntatori) blocchi liberi e FCB liberi

  Se si ha un crollo del sistema si possono avere incoerenze fra le strutture
  - Il contatore degli FCB liberi potrebbe indicare un FCB allocato…
  - …ma la directory non contiene un puntatore all'elemento relativo

Ripristino – 2

- Verificatore di coerenza — (*fsck* in UNIX, *chkdsk* in DOS/Windows) confronta i dati nella struttura di directory con i blocchi di dati sul disco e tenta di fissare le eventuali incoerenze
  - Per esempio, se si perde un elemento di directory:
    - si può ricostruire un file con allocazione concatenata
    - …ma non uno allocato tramite indice
- Si impiegano programmi di sistema per copiare (back–up) dati dal disco ad un altro dispositivo di memorizzazione (altri dischi magnetici, supporti ottici, etc.)
- Si recuperano file persi o il contenuto di dischi danneggiati ricaricandoli dal back–up

Appendice
Il Network File System



Network File System (NFS) – 1

- Rappresenta sia una realizzazione che una definizione di un sistema per accesso a file remoti attraverso LAN o WAN
- Nasce in ambiente UNIX (Solaris e SunOS) ed usa i protocolli UDP/IP (Unreliable Datagram Protocol su Ethernet) o TCP/IP, secondo la rete di comunicazione
- È supportato da Linux

NFS – 2

- Nel contesto dell'NFS si considera un insieme di stazioni di lavoro interconnesse come un insieme di calcolatori indipendenti con file system indipendenti
  - Garantire (un certo grado di) condivisione tra i file system, su richiesta esplicita, in modo trasparente
  - Una directory remota viene montata su una directory del file system locale
    - La directory montata assume l'aspetto di un sottoalbero integrante del file system locale e sostituisce il sottoalbero che discende dalla directory locale

NFS – 3

- La directory remota si specifica come argomento dell'operazione di montaggio in modo esplicito: occorre fornire la locazione (o il nome del calcolatore) della directory remota
  - I file nella directory remota divengono quindi accessibili in modo del tutto trasparente
- Potenzialmente, ogni file system, o ogni directory in un file system, nel rispetto dei diritti di accesso, può essere montato in modo remoto su qualsivoglia directory locale

NFS – 4

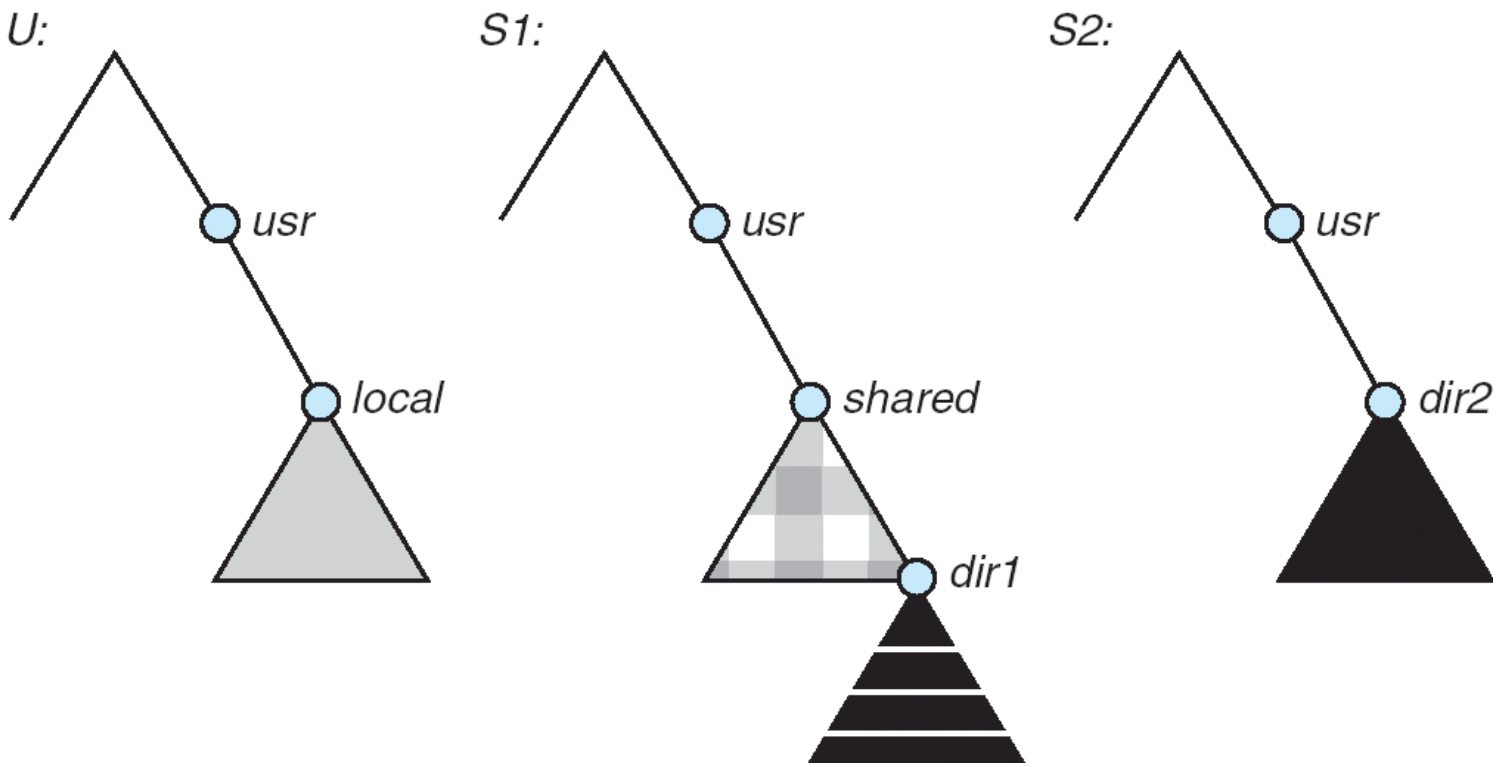
- L'NFS è progettato per operare in un ambiente eterogeneo di calcolatori, sistemi operativi e architetture di rete: la realizzazione di NFS è indipendente dall'ambiente hardware/software che fa da substrato al file system
- Tale indipendenza si ottiene utilizzando primitive RPC costruite su un protocollo di rappresentazione esterna dei dati (External Data Representation, XDR) usato tra interfacce indipendenti
- La definizione di NFS distingue tra i servizi offerti dal meccanismo di montaggio (protocollo di montaggio) e gli effettivi servizi di accesso ai file remoti (protocollo NFS)

NFS – 5

- Vantaggi
  - In locale, si usa meno spazio disco, perchè i dati possono essere conservati su una singola macchina e restano accessibili a tutte le altre macchine connesse alla rete
  - Gli utenti non devono avere home directory separate su ogni macchina in rete: le home directory possono essere poste sul server NFS e rese disponibili attraverso la rete
  - I dispositivi di archiviazione come unità CD–ROM e USB possono essere utilizzati dagli altri computer della rete: riduzione del numero di unità per supporti rimovibili presenti nella rete

Mounting in NFS – 1

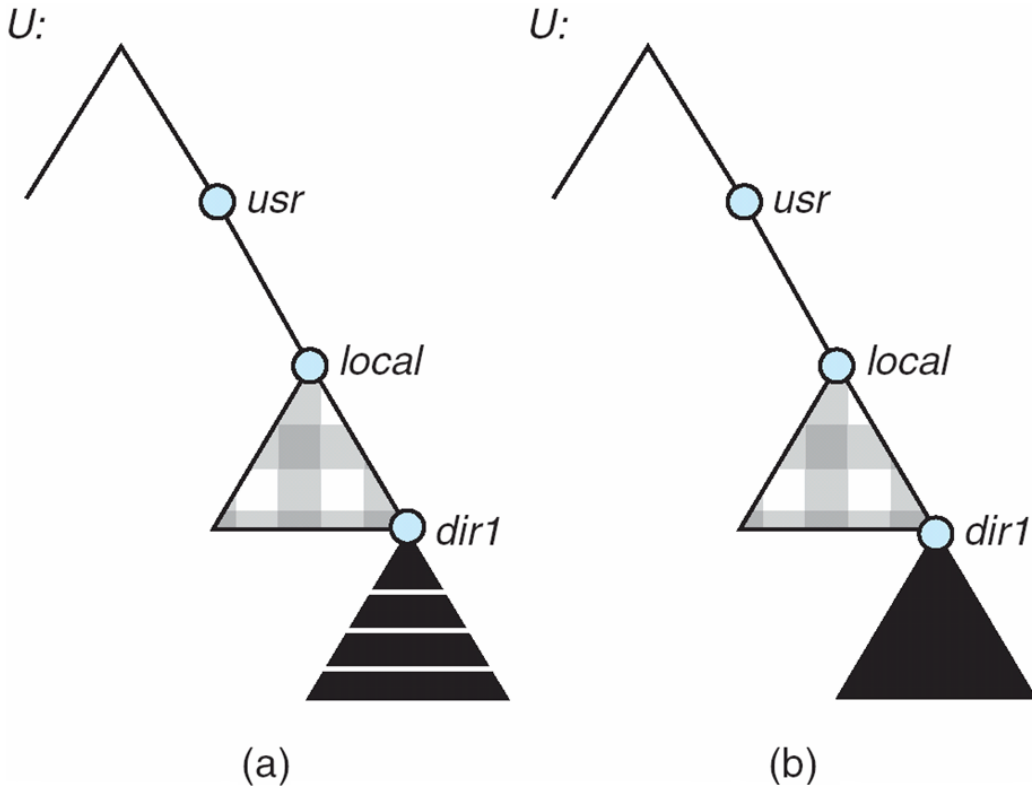
- Tre file system indipendenti



Mounting in NFS – 2

- Dato un insieme di stazioni di lavoro C_i connesse e dotate ciascuna del proprio file system, si distinguono due fasi:
  - Montaggio: C_1 monta esplicitamente una directory **dir** di C_2 su di una directory **loc** del proprio file system
  - Accesso: gli utenti di C_1 accedono trasparentemente a **dir** facendo riferimento a **loc**

Mounting in NFS – 3

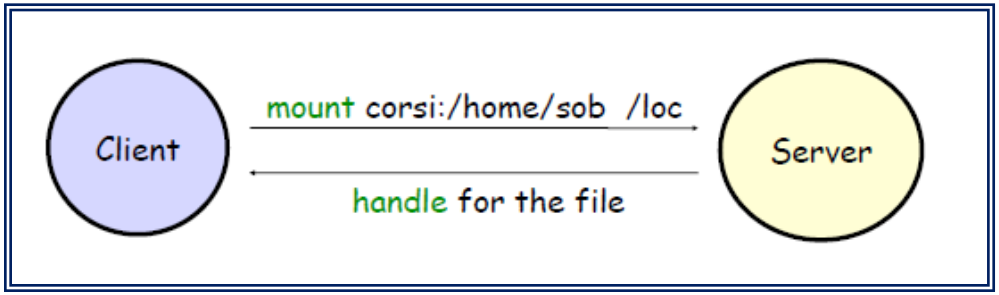


Montaggio del file system remoto S1:/usr/shared in U:/usr/local

Montaggio a cascata di S2:/usr/dir2 in U:/usr/local/dir1

Protocollo di montaggio in NFS – 1

- Stabilisce la connessione logica iniziale tra un server ed un client



  - L'operazione di montaggio comprende il nome della directory remota da montare ed il nome del server in cui tale directory è memorizzata
  - La richiesta di montaggio si associa alla RPC corrispondente e si invia al processo server di montaggio in esecuzione sullo specifico calcolatore server
  - Lista di esportazione – specifica i file system locali esportati per il montaggio e i nomi dei calcolatori a cui tale operazione è permessa

Protocollo di montaggio in NFS – 2

- Quando il server riceve una richiesta di montaggio conforme alla propria lista di esportazione, riporta al client un **file handle**, da utilizzare per tutti i successivi accessi a file che si trovano nel file system montato
- File handle — contiene un identificatore del file system esportato ed un numero di inode per identificare la directory montata all'interno dello stesso
- L'operazione di montaggio cambia solo la vista che il client ha del file system, ma non provoca nessuna modifica nello stato del server
- Altre operazioni del protocollo:
  - Unmount
  - Fornire lista di esportazione

Protocollo NFS – 1

- Offre un insieme di RPC per operazioni su file remoti, che svolgono le seguenti operazioni:
  - Ricerca di un file in una directory
  - Lettura di un insieme di elementi di directory
  - Gestione di link e directory
  - Accesso agli attributi dei file
  - Lettura e scrittura di file
- Si osservi che mancano **open** e **close**:
  - il protocollo, fino alle versione 3, è **stateless**: il server non conserva informazioni sul client da un accesso all'altro (la v4, con stato, è significativamente diversa)
  - non c'è tabella dei file aperti sul server

Protocollo NFS – 2

- I server NFS sono privi di stato: ogni richiesta deve fornire un insieme completo di argomenti, tra cui un identificatore unico di file e un offset assoluto all'interno del file, per svolgere le operazioni appropriate
- I dati modificati devono essere salvati sul disco del server prima che i risultati siano riportati al client (decadimento delle prestazioni perché si perdono i vantaggi della memorizzazione in cache)

NFS e concorrenza

- NFS garantisce l'atomicità delle singole operazioni di scrittura e la non interferenza con altre scritture
- Non fornisce alcun meccanismo di controllo della concorrenza
  - Due utenti che scrivono sullo stesso file remoto possono riscontrare interferenze nei loro dati
  - L'implementazione di meccanismi di lock richiederebbe informazioni sul client lato server
- Per coordinare l'accesso ai file gli utenti devono utilizzare meccanismi esterni a NFS

Daemon – 1

- È probabilmente l'acronimo di "Disk And Execution MONitor" (sorvegliante di disco ed esecuzione), un processo che girava allora sulle macchine Unix (ed è ancora oggi presente nei sistemi da esso derivati)
- Piccolo programma, non direttamente visibile dall'utente e normalmente in stato "di riposo" (non consuma risorse di CPU), che risiede stabilmente nella memoria del sistema e si occupa di compiti specifici, come ad esempio la gestione dei processi di stampa o il trasferimento di dati attraverso una porta fisica di comunicazione, connessa ad una periferica di input/ output
- In un server HTTP, il daemon attende, accetta ed inoltra le richieste dei browser degli utenti collegati, compresi i trasferimenti di e–mail

Daemon – 2

- Daemon attivi sul server
  - mountd: È il demone che si occupa di gestire il montaggio del file system di rete dal lato del server; generalmente, viene avviato dalla procedura di inizializzazione del sistema
    - Mantiene il file `/etc/rmtab` che elenca i montaggi in essere; tuttavia, non è garantito che il contenuto del file sia esatto, per cui non lo si può utilizzare per determinare con certezza quali siano le connessioni in corso
  - nfsd: È il demone che si occupa di gestire le richieste, da parte dei client, per i servizi NFS; viene avviato generalmente dalla procedura di inizializzazione del sistema, subito dopo mountd

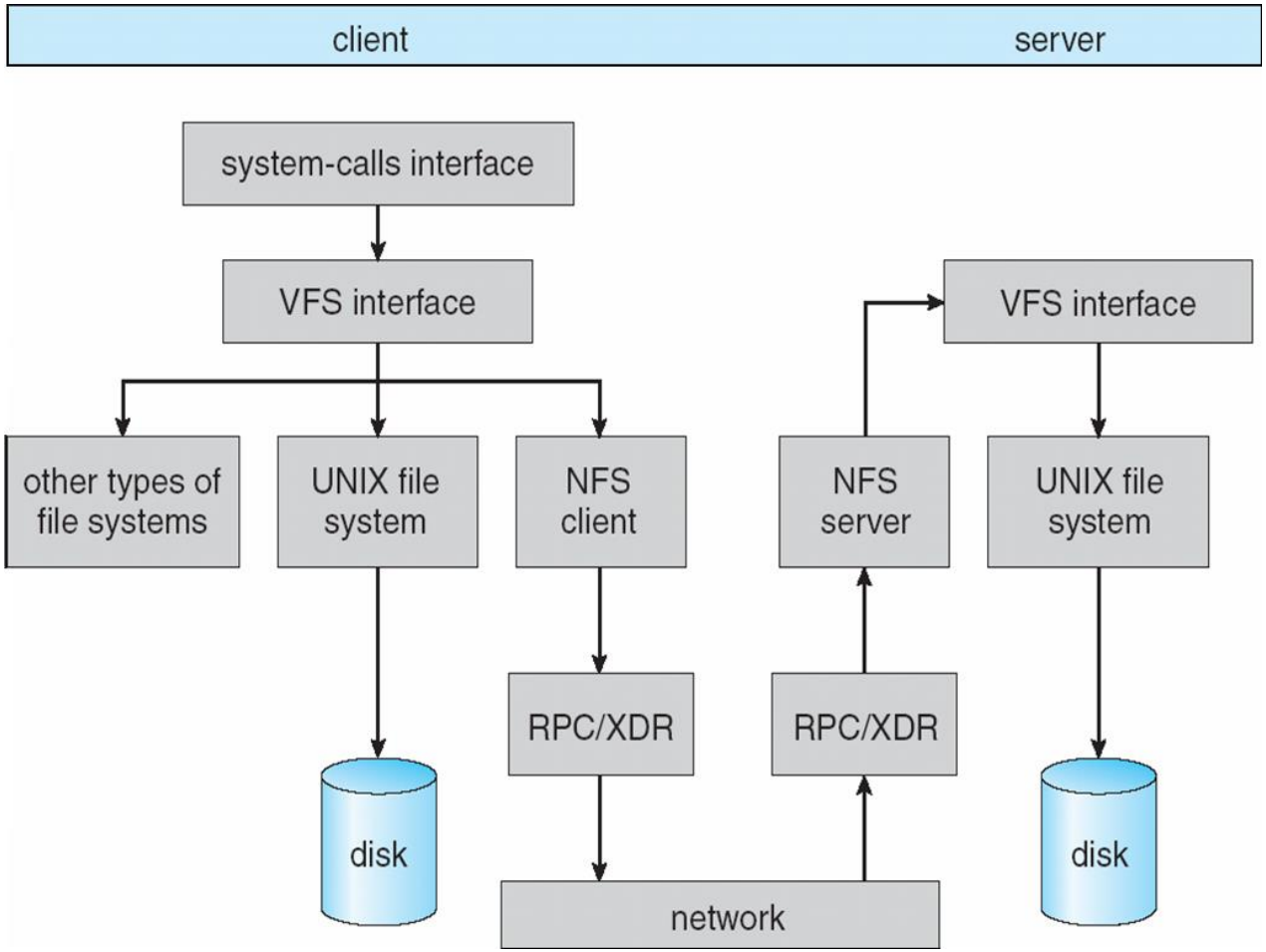
Daemon – 3

- Inoltre…
  - Il file **`/etc/exports`** contiene l'indicazione delle porzioni di filesystem locale da concedere in condivisione alla rete; viene utilizzato dai demoni **mountd** e **nfsd**
  - Se il file non c'è o è vuoto, non viene concesso l'utilizzo di alcuna parte del file system locale all'esterno
  - Ogni record del file è composto da:
    - l'indicazione di una directory a partire dalla quale si concede la condivisione
    - una serie di nodi o reti cui viene concesso l'utilizzo della directory con eventuali specifiche di accesso
- Daemon attivi sul client
  - **nfslogd:** supporta il logging sul server remoto

Architettura a strati di NFS

- L'NFS è integrato nel SO tramite un VFS
- Interfaccia al file system UNIX (basata sulle chiamate di sistema **`open`**, **`read`**, **`write`**, **`close`** e sui descrittori di file – inode)
- Strato del Virtual File System (VFS) — distingue i file locali dai file remoti; i file locali vengono ulteriormente distinti in base al tipo di file system cui appartengono
  - Il VFS attiva le operazioni opportune sul particolare file system locale, o…
  - …invoca l'opportuna procedura NFS per servire la richiesta remota
- Strato dei servizi NFS — implementa il protocollo NFS

Schema dell'architettura NFS



Traduzione dei pathname

- Si compie suddividendo il percorso (path) in nomi componenti ed eseguendo una chiamata al servizio di lookup di NFS per ogni nome di componente e vnode di directory
- Una cache per la ricerca dei nomi delle directory, nel sito del client, conserva i vnode per i nomi delle directory remote
  - ⇨ Si accelerano i riferimenti ai file con lo stesso percorso iniziale

Funzionamento remoto

- Eccetto che per l'apertura e la chiusura dei file, esiste una corrispondenza uno ad uno tra le system call di UNIX e le RPC del protocollo
- L'NFS aderisce al paradigma del servizio remoto, ma utilizza tecniche di memorizzazione transitoria e cache per migliorare le prestazioni
- Non rispetta la semantica della coerenza di UNIX
  - File nuovi creati in un server possono non essere visibili nel client per 30 secondi
  - Non è stabilito se e quando le scritture di un client siano visibili agli altri che hanno aperto il file in lettura
  - Le nuove open sul file vedono solo le modifiche già inviate al server

Esercizi

- **Esercizio 1**
  Si consideri la chiamata **`read(4,buf,2000)`** di un sistema UNIX–like, dove il file descriptor 4 corrisponde all'i–node 15. L'i–node contiene 5 indirizzi di blocchi diretti, che hanno rispettivamente valore 512, 567, 45, 34, 28, oltre agli indirizzi di 2 blocchi indiretti. I primi 5 indirizzi del blocco indiretto semplice sono 56, 47, 67, 89, 23. I blocchi del disco hanno ampiezza pari a 1024 byte e la lunghezza corrente del file è di 10.000 byte. Il puntatore alla posizione corrente di lettura ha il valore 8500.
  - Quali blocchi fisici vengono letti per eseguire l'operazione?
  - Quanti caratteri vengono copiati in **`buf`** da ogni blocco interessato alla lettura?
  - Qual è il valore intero restituito dalla chiamata (ovvero il numero totale di caratteri letti)?

Esercizi (Cont.)

- **Esercizio 2**

  Un file system può tenere traccia dei blocchi liberi utilizzando una bitmap o una lista concatenata. Supponendo che nel sistema siano complessivamente disponibili T blocchi, U dei quali sono attualmente utilizzati, e che l'informazione relativa a ciascun blocco, nel caso di gestione tramite lista occupi S bit, si calcoli la dimensione in bit della bitmap e della lista dei blocchi liberi. Fissati S e T, si determini inoltre per quale valore di U l'occupazione della bitmap è inferiore a quella della lista dei blocchi liberi.

- **Esercizio 3**

  Calcolare la dimensione (in byte) della FAT per un disco da 512MB con blocchi da 16 KB e indirizzi dei blocchi a 16 bit. Quanti blocchi occuperebbe la FAT se memorizzata su disco? Quanti accessi alla memoria occorrono per recuperare il byte 125384 di un certo file del file system in questione?

Fine del Capitolo 13