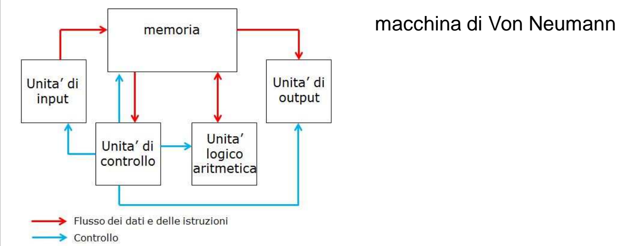
LABORATORIO DI PROGRAMMAZIONE 2
Corso di laurea in matematica

Sistemi Operativi : memoria secondariae file system

Marco Lapegna
Dipartimento di Matematica e Applicazioni
Universita' degli Studi di Napoli Federico II

principali componenti di un calcolatore



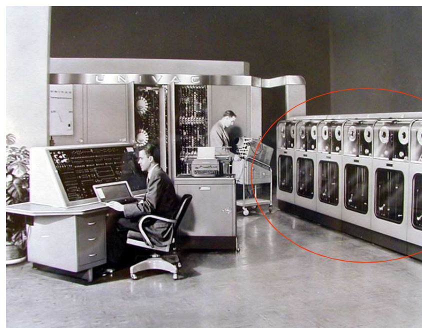
macchina di Von Neumann

- CPU (UC + ALU) → gestione dei processi
- dispositivi di I/O
- mamoria
  - RAM → gestione della memoria virtuale
  - HD → gestione del file system e dei RAID

La memoria secondaria

- Oltre alla memoria centrale (veloce, di piccole dimensioni e volatile) i calcolatori necessitano di un supporto di memorizzaione di piu' grandi dimensioni e non volatile

La memoria secondaria

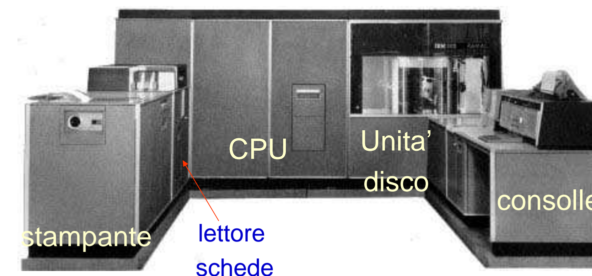


I primi calcolatori utilizzavano come supporto di memorizzazione secondaria i nastri magnetici

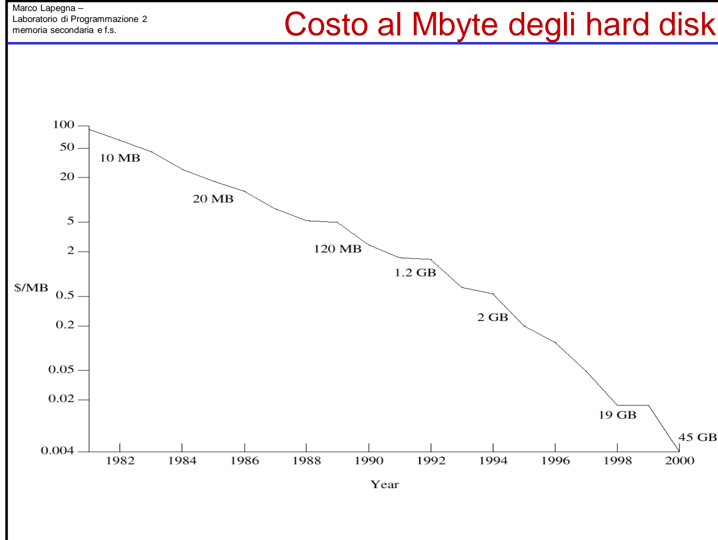
1951: UNIVAC 1

La memoria secondaria

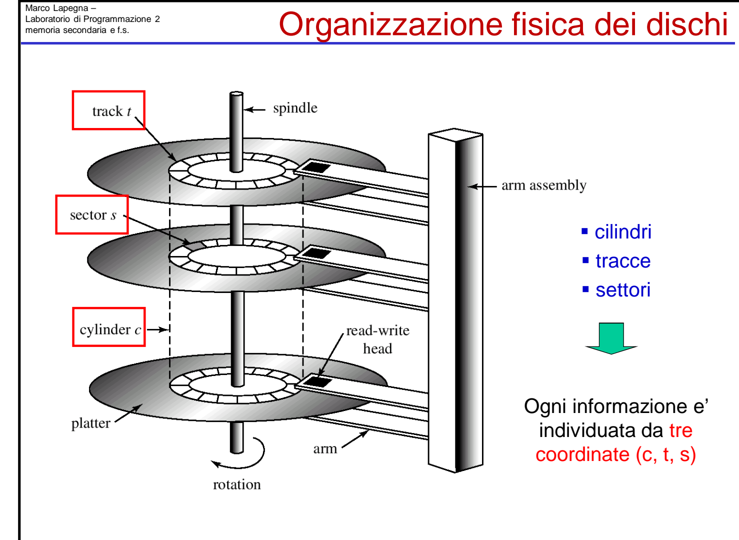
- Il principale problema dei nastri e' il loro accesso sequenziale
- Nel 1956 l' IBM vende il primo calcolatore con hard disk (RAMAC 305)
  - Capienza 5 Mbyte (50 piatti magnetici di 1 metro di diametro)
  - Costo del disco 50000 USD (10000 USD al Mbyte)



Costo al Mbyte degli hard disk



Organizzazione fisica dei dischi



- cilindri
- tracce
- settori

Ogni informazione e' individuata da tre coordinate (c, t, s)

Organizzazione logica dei dischi

- A causa della lentezza dei tempi di accesso sarebbe impensabile effettuare un accesso al disco e leggere un solo byte
- I dischi vengono indirizzati come vettori monodimensionali di blocchi logici, dove il *blocco logico* rappresenta la minima unità di trasferimento (dim. tipica 512 byte).
- L'array di blocchi logici viene mappato sequenzialmente nei settori del disco:
  - Il blocco 0 è il primo settore della prima traccia del cilindro più esterno.
  - La corrispondenza prosegue ordinatamente lungo la prima traccia, quindi lungo le rimanenti tracce del primo cilindro, e così via, dall'esterno verso l'interno.

Scheduling del disco

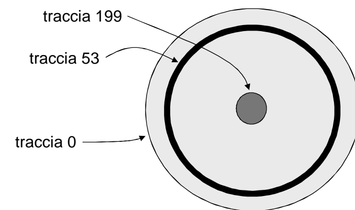
- Il SO è responsabile dell'uso efficiente dell'hardware. Per i dischi ciò significa garantire tempi di accesso contenuti e ampiezze di banda maggiori.
- Una richiesta di accesso al disco può venire soddisfatta immediatamente se unità a disco e controller sono disponibili; altrimenti la richiesta deve essere aggiunta alla coda delle richieste inevase per quell'unità.
- Il SO ha l'opportunità di scegliere quale delle richieste inevase servire per prima: uso di un algoritmo di scheduling

Ottimizzazione del seek time

- Gli algoritmi di scheduling del disco verranno testati sulla coda di richieste per le tracce (0–199):

  98, 183, 37, 122, 14, 124, 65, 67

  La testina dell'unità a disco è inizialmente posizionata sulla traccia 53.

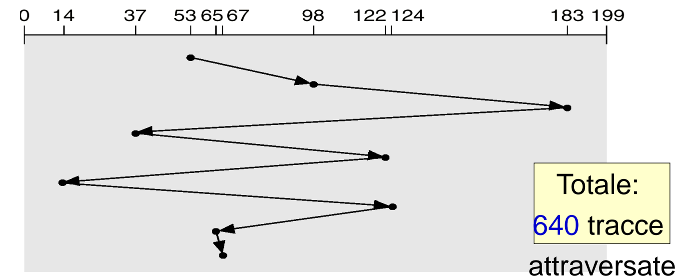


Marco Lapegna –
Laboratorio di Programmazione 2
memoria secondaria e f.s.

Scheduling First Come First Served (FCFS)

§ Soddisfa le richieste secondo l'ordine di arrivo

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



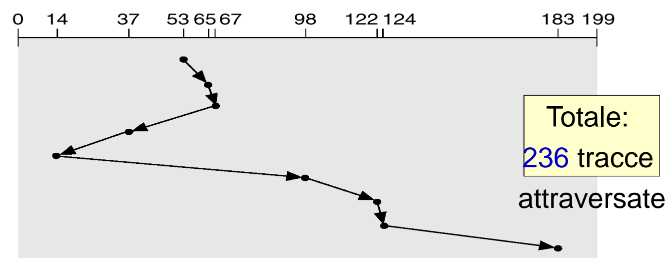
Totale:
640 tracce
attraversate

Marco Lapegna –
Laboratorio di Programmazione 2
memoria secondaria e f.s.

Scheduling Shortest Seek Time First (SSTF)

- Seleziona la richiesta con il minor tempo di seek a partire dalla posizione corrente della testina.
- Ordine di richieste molto localizzato
- Rischio di attesa indefinita delle richieste sulle tracce esterne e interne

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



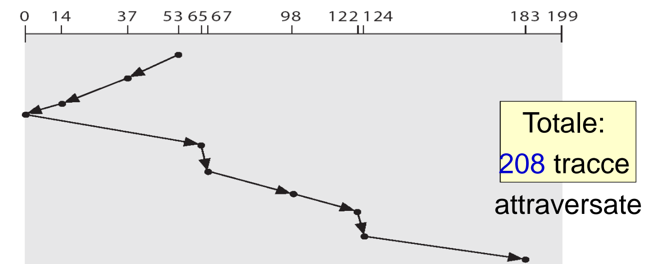
Totale:
236 tracce
attraversate

Marco Lapegna –
Laboratorio di Programmazione 2
memoria secondaria e f.s.

Scheduling SCAN

- Il braccio della testina si muove da un estremo all'altro del disco, servendo sequenzialmente le richieste;
- giunto ad un estremo inverte la direzione di marcia e, conseguentemente, l'ordine di servizio. È chiamato anche algoritmo dell'ascensore.

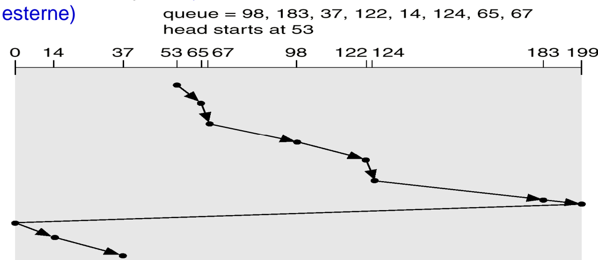
queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Totale:
208 tracce
attraversate

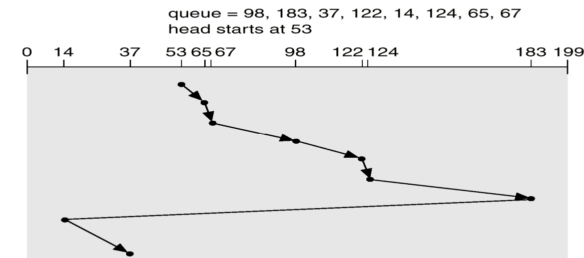
Scheduling Circular SCAN (CSCAN)

- Garantisce un tempo di attesa più uniforme rispetto a SCAN.
- La testina si muove da un estremo all'altro del disco servendo sequenzialmente le richieste. Quando raggiunge l'ultima traccia ritorna immediatamente all'inizio del disco, senza servire richieste durante il viaggio di ritorno.
- Considera le tracce come una lista circolare, con l'ultima traccia adiacente alla prima (minore discriminazione delle tracce interne e esterne)



Scheduling Circular LOOK (CLOOK)

- Versione ottimizzata (e normalmente implementata) di C–SCAN.
- Il braccio serve l'ultima richiesta in una direzione e poi inverte la direzione senza arrivare al termine del disco.



Scelta di un algoritmo di scheduling

- C-LOOK e C-SCAN hanno una varianza del tempo di risposta inferiore alle rispettive versioni non circolari
- Alcuni studi hanno comunque mostrato che la migliore politica di scheduling dovrebbe essere basata su due livelli
  - Versione circolare se il carico del disco e' alto
  - Versione non circolare se il carico del disco e' basso

Scelta di un algoritmo di scheduling

- SSTF è comune ed ha un comportamento "*naturale*".
- LOOK e C–LOOK hanno migliori prestazioni per sistemi con un grosso carico di lavoro per il disco (minor probabilità di blocco indefinito).
- Le prestazioni dipendono dal numero e dal tipo di richieste.
- Le richieste di servizio al disco possono essere influenzate dal metodo di allocazione dei file.
- L'algoritmo di scheduling del disco dovrebbe rappresentare un modulo separato del SO, che può essere rimpiazzato da un algoritmo diverso qualora mutassero le caratteristiche del sistema di calcolo.
- Sia SSTF che LOOK sono scelte ragionevoli per un algoritmo di tipo generale.

Altri compiti:

- *Spazio di swap*: la memoria virtuale impiega lo spazio su disco come un'estensione della memoria centrale.
- Lo spazio di swap può essere ricavato all'interno del normale file system o, più comunemente, si può trovare in una partizione separata del disco.
- Deframmentazione
  - Sistema dati correlati tra loro in settori contigui
  - Diminuisce il numero di ricerche necessarie sul disco
  - Partizionameneto puo' contribuire a ridurre la frammentazione
- Compressione
  - I dati consumano meno spazio sul disco
  - migliora il trasferimento e il tempo di accesso
  - Aumenta l'overhead necessario per la compressione / decompressione

Strutture RAID

La velocita' di trasferimento dei dischi non cresce allo stesso modo della velocita' doperativa della CPU

Peggioramento del througput

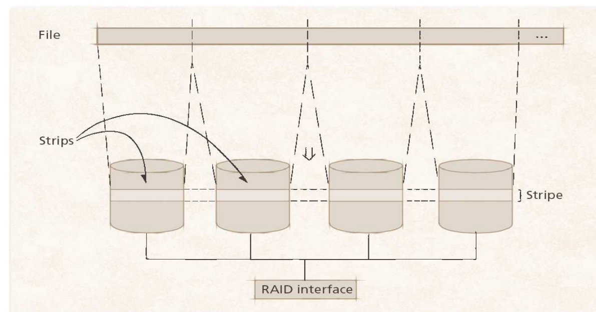
IDEA

Utilizzare piu' dischi da utilizzare "in parallelo"

RAID, *Redundant Array of Independent Disks*

Caratteristiche dei RAID

- I dischi sono visti come un'unica unita' di memorizzazione
- I file sono divisi in strisce (strips) distribuite sui vari dischi
- I pezzi della striscia relativi allo stesso file hanno la stessa locazione in ogni disco



Caratteristiche dei RAID

- La gestione dei RAID e' un'operazione dispendiosa che puo' essere causa di rallentamento della CPU e del S.O.

Hardware specializzato dedicato allo scopo (RAID controller)

- Un maggior numero di dischi aumenta la probabilita' di un guasto con conseguente perdita di dati

Ridondanza delle informazioni memorizzate

Quindi

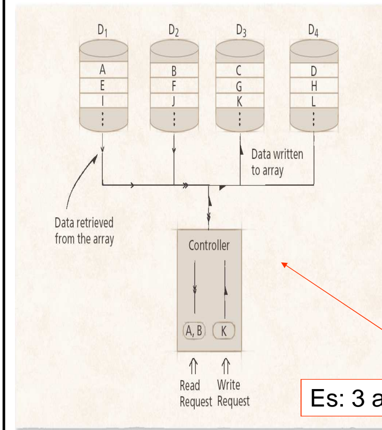
Motivazioni dei RAID

Efficienza (maggiore velocita' di accesso ai dischi)

Affidabilita' (maggiore sicurezza di non perdere dati)

Esistono molteplici schemi di gestione dei RAID chiamati Livelli RAID

Livello 0 (striping)

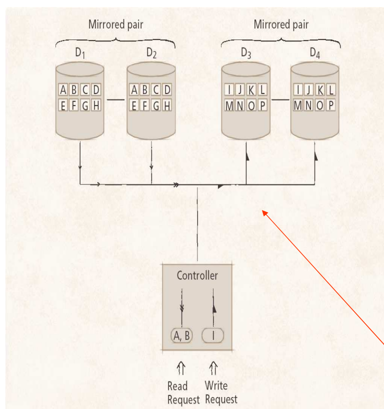


- Piu' semplice schema RAID
- striping senza ridondanza
- elevata efficienza
- rischio di perdere i dati

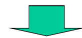
indicato quando le esigenze di performance superano quelle dell'affidabilita'

Es: 3 accessi contemporanei

Livello 1 (mirroring)



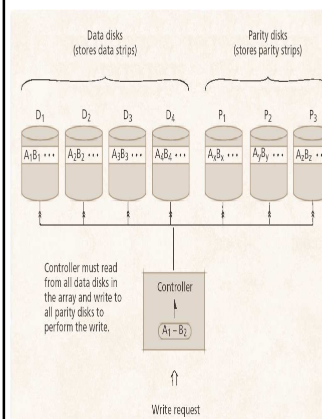
- Ogni disco ha un suo disco di copia
- ridondanza senza striping
- Buona performance in lettura, meno in scrittura
- Alto overhead di storage



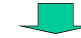
Adatto quando le esigenze di affidabilita' sono importanti

Es: tutti i dati sono duplicati

Livello 2 (Hamming error correcting code)

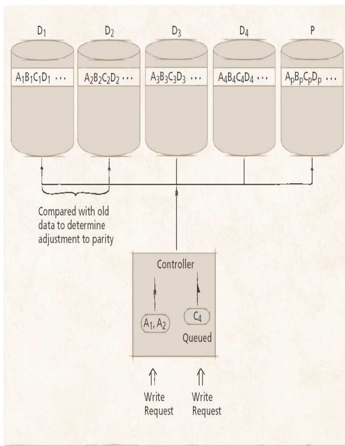


- Ridondanza e striping
- Dischi per i dati e dischi per "bit di parita' " per correggere errori
- Overhead di storage (ma meno del livello 1) e performance

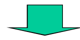


Poco utilizzato

Livello 3 (XOR error correcting code)

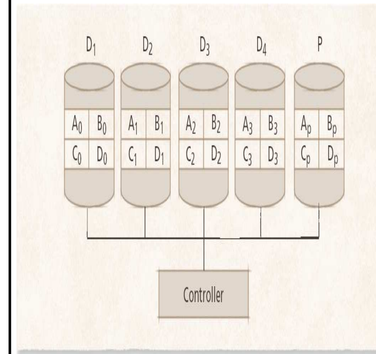


- Ridondanza e striping
- simile al livello 2
- correzione di errori con diverso meccanismo
- 1 solo disco per i bit di parita'
- scarsa efficienza (bottleneck del disco di parita')



Poco utilizzato

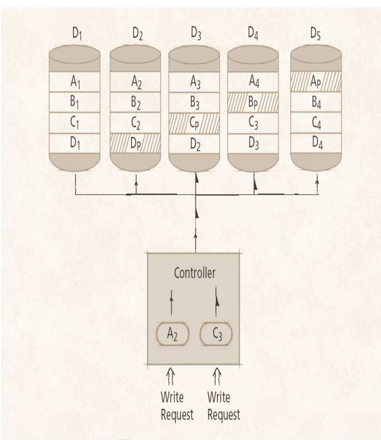
Livello 4 (block XOR eec)



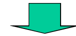
- Ridondanza e striping
- versione a blocchi del livello 3

Poco utilizzato

Livello 5 (block distributed XOR eec)



- I blocchi di parita' sono distribuiti sui dischi dei dati
- Rimozione del bottleneck
- Piu' veloce dei livelli 2, 3 e 4
- Difficile da implementare



Miglior compromesso tra efficienza e affidabilita'

Soluzione piu' utilizzata per i sistemi general purpose

Livelli RAID

- *Ce ne sarebbero tanti altri…*
  - RAID level 6 utilizza altre informazioni per migliorare la fault tolerance
  - Varie combinazioni
    - Livelli 0+1, 1+0, 0+3, 0+5, 1+5, …
- Ognuno con un differente compromesso efficienza / affidabilita'

esempio

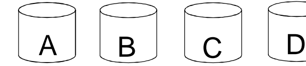
Si supponga di avere D=4 dischi da 4 GB tali che:

- Mean Time to Failure (MTF) = 1000 giorni
- Mean Time to Repair (MTR) = 10 giorni

Si calcoli la probabilita' di perdere i dati nei casi di

- sistema RAID liv. 0
- sistema RAID liv. 1
- sistema RAID liv. 5

Soluzione RAID liv. 0 (striping)



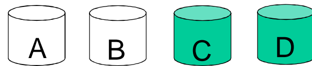
Mem.util. = 16 GB

Dischi solo per i dati
Perdita di dati se
guasto un disco qualunque

Prob. che un disco (ad es. A) si guasti = 0.001
(in generale 1/MTF)

Prob. che si guasti uno dei 4 dischi = 4*0.001 = 0.004
(in generale D/MTF)

Soluzione RAID liv. 1 (mirroring)



Mem.util. = 8 GB

C copia di A
D copia di B
Perdita di dati se
guasti A e C oppure B e D

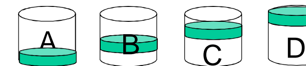
Prob. che un disco (ad es. A) si guasti = 1/MTF = 0.001

Prob. che si guasti C in un arco di 10 giorni = 10*0.001 = 0.01
(in generale MTR/MTF)

Prob. che si guastino A e C insieme nell'arco di 10 giorni = 0.00001
(in generale MTR/MTF^2)

Prob. che si guastino nell'arco di 10 giorni
A e C oppure B e D = 2*0.00001 = 0.00002
(in generale D*MTR/(2*MTF^2))

Soluzione RAID liv. 5 (Distributed EEC)



Mem.util. = 12 GB

Blocchi distribuiti per la correzione di errori
Perdita di dati se
guasti due dischi qualunque

Prob. che un fissato disco (ad es. A) si guasti = 1/MTF = 0.001

Prob. che si guastino

- A e B nell'arco di 10 giorni = 0.00001
- A e C nell'arco di 10 giorni = 0.00001
- A e D nell'arco di 10 giorni = 0.00001

} 3*0.00001 = 0.00003

(in generale (D-1)*MTR*MTF^2)

Prob. che si guastino nell'arco di 10 giorni

- B e C oppure B e D oppure C e D 0.00003

una qualunque coppia di dischi = 0.00006
(in generale D*(D-1)*MTR/(2*MTF^2))

quindi

Con 4 dischi da 4 GB tali che:
- Mean Time to Failure (MTF) = 1000 giorni
- Mean Time to Repair (MTR) = 10 giorni

	Memoria utilizzabile	Probabilita' di perdere dati
liv. 0	16 GB	0.004
liv. 1	8 GB	0.00002
liv. 5	12 GB	0.00006

Dispositivi di memorizzazione terziaria

- I compiti principali del SO sono la gestione dei dispositivi fisici e la presentazione di una macchina virtuale astratta alle applicazioni utente.
- La caratteristica fondamentale della memorizzazione terziaria è il basso costo.
- Generalmente, viene effettuata su mezzi rimovibili, ad esempio floppy disk, CD–ROM, nastri magnetici.
- La maggior parte dei SO gestisce i dischi rimovibili come i dischi fissi: un nuovo supporto viene formattato, e un file system vuoto viene generato sul disco.
- I nastri sono presentati come un mezzo di memorizzazione a basso livello, e le applicazioni non aprono un file, ma l'intero nastro.
- In genere l'unità a nastro è riservata per un'applicazione alla volta.

File

- Il maggior problema nella gestione della memoria secondaria e' la necessita' di fare riferimenti alla organizzazione fisica dei dispositivi.
- Esempio:
  - un programma in linguaggio macchina puo' risiedere nei blocchi 45, 51, 68, 73 e 99 di un disco
  - Una modifica al codice puo' produrre una diversa memorizzazione
  - Necessita' di fare riferimento ai blocchi fisici anche se l'entita' logica (il programma) e' sempre la stesso

FILE

Una **raccolta di dati associata ad un nome**, residente in memoria secondaria e manipolabile come una unica entita' logica

Tabella dei file aperti

I programmi possono fare numerosi riferimenti ad uno stesso file

Problema: numerosi accessi al file
e quindi numerose ricerche del file

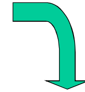
Il sistema operativo definisce una tabella dei file aperti

open ➔ crea una entry nella tabella dei file aperti
read/write ➔ il file viene individuato in base all'indice della tabella
close ➔ viene rimossa la entry nella tabella dei file aperti

Esempio: Unix

- Esempio di indice ottenuto con la s.c. open

```
#include<stdio.h>
main( ){
int fd;
fd=open("pippo", 0644);
printf("file descriptor = %d\n", fd);
close(fd);
}
```



```
prompt  ->  ./a.out
file descriptor = 3
prompt ->
```


Struttura interna dei file

- Un file e' costituito da sequenze di dati piu' o meno strutturati (byte, sequenze o record)
- Tali dati (record logici) devono essere impacchettati nei blocchi fisici del disco che hanno dimensioni fissate
- Esempio:
  - Un file di 1949 byte richiede 4 blocchi di 512 byte per un totale di 2048 byte
  - I 99 byte dell'ultimo blocco non utilizzati costituiscono la frammentazione interna del disco

Maggiore e' la dimensione del blocco
maggiore e' la frammentazione !!

Metodi di accesso

Indipendentemente dalla sua memorizzazione un file puo' avere differenti metodi di accesso

- Accesso sequenziale: i dati si elaborano ordinatamente un record dopo l'altro. Si incrementa il file pointer dopo ogni lettura o scrittura. Sono possibili operazioni esplicite di spostamento avanti o indietro
- Accesso diretto: si puo' accedere ai dati in un ordine qualunque specificando la posizione del dato.
- Accesso indicizzato: si accede al file mediante una tabella con gli indici dei blocchi. Per reperire un elemento del file occorre prima cercare nell'indice l'elemento corrispondente e utilizzare il puntatore in esso contenuto per accedere ai dati. Se la tabella e' grande si usa un indice a piu' livelli (es. Indexed Sequential Access Method dell'IBM)

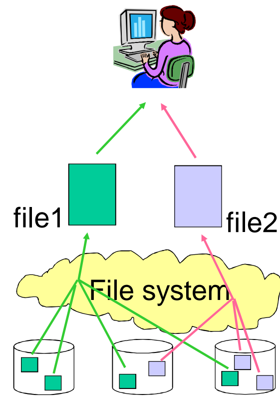
Il file system

- Un sistema operativo puo' gestire
  - milioni di file,
  - di differenti utenti
  - di tipi differente

Necessita' di organizzare logicamente i file

L'organizzazione logica dei file di un sistema operativo e' chiamata FILE SYSTEM

Il file system:



- Organizza i file e gestisce l'accesso ai dati
- Responsabile della integrita' dei file, della gestione dei metodi di accesso (lettura/scrittura) e della gestione della memoria secondaria

Visione dei file indipendente dai dispositivi fisici !!!

Directory

- Lo strumento piu' comune usato per realizzare un file system e' la directory *(elenco o cartella)*
- Una directory e' un file che contiene nomi e locazioni di file presenti nel file system (non possono contenere dati utente)
- Un elemento di una directory conserva informazioni come:
  - Nome file
  - Posizione
  - grandezza
  - Tipo
  - Privilegi di accesso
  - Tempo di creazione e modifica

esempio: Linux

- Il comando LINUX ls elenca le caratteristiche degli elementi di una directory

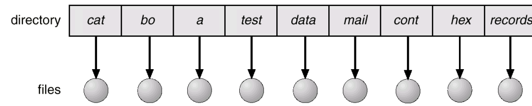
```
[lapegna@renato lapegna]$ ls -la
totale 27492
drwx------    30 lapegna  users        4096 gen 11 11:04 .
drwxr-xr-x    20 root     root         4096 dic 12 13:16 ..
-rw-r--r--     1 lapegna  users           0 set 26 10:20 aaa
drwxr-xr-x     2 lapegna  users        4096 set 26 12:44 anastasia
-rwxr-xr-x     1 lapegna  users       13961 gen 11 11:02 a.out
-rw-------     1 lapegna  users       14403 nov 29 17:15 .bash_history
-rw-r--r--     1 lapegna  users          24 lug  9  2001 .bash_logout
-rw-r--r--     1 lapegna  users         390 mar 30  2005 .bash_profile
-rw-r--r--     1 lapegna  users         138 gen 17  2005 .bashrc
```


Operazioni sulle directory

Poiche' una directory e' un file, sui suoi elementi e' possibile fare le stesse operazioni che e' possibile fare sugli elementi di un file

- Ricerca di un file
- Creazione di un file
- Cancellazione di un file
- Elenco dei contenuti di una directory
- Ridenominazione di un file
- Attraversamento del file system

File system monolivello (file system piatto)



- gli elementi di una directory sono solamente file
- Una directory unica per tutti i file.
- Problemi di nominazione: occorre scegliere un nome diverso per ogni file.
- Problemi di performance: ricerca lineare nella directory
- Nessun raggruppamento logico.

- raramente implementato

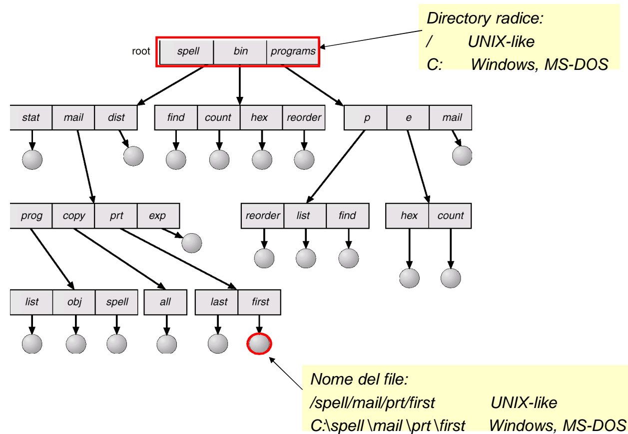
File system gerarchico (ad albero)

- Gli elementi di una directory possono essere file o directory
- Una directory principale (root directory, master file directory) indica la radice dell'albero dalla quale partono le directory di secondo livello
- L'unicita' dei nomi di file e' realizzata mediante il "pathname assoluto" costituito dai nomi delle directory incontrate dalla radice fino al file

- I nomi dei file possono essere unici solo all'interno della stessa directory (pathname relativo) ➔ nominazione
- Ricerca solo nella directory corrente ➔ efficienza
- Possibilita' di raggruppamenti logici ➔ grouping

- Implementato nella quasi totalita' dei s.o.

File system gerarchico (ad albero)



Directory radice:
/ UNIX-like
C: Windows, MS-DOS

Nome del file:
/spell/mail/prt/first UNIX-like
C:\spell\mail\prt\first Windows, MS-DOS

Protezione: esempio Unix

- Esempio Unix: Tre classi di utenti e tre tipi di accesso definiti mediante bit
  - 1 si
  - 0 no

	RWX	
Proprietario	1 1 1	➔761
Gruppo	1 1 0	
Altri	0 0 1	

- Al momento dell'apertura del file il s.o. ritorna un identificativo o un errore a secondo se sono soddisfatti i requisiti di sicurezza o meno

- Solaris 2.6 usa uno schema ibrido

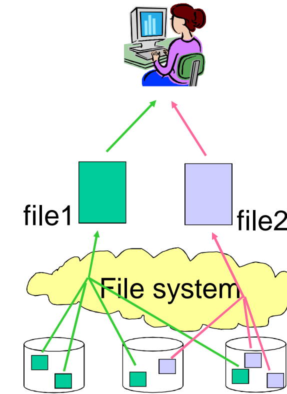
Condivisione di file

- Un problema comune nei file system in ambiente multiutente e' la condivisione dei file
  - Condivisione all'interno di uno stesso file system
  - Condivisione in un Network File System (file system di rete)
- Stabilito chi e come puo' accedere ai file sono necessarie delle regole di sincronizzazione alle risorse (semantica della consistenza)
- Esempio tipo: problema lettori/scrittori

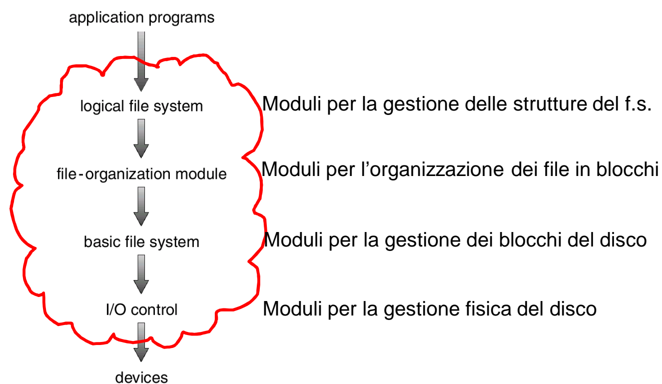
Problema

Un problema fondamentale nella realizzazione di un f.s. e' la definizione di algoritmi e strutture dati per far corrispondere il file system logico (file, directory,…) al file system fisico (dispositivi fisici della memoria secondaria)

Organizzazione a livelli



Struttura del file system



Moduli per la gestione delle strutture del f.s.

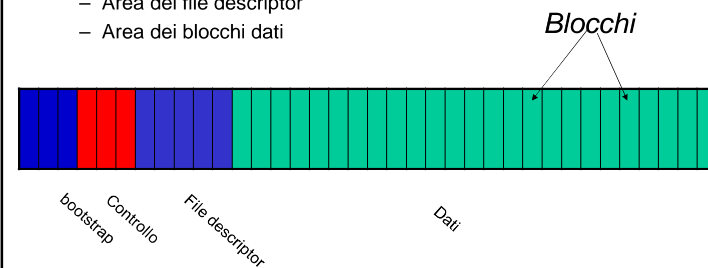
Moduli per l'organizzazione dei file in blocchi

Moduli per la gestione dei blocchi del disco

Moduli per la gestione fisica del disco

Organizzazione di una partizione

- Dipende dal s.o. e dal file system
- In generale una partizione e' divisa in 4 aree distinte
  - Area di bootstrap
  - Area di controllo
  - Area dei file descriptor
  - Area dei blocchi dati



Oranizzazione di una partizione

- Area di bootstrap
  - Contiene il programma per l'inizializzazione del sistema
  - Di solito e' il primo blocco di una partizione
  - Per uniformita' ce n'e' una in ogni partizione
  - Diversi nomi
    - Boot block (Unix file system)
    - Partition boot sector (NTFS)
- Area dei file descriptor
  - Contiene le strutture dati e gli indirizzi dei blocchi dei dati dei file
  - Diversi nomi
    - i-list (UFS)

Organizzazione di una partizione

- Area di controllo
  - Contiene informazioni su tutta la partizione
    - Dimensione della partizione
    - Numero e posizione dei blocchi liberi
    - Dimensione della i-list (area dei file descriptor)
  - Diversi nomi
    - Superblocco (UFS)
    - Master file table (NTFS)
- Area dei dati
  - Contiene i blocchi dei dati dei file e i blocchi liberi

Area dei blocchi dei dati

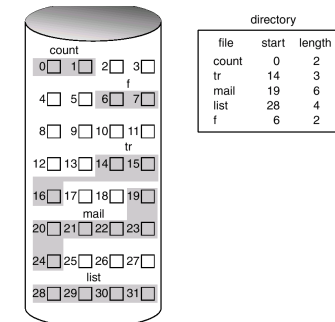
- Problema: come allocare lo spazio disco ai file in modo da avere spreco minimo di memoria e rapidità di accesso?
- Il metodo di allocazione dello spazio su disco descrive come i blocchi fisici del disco vengono allocati ai file:
  - Allocazione contigua
  - Allocazione concatenata
  - Allocazione indicizzata

Allocazione contigua

- 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).

Principali problemi

- i file non possono crescere
- frammentazione

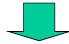


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

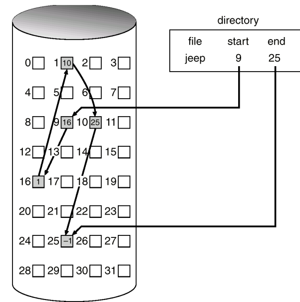
Allocazione concatenata

- Ciascun file è una lista concatenata di blocchi su disco:
- i blocchi possono essere sparsi ovunque nel disco.
- La directory contiene l'indirizzo del blocco iniziale.
- Ogni blocco possiede un'area puntatore



Principali problemi

- per trovare un blocco e' necessario scorrere tutta la lista sul disco con numerosi posizionamenti della testina
- Perdita di efficienza

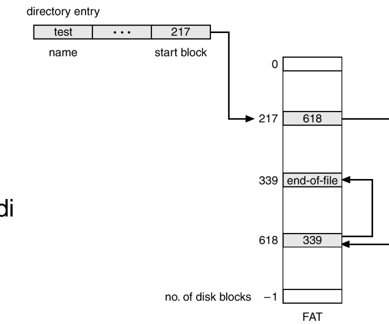


Allocazione tabellare

- Rappresenta una possibile soluzione al problema dello scorrimento della lista sul disco
- Mantenere una tabella contenuta in un unico blocco che contiene la lista dei puntatori ai blocchi del disco
- Per trovare un blocco si scorre la lista nella tabella e non sul disco

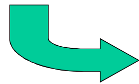
File Allocation Table
FAT

Implementata nei f.s. di
- MS-DOS
- OS/2



FAT (Microsoft)

- Prima versione della FAT dell'MS-DOS 1.0 (FAT12) usava 12 bit per l'indirizzamento dei blocchi
- 2^{12} = 4096 blocchi



Dim. del disco	Dim. dei blocchi
4 MB	1K
16 MB	4K
64 MB	16K

- Successive versioni a 16 e 32 bit

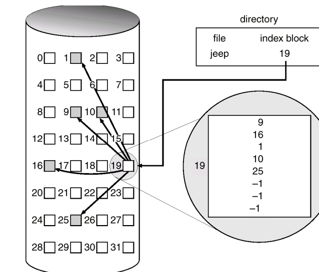
Blocchi grandi
=
frammentazione

Allocazione indicizzata

- Colleziona tutti i puntatori in un unico *blocco indice*.
- La directory contiene l'indirizzo del blocco indice
- Richiede una tabella indice.
- Permette l'accesso dinamico senza frammentazione esterna;

Principali problemi

- overhead per l'accesso al blocco indice (comunque inferiore all'allocazione concatenata)



Esempio

- Area dati = 64 Mbyte
- Blocchi di 1 Kbyte

→ 65536 blocchi

↓

Necessari 16 bit (2 byte) per indirizzarli

512 entry di 16 bit nel blocco indice

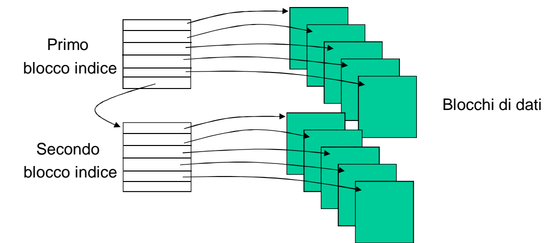
↓

Massima dimensione di un file

512 * 1 Kbyte = 512 Kbyte

File di grandi dimensioni

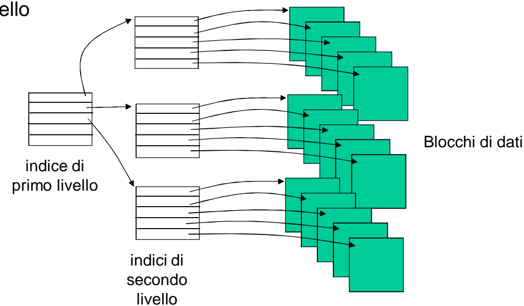
- Nell'esempio precedente se il file ha dimensione maggiore di 256Kbyte e' possibile concatenare i blocchi indice



Allocazione indicizzata – schema concatenata

Una alternativa

- Il blocco indice punta ad altri blocchi indice di secondo (e terzo) livello



Allocazione indicizzata – schema multilivello

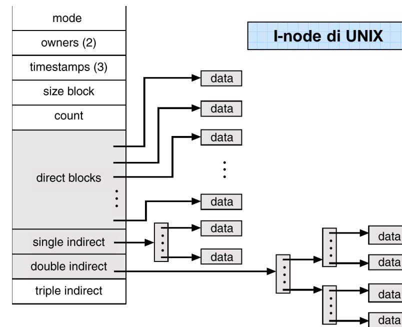
(e multiaccesso !!)

Esempio:

- Blocchi di 1 Kbyte e puntatori di 32 bit (4byte)
- Indirizzamento a 1 livello
  - 1024/4 = 256 blocchi indirizzabili
  - Dimensione massima di un file = 256 Kbyte
- Indirizzamento a 2 livelli
  - 256*256 = 65536 blocchi indirizzabili
  - Dimensione massima di un file = 64 Mbyte
- Indirizzamento a 3 livelli
  - 256*256*256 = ~ 16 milioni di blocchi indirizzabili
  - Dimensione massima di un file = 16 Gbyte (! ! !)

Schema combinato: Unix

- Nel Unix File System, l'area dei descrittori (chiamata I-list: index list) contiene una tabella di descrittori chiamati i-node (index node)
- Ogni i-node e' accessibile attraverso l'indice della tabella (i-number)
- Un i-node contiene gli attributi di un file



- Indirizzi dei blocchi
  - 12 diretti
  - 1 indiretto singolo
  - 1 indiretto doppio
  - 1 indiretto triplo

Indirizzamento

- Blocchi di 1 kbyte
- Indirizzi di 4 byte
- 1024/4 = 256 indirizzi per blocco

- Dimensione massima per un file
  - 12 * 1 Kbyte = 12 Kbyte +
  - 256 * 1 Kbyte = 256 Kbyte +
  - 256^2 * 1 Kbyte = 64 Mbyte +
  - 256^3 * 1 Kbyte = 16 Gbyte =
  - Totale > 16 Gbyte

- Indirizzamenti a 64 bit

Superiore alla capacita' di indirizzamento di 32 bit

osservazione

- Qual'e' l'idea alla basedi tale organizzazione?

- Il numero di accessi al disco e' proporzionale al livello di indirizzamento

- File piccoli (< 12 Kbyte) 2 accessi
- File medi (< 256 Kbyte) 3 accessi
- File grandi (< 64 Mbyte) 4 accessi
- File molto grandi (< 16 Gbyte) 5 accessi

Gestione dello spazio libero

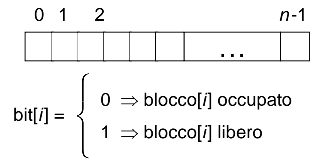
- Poiche' i file sono entita' estremamente dinamiche (aumentano e diminuiscono in numero e in dimensione) e' necessario assicurarsi una gestione efficiente dello spazio libero

- L'area di controllo (superblocco) di una partizione contiene le informazioni sulla posizione dei blocchi liberi

- Tre possibili gestioni:
  - Bitmap
  - Lista dello spazio libero
  - raggruppamento

Bitmap

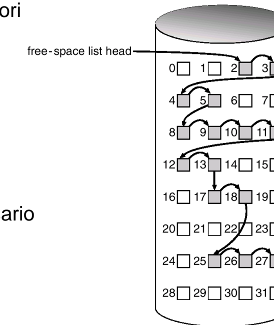
- Vettore di bit (*n* blocchi)



- Vantaggi: Buone prestazioni se il vettore è conservato in memoria centrale.
- Svantaggi: per trovare un blocco libero puo' essere necessario esaminare gran parte della bitmap

Lista concatenata

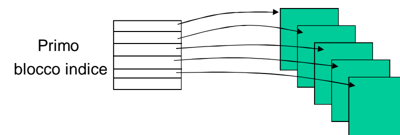
- Lista di blocchi liberi legati da puntatori
- Puntatori alla testa e alla coda sono memorizzati nel superblocco



- Vantaggio: 1 accesso per trovare un blocco libero
- Svantaggio: inefficiente se e' necessario trovare molti blocchi

Raggruppamento

- Raggruppamento: memorizzazione degli indirizzi di *n* blocchi liberi sul primo di tali blocchi. Sull'ultimo sono indirizzati altri blocchi, etc.



- Vantaggio: rapida ricerca di blocchi liberi
- Svantaggio: gestione inefficiente se il numero di blocchi e' grande

Descrittori di file

Alla richiesta di aprire un file esistente o di creare un nuovo file il kernel ritorna un descrittore di file al processo chiamante (un intero non negativo)

Per convenzione
Il descrittore 0 viene associato allo **standard input**,
Il descrittore 1 viene associato allo **standard output**
Il descrittore 2 viene associato allo **standard error**

I numeri 0, 1 e 2 possono essere sostituiti con le costanti
STDIN_FILENO **STDOUT_FILENO** **STDERR_FILENO**
definite nell'header <unistd.h>

Chiamata di sistema open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path,  int oflag,   mode_t mode );
```

Rit.: descrittore se ok, -1 se errore

Funzione per aprire o creare file

path e' il nome del file da aprire

Chiamata di sistema open

oflag può assumere diversi valori (definiti in <fcntl.h>)

O_RDONLY apri solo in lettura

O_WRONLY apri solo in scrittura

O_RDWR apri in lettura e scrittura

Solo una delle precedenti **costanti** può essere specificata

Chiamata di sistema open

Le precedenti costanti possono pero' essere combinate, mediante **OR** con le costanti

O_APPEND esegue un *append* alla fine del file per ciascuna write

O_CREAT crea il file se non esiste

O_EXCL se utilizzato insieme a O_CREAT, ritorna un errore se il file esiste

O_TRUNC se il file esiste, lo tronca a lunghezza zero

…

Chiamata di sistema creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char *path, mode_t mode);
```

Rit.: descrittore se ok, -1 se errore

Funzione per creare un file. Il file e' aperto in sola scrittura

Essa è equivalente a:

open(path, O_WRONLY | O_CREAT | O_TRUNC, mode);

Chiamata di sistema close

```
#include <unistd.h>
int close(int fildes);
                         Rit.: 0 se ok, -1 se errore
```

La funzione close chiude un file e rende inutilizzabile il descrittore di file ad esso associato

fildes e’ il descrittore di file ritornato da open

Quando un **processo termina**, tutti i **file** aperti vengono automaticamente **chiusi**

Chiamata di sistema read

```
#include <unistd.h>
ssize_t read(int fildes, void *buff, size_t nbytes);
                    Rit.: numero bytes letti se ok, -1 se errore
```

Effettua una lettura in un file

fildes e’ il descrittore del file

buff e’ il buffer in cui leggere

nbytes e’ il numero di byte da leggere a partire dalla posizione corrente

Aggiorna la posizione corrente

Chiamata di sistema write

```
#include <unistd.h>
ssize_t write(int fildes, const void *buff, size_t nbytes);
                         Rit.: num bytes scritti se ok, -1 se errore
```

Effettua una scrittura in un file

fildes e’ il descrittore del file

buff e’ il buffer da scrivere

nbytes e’ il numero di byte da scrivere a partire dalla posizione corrente

Aggiorna la posizione corrente

Esempio: scrittura su file

File scrivi.c

```
#include        <sys/types.h>
#include        <sys/stat.h>
#include        <fcntl.h>

int main(void) {
          int   fd;
          char      buf1[ 10] = "abcdefghij";
          char      buf2[ 10] = "ABCDEFGHIJ";

          fd = open(“pippo", O_CREAT | O_WRONLY);

          write(fd, buf1, 10) ;

          close(fd);

          exit(0);
}
```

Nel file ci sono le minusole

Esempio lettura da file

File leggi.c

```
#include        <sys/types.h>
#include        <sys/stat.h>
#include        <fcntl.h>

int main(void) {
        int   fd;
        char     buf1[ 10] = "abcdefghij";
        char     buf2[ 10] = "ABCDEFGHIJ";

        fd =open(“pippo", O_RDONLY);

        read(fd, buf2, 10) ;

        write(STDOUT_FILENO, buf2, 10);

        close(fd);

        exit(0);
}
```

Sovrascrivo buf2

Esempio (cont.)

```
[lapegna%] cc scrivi.c

[lapegna%] a.out

[lapegna%] cat pippo

abcdefghij [lapegna%] cc leggi.c

[lapegna%] a.out

abcdefghij [lapegna%]
```


funzioni chmod e fchmod

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod (const char *path, mode_t mode);
int fchmod (int fildes, mode_t mode);

                                    rit.: 0 se ok, -1 se errore
```

chmod cambia i permessi di accesso al file *path*

fchmod cambia i permessi di accesso al file con descrittore *fildes*

funzioni chmod e fchmod

mode e’ una combinazione di

S_IRUSR	00400	owner read
S_IWUSR	00200	owner write
S_IXUSR	00100	owner execute
S_IRGRP	00040	group read
S_IWGRP	00020	group write
S_IXGRP	00010	group execute
S_IROTH	00004	others read
S_IWOTH	00002	others write
S_IXOTH	00001	others execute

esempio

```
#include        <sys/types.h>
#include        <sys/stat.h>

int main(void)
{
  struct stat     statbuf;

/* definisci permessi di accesso a pippo come  "rw-r--r--" */
  chmod("pippo", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) ;


  /* elimina il permesso di esecuzione al gruppo per il file pluto */
   stat("pluto", &statbuf) ;
   chmod("pluto",  statbuf.st_mode & ~S_IXGRP) ;

exit(0);
}
```


funzioni chown e fchown

```
#include <sys/types.h>
#include <unistd.h>

int chown (const char *path, uid_t owner, gid_t group);
int fchown (int fd, uid_t owner, gid_t group);
                                        rit.: 0 se ok, -1 se errore
```

chown cambia userID e groupID del file *path*

fchmod cambia userID e groupID del file al file con descrittore *fildes*

Esercizio 1

Un floppy disk ha 40 tracce. Un'operazione di ricerca richiede 6msec per lo spostamento tra una traccia e l'altra, la latenza rotazionale media è di 10msec ed il tempo di trasferimento è di 25msec per blocco.

a) Quanto tempo è necessario per leggere un file costituito da 20 blocchi e memorizzato sul dischetto, se i blocchi logicamente contigui del file distano mediamente 13 tracce l'uno dall'altro?

b) Quanto tempo è necessario per leggere un file con 100 blocchi mediamente distanti 2 tracce?

soluzione

Indipendentemente dal numero delle tracce del disco..

a) Il tempo medio per accedere e quindi trasferire in memoria principale un blocco del file è:

[(6*msec* x 13) + 10*msec* + 25*msec*],

corrispondente alla somma del tempo medio di ricerca, tempo medio di latenza e tempo di trasferimento effettivo. Conseguentemente, il tempo totale necessario a trasferire tutto il file è

20 x [(6*msec* x 13) + 10*msec* + 25*msec*] = 2260*msec* ~ 2.3sec

b) Il tempo totale per il trasferimento del file è

100 x [(6*msec* x 2) + 10*msec* + 25*msec*] = 4700*msec* = 4.7sec

Esercizio 2

Al driver di un disco arrivano, nell'ordine, richieste per le tracce

10, 22, 20, 2, 40, 6 e 38.

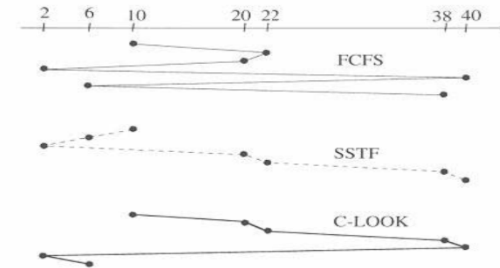
Uno spostamento da una traccia a quello adiacente richiede 6*msec*.

Si calcoli il tempo di ricerca necessario per servire le richieste con:

- la politica FCFS,
- la politica SSTF,
- la politica C-LOOK (ordine ascendente)

Si assuma, per tutti i casi, che il braccio si trovi inizialmente posizionato sulla traccia 10.

soluzione



t_{FCFS} = 6 x (12 + 2 + 18 + 38 + 34 + 32)msec = 816msec

t_{SSTF} = 6 x (4 + 4 + 18 + 2 + 16 + 2)msec = 276msec

$t_{C\text{-}LOOK}$ = 6 x (10 + 2 + 16 + 2 + 38 + 4)msec = 432msec

Esercizio 3

Un disco possiede 5.000 tracce , numerati da 0 a 4999. Il driver del disco sta attualmente servendo una richiesta alla traccia 153.

La coda di richieste in attesa, in ordine FIFO, è:

85, 1470, 913, 1774, 948, 130.

A partire dalla posizione corrente, qual'è la distanza totale (indicata in tracce) che deve percorrere il braccio del disco per soddisfare tutte le richieste in attesa, per ciascuno dei seguenti algoritmi di scheduling?

1. FCFS
2. SSTF
3. SCAN (inizio disc.)
4. C-SCAN (inizio disc.)
5. C-LOOK (inizio disc.)

Soluzione

*dist(*FCFS*)* =
(68 + 1385 + 557 + 861 + 826 + 818) = 4515 tracce

dist(SSTF) =
(23 + 45 + 828 + 35 + 522 + 304) = 1757 tracce

dist(SCAN) =
(23 + 45 + 85 + 913 + 35 + 522 + 304) = 1927 tracce

dist(C-SCAN) =
(23 + 45 + 85 + 5000 + 3226 + 304 + 522 + 35)= 9240 tracce

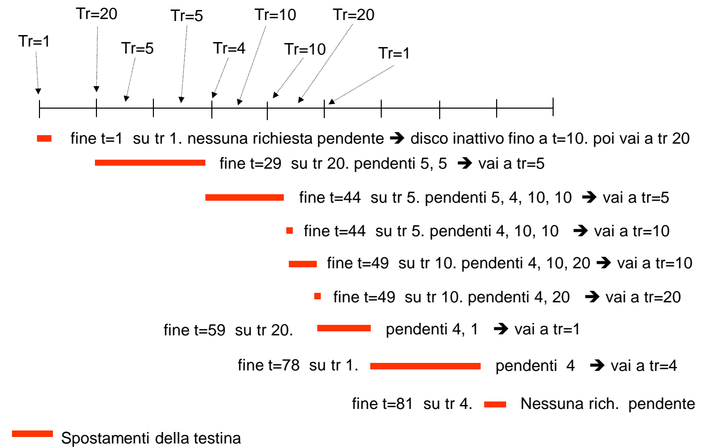
dist(C-LOOK) =
(23 + 45 + 1689 + 304 + 522 + 35) = 2618 tracce

Esercizio 4

Sia dato un disco con velocita' di seek di 1 traccia per ms e tempi di latenza rotazionale e di trasferimento trascurabili . La posizione iniziale della testina e' sulla tracia 0. Si descriva il percorso delle testine e si calcoli il tempo necessario per completare la seguente sequenza di richieste con l'algoritmo C-LOOK (in ordine ascendente).

- t=0 ms traccia 1
- t=10 ms traccia 20
- t=15 ms traccia 5
- t=25 ms traccia 5
- t=30 ms traccia 4
- t=35 ms traccia 10
- t=40 ms traccia 10
- t=45 ms traccia 20
- t=50 ms traccia 1

soluzione



Esercizio 5

- Sia dato un disco di 10 tracce numerate da 0 a 9 tale che
  - Seek time pr traccia di 1 ms
  - Lat. Rotazionale media 1 ms
  - Tempo di trasf per un blocco di dati di 1 ms
- Testina inizialmente su traccia 0 ed arrivano le seguenti richieste
  - t=0 2 richieste per traccia 1
  - t=5 2 richieste per traccia 0 e 2 richieste per traccia 2
  - t=10 2 richieste per traccia 1 e 2 richieste per traccia 3
  - ---
  - t=40 2 richieste per traccia 7 e 2 richieste per traccia 9
  - t=45 2 richieste per traccia 8

MOLTE RICHIESTE

- confrontare le politiche di scheduling LOOK e CLOOK

soluzione LOOK

- ordine di servizio per LOOK
  1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
- ogni richiesta richiede seek + 2* lat.rot. + 2* trasf = 5 ms

tr	1	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2	1	0
Ta	0	5	10	15	20	25	30	35	40	45	40	35	30	25	20	15	10	5
Tf	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Tc	5	5	5	5	5	5	5	5	5	5	15	25	35	45	55	65	75	85

Ta = tempo di arrivo, Tf = tempo di fine, Tc = tempo di completamento

soluzione CLOOK

§ ordine di servizio per CLOOK

1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8

§ ogni richiesta e' composta da seek + 2*lat.rot. + 2*trasf = 5 ms

MA tra traccia 9 e traccia 0 c'e' un overhead di 8ms

tr	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8
Ta	0	5	10	15	20	25	30	35	40	5	10	15	20	25	30	35	40	45
Tf	5	10	15	20	25	30	35	40	45	58	63	68	73	78	83	88	93	98
Tc	5	5	5	5	5	5	5	5	5	53	53	53	53	53	53	53	53	53

LOOK vs CLOOK

- l'esempio e' significativo per un "disco con molte richieste"

	LOOK	CLOOK
Tempi medi	25	29
Dev. standard	27	24

- L'overhead e' in parte distribuito su tutte le richieste
- Tempi medi un po' piu' alti per CLOOK ma piu' omogenei

esercizio LOOK vs CLOOK (parte 2)

- Sia dato un disco di 10 tracce numerate da 0 a 9 tale che
  - Seek time pr traccia di 1 ms
  - Lat. Rotazionale media 1 ms
  - Tempo di trasf per un blocco di dati di 1 ms
- Testina inizialmente su traccia 0 ed arrivano le seguenti richieste
  - t=0 1 richiesta per traccia 2
  - t=4 1 richiesta per traccia 0 e 1 richiesta per traccia 4
  - t=8 1 richiesta per traccia 2 e 1 richiesta per traccia 6
  - t=12 1 richiesta per traccia 4 e 1 richiesta per traccia 8
  - t=16 1 richiesta per traccia 6

POCHE RICHIESTE

- confrontare le politiche di scheduling LOOK e CLOOK

soluzione LOOK

- ordine di servizio per LOOK
  2, 4, 6, 8, 6, 4, 2, 0
- ogni richiesta richiede 2* seek + lat.rot. + trasf = 4 ms

tr	2	4	6	8	6	4	2	0
Ta	0	4	8	12	16	12	8	4
Tf	4	8	12	16	20	24	28	32
Tc	4	4	4	4	4	12	20	28

Ta = tempo di arrivo, Tf = tempo di fine, Tc = tempo di completamento

soluzione CLOOK

§ ordine di servizio per CLOOK
2, 4, 6, 8, 0, 2, 4, 6

§ ogni richiesta e' composta da 2* seek +lat.rot. +trasf = 4 ms
MA tra traccia 8 e traccia 0 c'e' un overhead di 6ms

tr	2	4	6	8	0	2	4	6
Ta	0	4	8	12	4	8	12	16
Tf	4	8	12	16	26	30	34	38
Tc	4	4	4	4	22	22	22	22

LOOK vs CLOOK

- l'esempio e' significativo per un "disco con poche richieste"

	LOOK	CLOOK
Tempi medi	10	13.5
Dev.standard	8.7	9.5

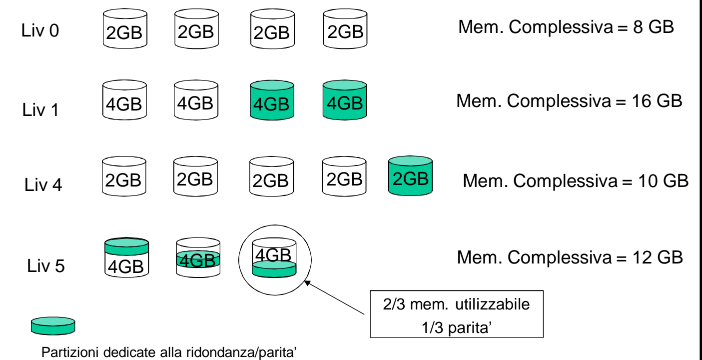
- Tempi piu' alti e meno omogenei per CLOOK

Esercizio 6

Illustrare la composizione (memoria complessiva) di un sistema RAID con 8 GB di memoria secondaria effettivamente utilizzabile con riferimento a :

a) Configurazione di livello 0 e 4 dischi
b) Configurazione di livello 1 e 4 dischi
c) Configurazione di livello 4 e 5 dischi
d) Configurazione di livello 5 e 3 dischi

soluzione



Esercizio 7

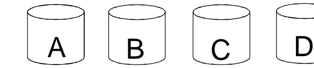
Si supponga di avere D=4 dischi da 4 GB tali che:

- Mean Time to Failure (MTF) = 1000 giorni
- Mean Time to Repair (MTR) = 10 giorni

Si calcoli la probabilita' di perdere i dati nei casi di

- sistema RAID liv. 0
- sistema RAID liv. 1
- sistema RAID liv. 5

Soluzione RAID liv. 0 (striping)



Mem.util. = 16 GB

Dischi solo per i dati

Perdita di dati se

guasto un disco qualunque

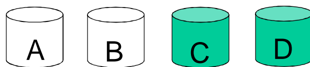
Prob. che un disco (ad es. A) si guasti = 0.001

(in generale 1/MTF)

Prob. che si guasti uno dei 4 dischi = 4*0.001 = 0.004

(in generale D/MTF)

Soluzione RAID liv. 1 (mirroring)



Mem.util. = 8 GB

C copia di A

D copia di B

Perdita di dati se

guasti A e C oppure B e D

Prob. che un disco (ad es. A) si guasti = 1/MTF = 0.001

Prob. che si guasti C in un arco di 10 giorni = 10*0.001 = 0.01

(in generale MTR/MTF)

Prob. che si guastino A e C insieme nell'arco di 10 giorni = 0.00001

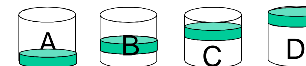
(in generale MTR/MTF^2)

Prob. che si guastino nell'arco di 10 giorni

A e C oppure B e D = 2*0.00001 = 0.00002

(in generale $D*MTR/(2*MTF^2)$)

Soluzione RAID liv. 5 (Distributed EEC)



Mem.util. = 12 GB

Blocchi distribuiti per la correzione di errori

Perdita di dati se

guasti due dischi qualunque

Prob. che un fissato disco (ad es. A) si guasti = 1/MTF = 0.001

Prob. che si guastino

- A e B nell'arco di 10 giorni = 0.00001
- A e C nell'arco di 10 giorni = 0.00001
- A e D nell'arco di 10 giorni = 0.00001

} 3*0.00001 = 0.00003

(in generale $(D-1)*MTR*MTF^2$)

Prob. che si guastino nell'arco di 10 giorni

- B e C oppure B e D oppure C e D 0.00003

una qualunque coppia di dischi = 0.00006

(in generale $D*(D-1)*MTR/(2*MTF^2)$)

quindi

Con 4 dischi da 4 GB tali che:

- Mean Time to Failure (MTF) = 1000 giorni
- Mean Time to Repair (MTR) = 10 giorni

	Memoria utilizzabile	Probabilita' di perdere dati
liv. 0	16 GB	0.004
liv. 1	8 GB	0.00002
liv. 5	12 GB	0.00006

Esercizio 1

Un file è dotato di una struttura a record di 512 byte ed è allocato su un disco caratterizzato da blocchi anch'essi di 512 byte.

Si considerino le tre strategie di memorizzazione del file, contigua, concatenata e indicizzata, e si supponga che le informazioni che riguardano il file (i-node), in ognuno dei tre casi, siano già in memoria centrale.

L'ultimo record letto dal file è il record 3,
Il prossimo record da leggere è il record 8.

Per ognuno dei tre casi, si calcoli quanti accessi a disco sono necessari per la lettura del record 8 e si motivi la risposta.

soluzione

- Notiamo innanzitutto, che, date le dimensione di record e blocco sopra specificate, ogni record del file occupa esattamente un blocco del disco. Quindi:
- allocazione contigua: **1 accesso alla memoria** di massa permetterà di reperire direttamente il record 8.
- allocazione concatenata: **5 accessi alla memoria**, in quanto i record che separano il terzo dall'ottavo devono essere letti sequenzialmente.
- allocazione indicizzata: **1 access0 alla memoria**, (perche' gia' ho avuto accesso al blocco indice).

Esercizio 2

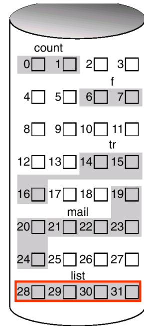
Si consideri un file formato da 70 record e le sue possibili allocazioni su disco di tipo contigua, concatenata e con tabella indice. In ognuno di questi casi i record del file sono memorizzati uno per blocco.

Le informazioni che riguardano il file sono già in memoria centrale e la tabella indice è contenuta in un unico blocco.

Si dica quanti accessi al disco (letture e/o scritture) sono necessari, in ognuna di queste situazioni, per cancellare (eliminare):

1. il primo record
2. il 40-esimo record
3. l'ultimo record

Soluzione (all. contigua)



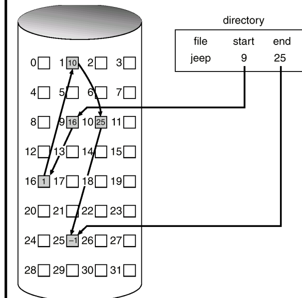
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- per cancellare il primo record e' sufficiente modificare il puntatore al blocco di inizio ➔ 0 accessi al disco
- per cancellare l'ultimo record e' sufficiente modificare l'indicazione della lunghezza del file ➔ 0 accessi al disco
- per cancellare il 40-mo record e' necessario:
  - copiare il 41-mo sul 40-mo blocco
  - copiare il 42-mo sul 41-mo blocco
  - ……
  - copiare il 70-mo sul 69-mo blocco

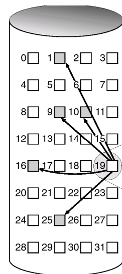
30 accessi in lettura e 30 in scrittura

Soluzione (all. concatenata)



- per cancellare il primo record e' necessario modificare il puntatore del blocco di inizio con il valore che si trova nel primo blocco ➔ 1 accesso al disco
- per cancellare l'ultimo record e' necessario modificare il puntatore del penultimo blocco ➔ 68 accessi in lettura e 1 in scrittura al disco
- per cancellare il 40-mo record sono necessari 40 in lettura per ottenere l'indirizzo del 39° e 41° blocco e 1 in scrittura per modificare l'indirizzo contenuto nel 39° blocco

Soluzione (all. indicizzata)



directory

file	index block
jeep	19

- per cancellare il
  - primo record
  - l'ultimo record
  - il 40-mo record
- e' necessario modificare solo l'indice del file ➔ 1 accesso in lettura (per accedere a tutto l'indice) e 1 in scrittura

Esercizio 3

Gli i-node di UNIX impiegano indirizzamento di 12 blocchi diretto e 1 indirizzamento indiretto singolo, doppio e triplo per localizzare i blocchi di un file fisico.

Si supponga che ciascun indirizzo occupi quattro byte.

Si calcoli la dimensione del più grande file che può essere gestito con questo schema di indirizzamento, per dimensioni dei blocchi di 512, 1024 e 4096 byte.

Soluzione dim blocchi=512

- Blocchi di 512 byte
- Indirizzi di 4 byte
- 512/4 = 128 indirizzi per blocco

- Dimensione massima per un file
  - 12 * 512 byte = 6 Kbyte +
  - 128 * 512 byte = 64 Kbyte +
  - 128^2 * 512 byte = 8 Mbyte +
  - 128^3 * 512 byte = 2 Gbyte

Soluzione dim blocchi=1024

- Blocchi di 1 kbyte
- Indirizzi di 4 byte
- 1024/4 = 256 indirizzi per blocco

- Dimensione massima per un file
  - 12 * 1 Kbyte = 12 Kbyte +
  - 256 * 1 Kbyte = 256 Kbyte +
  - 256^2 * 1 Kbyte = 64 Mbyte +
  - 256^3 * 1 Kbyte = 16 Gbyte

Soluzione dim blocchi=4096

- Blocchi di 4 kbyte
- Indirizzi di 4 byte
- 4096/4 = 1024 indirizzi per blocco

- Dimensione massima per un file
  - 12 * 4 Kbyte = 48 Kbyte +
  - 1024 * 4 Kbyte = 4 Mbyte +
  - 1024^2 * 4 Kbyte = 4 Gbyte +
  - 1024^3 * 4 Kbyte = 4 Tbyte

Esercizio 4

Sia dato un file system dove la dimensione del blocco e' B=1024 bytes e la dimensione dell'indirizzo di blocco p=2 byte.

Supponiamo che lo schema di allocazione sia indicizzato mediante indirizzamento di
10 blocchi diretti
1 blocco indiretto di primo livello
1 blocco indiretto di secondo livello
1 blocco indiretto di terzo livello

Sia dato un file nel file system descritto.

Il byte 300.000 del suddetto file si trova in un blocco dati diretto, indiretto, doppiamente indiretto o triplamente indiretto ?

Soluzione

Un file in Unix è un flusso di bytes contenuti in blocchi del file system.

300000 / 1024 = 292.96 → Il byte 300.000, in un file system di blocchi da 1024 bytes, è contenuto nel blocco 293 del file.

Nel file system in oggetto, ogni blocco può contenere 1024/2 = 512 indirizzi

I primi 10 blocchi (da 0 a 9) di un file sono indirizzati direttamente.
I blocchi da 10 a 521 (9+512) del file sono singolarmente indirizzati.
I blocchi da 522 a (521 + 512*512) sono doppiamente indirizzati.
I blocchi da (521 + 512*512) a (521 + 512*512 + 512*512*512) sono triplamente indirizzati.

Perciò il byte 300000, che si trova nel blocco 293 del file, è contenuto in uno dei blocchi *singolarmente indirizzati* (dal 10 al 521).

Esercizio 5

- La File Allocation Table della Microsoft e' un esempio di allocazione tabellare dei file . Si determini la dimensione in byte di una FAT nei casi di
  - un floppy disk da 1.44 Mbyte che usa blocchi da 1 Kbyte
  - un disco da 200 Gbyte che usa blocchi da 4 Kbyte

soluzione

- Floppy disk da 1.44 Mbyte
  - 1440 blocchi da 1 Kbyte
  - Necessari 11 bit (2^{11} = 2048 indirizzi)
  - Dim tabella = 11*1440 = 15840 bit = 1980 byte
- Disco da 200 Gbyte
  - ~ 50 milioni di blocchi
  - Necessari 26 bit (2^{26} = 67108864 indirizzi)
  - Dim tabella = 26*50*10^6 = 1300* 10^6 bit ~ 130 Mbyte

Esercizio 6

- Un file system puo' tenere traccia dei blocchi liberi mediante
  - Bitmap
  - Lista dei blocchi liberi
- Supponendo che il sistema ha un totale di
  - T blocchi,
  - U dei quali sono usati
  - ogni blocco e' memorizzato usando S bit,

1. si calcoli la dimensione in bit della bitmap e della lista dei blocchi liberi
2. Fissati S e T, si determini inoltre per quale valore di U l'occupazione della bitmap e' inferiore a quello della lista dei blocchi liberi

soluzione

- Bitmap
  - La bitmap tiene traccia di ogni blocco (sia occupato o libero) con un solo bit
  - Dimensione e' sempre T (numero di blocchi totali)

- Lista dei blocchi liberi
  - Tiene traccia dei blocchi liberi
  - (T-U) numero di blocchi liberi
  - S bit per ogni blocco
  - Dimensione totale S(T-U)

- fissati T e S
  - occupazione bitmap > occupazione lista se e solo se
    S(T-U) < T cioe' U > (ST-T)/S