

Il file system

- Il file system è la parte del SO che si occupa di mantenere i dati/programmi in modo persistente
- Tipicamente le astrazioni fornite sono:
 - **File**: unità di informazione memorizzata in modo persistente
fornisce, inoltre, uno strumento semplice per permettere a più processi di condividere informazioni e/o comunicare (es. pipe) *
 - **Directory**: astrazione che permette di raggruppare assieme più file (contenitore che racchiude file e directory)

1

* Consente di utilizzarli come nome, estensione, data ecc.
Avete solo file ma dei problemi, sempre meno ovvero per ogni file!

Gestione del File System

- Interfaccia verso l'utente
 - file e system call su di essi
 - directory e system call su di esse
 - comandi/interfaccia grafica (realizzati richiamando le system call)
- Implementazione del file system
 - file system logico (reperimento dei file all'interno della gerarchia delle directory)
 - memorizzazione dei files (allocazione sui blocchi liberi del disco)

↳ Noi gestiamo come i vari OS gestiscono l'interfaccia file e directory.
1. Dove li memorizzo? Disco, penne, ecc. Problema: dato file, su quali blocchi del disco sono memorizzate le sue info?

2

Attributi di un file

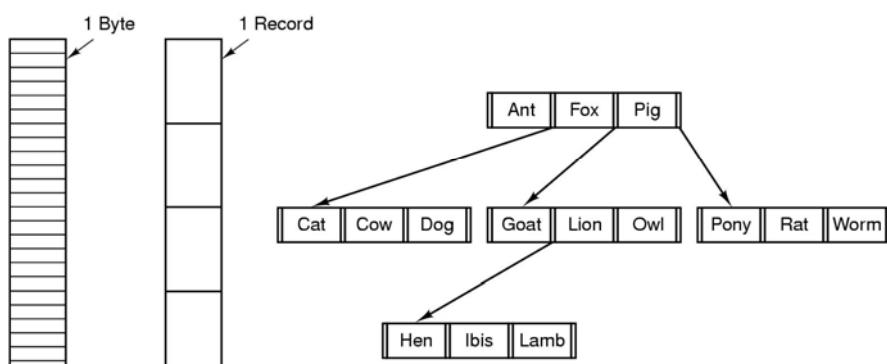
Il file è un "contenitore" per un insieme di informazioni correlate. L'utente deve essere in grado di identificare un file attraverso il suo nome che deve essere diverso per ogni file nel sistema. Ad ogni file inoltre sono associati alcuni attributi:

- dimensione (attuale ed eventualmente massima)
- data di creazione, modifica
- protezione (diritti di accesso)
- identificatore utente/gruppo proprietario
- ... altro

Fra gli attributi possono esservene alcuni che codificano il tipo del file. A volte tuttavia il "tipo" di file (che identifica una data codifica standard, oppure una applicazione capace di interpretarlo) è codificato in un suffisso del nome chiamato "estensione".

3

Struttura di un File



- Come può essere strutturata l'informazione all'interno di un file
 - sequenze di byte, sequenze di record a dim. fissa, alberi di record a dim. variabile con accesso a chiave

4

Com'è strutturato il file può essere strutturato? Modo più semplice: no struttura, solo sequenza di byte. Posso accedere alla metà al file col m.byte. Architettura dei gran parte dei file system.
(INDIPENDENTE DA QUESTO ESTENSIONE)

Potrei avere struttura di file da record, accedo a record n.

Tipo di accesso logico ai file

- Sequenziale

- a partire dall'inizio del file in lettura
- a partire dall'inizio o dalla fine (append) del file in scrittura

- Random (legato alla struttura del file)

- a partire dal byte n
- del record n
- del record con chiave "pippo"

Suggerisce che accedo non mettendo nel memoriale all'inizio.

5

Tipo di accesso fisico ai file

Il tipo di accesso è legato al tipo di dispositivo fisico):

- Sequenziale

- es. nastri magnetici, ma anche i CD-ROM in effetti sono "quasi" sequenziali, infatti i dati sono memorizzati su una lunga linea avvolta a spirale sul CD

- Random

- nato con i dischi magnetici che consentono l'accesso ad un blocco specifico del supporto

6

*Gli indirizzi per accedere è numero da blocco, che mi dà byte di blocco.
Posso leggere/scrivere blocco n.*

DOMANDA ESAME

- A parte da syscalls, specificando in file system, quali sono le op. che il SO fa se qualcuno chiama da creare file, leggere file ecc. IMPLEMENTAZIONE DA FILE SYSTEM.

System call relative ai file

1. Create
2. Delete
3. Open } (utili per predisporre alcune strutture dati del S.O.)
4. Close } al fine di rendere più efficienti le R/W successive)
5. Read } (parametri variabili a seconda del tipo di struttura
6. Write } e del tipo di accesso)
7. Append
8. Seek (consente il posizionamento nel caso di accesso random)
9. Get attributes
10. Set Attributes
11. Rename

7

Esempio

```

int fd;
...
fd=open (pathname, ...);
if (fd=-1) /* gestione errore */
.....
read (fd, ...);
.....
close(fd);
```

NB: Un file può essere aperto più volte, e quindi avere più file descriptor associati contemporaneamente

8

Tabella dei file aperti (vista logica)

Ogni volta che si fa una open di un file viene aggiornata una tabella del S.O. chiamata **tabella dei file aperti**. Nella tabella vengono mantenute le informazioni per reperire velocemente i contenuti del file da disco.

`fd = open(/d1/f1, "R")`

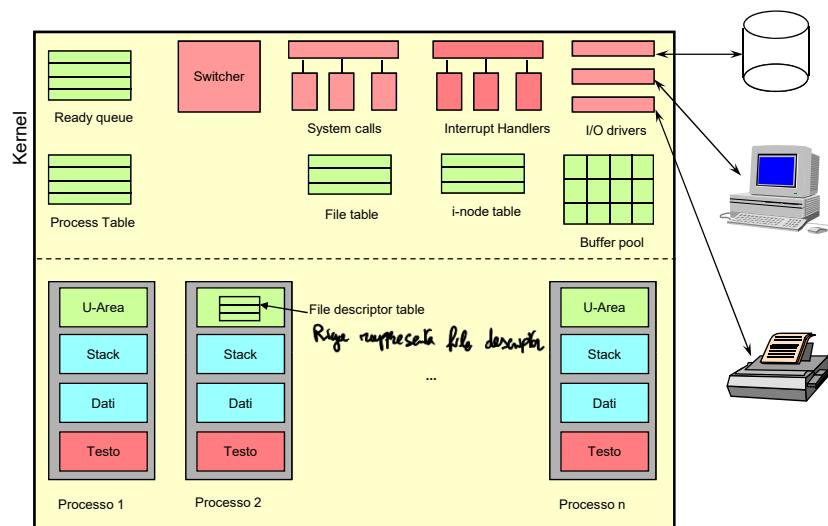
Tipo accesso	Cursore	Attributi	Allocazione file su disco
R	0		inizio = Blocco 30

Ogni successiva read/write deve fare riferimento al file tramite il **file descriptor fd** restituito dalla open.

9

Fase di apertura è quella che costa dei primi. Prevede memorizzare dati **info utili** per read/write successive. Es. mod. apertura, cursore, info che consentono di fare altro a seconda del file system.
 A OPEN (fino a close) ho val. di info sufficienzi per quel file e processa.
 Il file descriptor è indice di accesso alla tabella. File è in genere sparpagliato su disco, che compone le cose. In un file system però, con file dinamici non posso avere blocchi consecutivi.

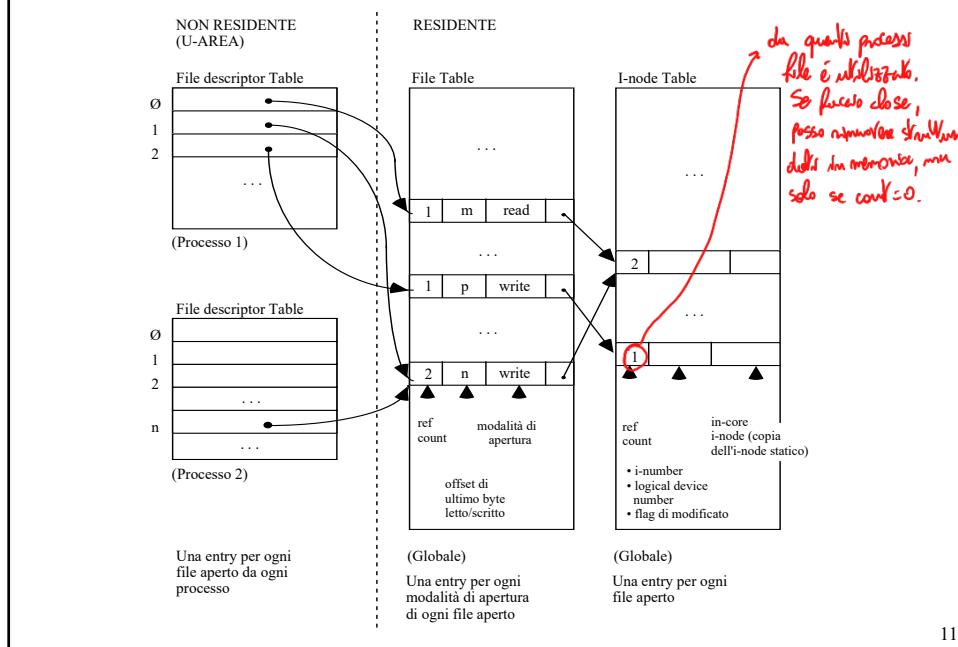
Gestione dei file: visione d'insieme



Nota: se ho file aperti da processi diversi, questi si trovano nella tabella di file dei processi entrate diverse. Ci sono però informazioni anche se file è aperto su più processi. Ultimo punto della tabella dei file aperti: il file aperto è stocca che ha memorizzato più volte. Con Unix, le info associate al file. Ogni file ha I-NODE. Tabella degli I-NODE dei file aperti è connessa con processi che usano lo stesso file.

09/05/2017

Gestione dei file: strutture dati del kernel



11

NOTA: Vibre cantiche solo struttura dati in memoria per il file, non tutto il file.

Ci sono casi particolari (es. Fork con padre e figlio), il figlio si trova i file aperti dal padre, con due processi padre figlio che puntano allo stesso riga della tabella dei file aperti.

In OS diversi, nelle tabelle, invece di avere puntatore a tabella I-NODE per file aperto ho visto come "dove è memorizzati primo blocco fisico del file" perché mi permette di accedere agli altri.

[CASO FILE SYSTEM BASATI SU FAT].

[PER FS SU UNIX ABBIANO STRUTTURA PIÙ COMPLICATA: PUNTATORE AD INDEX NODO PER QUEL FILE CONTENUTO DELLA TABELLA FILE APERTI]

Le directory

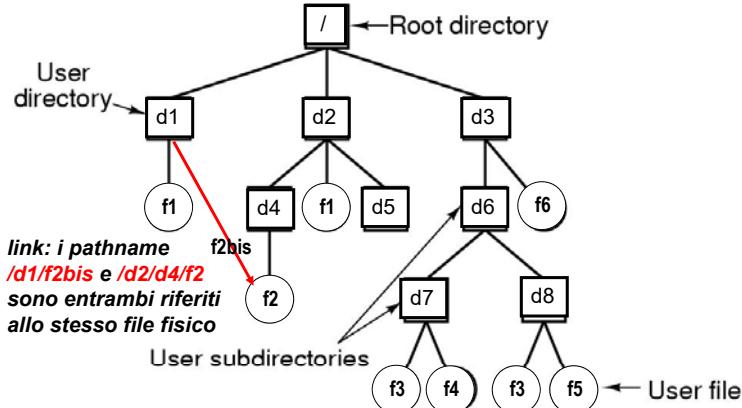
Le directory sono paragonabili a classificatori: contendono documenti, che nel nostro caso sono i file, e/o altri classificatori.

Possiamo avere un'unica directory, oppure una directory principale che contiene una sottodirectory per ciascun utente (in modo da evitare conflitti sui nomi dei file di clienti diversi), oppure ancora una gerarchia di directory (di profondità arbitraria).

La struttura della gerarchia è un albero, ma può essere utile permettere l'aggiunta di "archi" per dare la possibilità di "vedere" lo stesso file da due diverse directory: in questo caso la struttura diventa un grafo diretto aciclico (DAG).

12

Struttura gerarchica del file system



Si possono avere due o più pathname per uno stesso file, ma ovviamente non possono esserci due file con lo stesso pathname (ogni nome deve individuare univocamente un file).

13

System call relative alle directory

1. Create
2. Delete
3. ReadDir
4. Rename
5. Link
6. Unlink

La `readdir` consente di ottenere l'elenco dei file/sottodirectory contenuti in una data directory. Restituisce il prossimo elemento in una directory aperta (`opendir`)

Non è opportuno fornire all'utente funzioni di `read/write` "libere", analoghe a quelle disponibili per i file. Le `write` nella directory sono indirettamente richiamate ogni volta che si crea, cancella, rinomina o si modificano gli attributi di un file in quella directory.

14

Implementazione di un File System

- Come rappresentare i file ?
 - per ogni file si devono memorizzare gli attributi e la posizione dei singoli blocchi sul disco che ne memorizzano il contenuto
- Come rappresentare le directory ?
 - generalmente sono file con uno speciale formato che contengono indicazioni sui file/sottodirectory contenuti
 - consentono di individuare il singolo file all'interno della struttura ad albero
- Come organizzare lo spazio disco ?
 - assegnazione dei blocchi relativi ad un singolo file
 - gestione blocchi liberi
 - dimensionamento ottimale dei blocchi

15

File System deve sapere dove file i suoi dati su quali blocchi vengono memorizzati.

Rappresentazione dei file

Per ogni file va registrato il nome, gli attributi e dove memorizzare il contenuto sul dispositivo di memoria secondaria

Il problema principale è capire come allocare il contenuto di un file sui blocchi fisici (a dimensione fissa) del disco.

Tipi di allocazione dei file su disco:

- Contigua
- A lista concatenata
- Con nodo indice

16

Allocazione contigua: Pro: ho modo semplice per immagazzinare nella struttura dati associata al file dove sono salvati i file su disco. Serve solo punti di partenza e dimensione del file.

09/05/2017

PROBLEMA: frammentazione che si crea. Qua deframmentare è ancora più oneroso perché lavoro su disco.

NOTA: Vediamo logicamente il disco come un array di blocchi che posso usare.

Allocazione contigua



Situazione dopo l'allocazione del File A (4 blocchi) e B (3 blocchi)

File B (3 blocchi)

File D (3 blocchi)

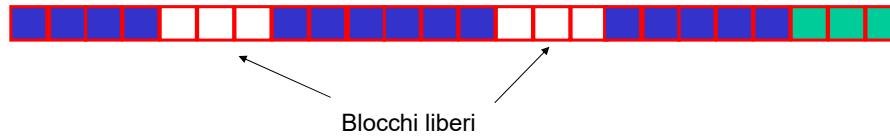
File G (3 blocchi)

File A (4 blocchi)

File C (6 blocchi)

File E (5 blocchi)

Situazione dopo la cancellazione di B e D



17

PROBLEMA GRAVE: Movimento del file rende tutto problematico

Allocazione contigua (vantaggi)

- Vantaggi dell'allocazione contigua:
 - è facile tenere traccia dei blocchi che appartengono a ciascun file
 - indirizzo su disco del blocco iniziale (B)
 - lunghezza del file (in blocchi)
 - se voglio l'indirizzo del blocco X del file
 - $\text{ind}(X) = B + X$
 - le prestazioni della lettura sono eccellenti
 - basta una seek ed è possibile leggere tutto il file in blocchi contigui (senza rotational delay ...)
↳ spostamento testina = problema!

18

Allocazione contigua (svantaggi)

- Svantaggi dell'allocazione contigua:
 - frammentazione interna: si spreca spazio nell'ultimo blocco
 - frammentazione esterna : esempio

Situazione dopo la cancellazione di B e D



- dopo un po' il disco sarà frammentato in un insieme di buchi (hole), troppo piccoli per contenere un file
- de-frammentazione del disco :



19

Allocazione contigua (svantaggi)

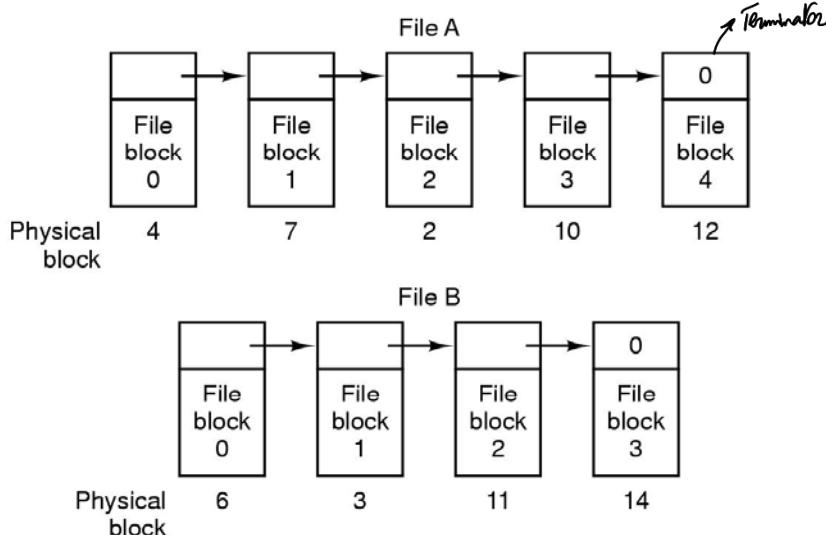
- Svantaggi dell'allocazione contigua (cont.):
 - l'ampiezza massima di un file deve essere decisa al momento della creazione
 - dobbiamo riservare un numero adeguato di blocchi per la sua crescita futura
 - ancora spazio sprecato
 - difficile da stimare ed utilizzare : es prima di iniziare ad editare un file di testo devo dire quanto sarà lungo ... altrimenti l'editing può fallire!!!!
 - L'allocazione contigua viene utilizzata nei FS dei CD ROM e DVD !!

20

Locazione non continua = chiedi file da m byte (file è una seq. di byte, in memoria),
devo memorizzare la sequenza dei numeri dei blocchi fisici su cui sono memorizzati i dati del file.
09/05/2017

Soluz. 1: memorizzo solo il primo n. blocco fisico su cui si trova blocco logico
Per ogni blocco leggo ho puntatore al prossimo blocco logico.

Lista concatenata di blocchi



Memorizzazione come lista concatenata di blocchi

21

Lista concatenata di blocchi

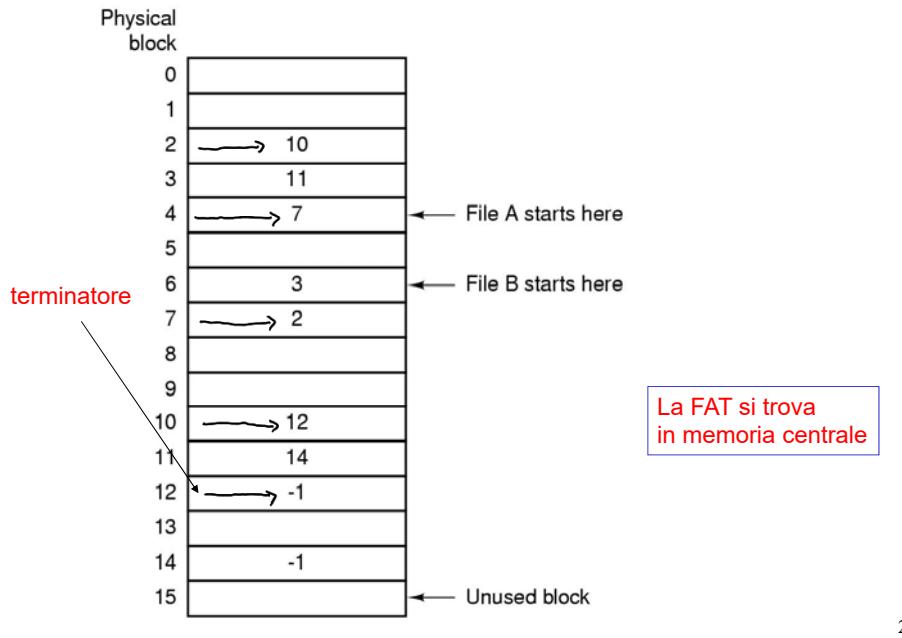
- Vantaggi della lista concatenata :
 - non c'è frammentazione esterna
 - per ogni file è necessario mantenere solo il puntatore al primo blocco
- Problemi :
 - lettura di tutto il file molto lenta
 - accesso random molto lento (per accedere al blocco n devo leggere gli n-1 blocchi precedenti)
 - perdita di un certo numero di byte iniziali *
 - l'ampiezza di blocco non è più una potenza di 2 (diventa più costoso il calcolo del blocco)

22

* Devo memorizzare n. del blocco fisico successivo.
Complicati calcoli per sapere dove è memorizzato uno specifico byte.

Prendo lista concatenata e la porto in RAM: file allocation table che porta lista partizioni per accesso ai file. 4-7-2-10-12 *

Liste concatenate con FAT



* Per i dati del file, devi salvare solo il primo indirizzo.

Struttura dati deve essere sempre permanente su disco, ma poi posso spostarla.

Quando ora per vedere i blocchi devi fare accesso alla fat per andare a blocchi.

Se devi partire da byte m, faccio m-1 accessi alla fat e un solo accesso a blocco fisico

Liste concatenate con FAT

Vantaggi :

- l'ampiezza di blocco è una potenza di due
- accesso casuale più veloce (accedere il k-esimo blocco di un file costerà $k-1$ accessi alla memoria più 1 su disco)

Svantaggi :

- tutta la FAT deve stare in memoria:

Es: disco 20 GB, blocchi 1K \Rightarrow ho $\frac{20 \text{ GB}}{1\text{KB}} = 20 \cdot 2^{30}$ blocchi \Rightarrow 20M blocchi

• ampiezza FAT = $4 \cdot 20\text{M}$

m byte che rappresenta m che va da 0 a 2^{30} .
qui ci deve stare m di blocchi fisi. \Rightarrow per rappresentare con $20 \cdot 2^{30} \approx 2^5 \cdot 2^{30} \Rightarrow 25$ bit per avere quel numero.

Dipende da: dimensione della partizione, del blocco logico.

\Rightarrow Primo multiplo maggiore è 32bit = 4 byte.

Dovrò avere tutte quelle quantità sono \rightarrow blocchi fisi.

\Rightarrow Lo ho fatto (dimensione partit.) / (dimensione blocco logico)

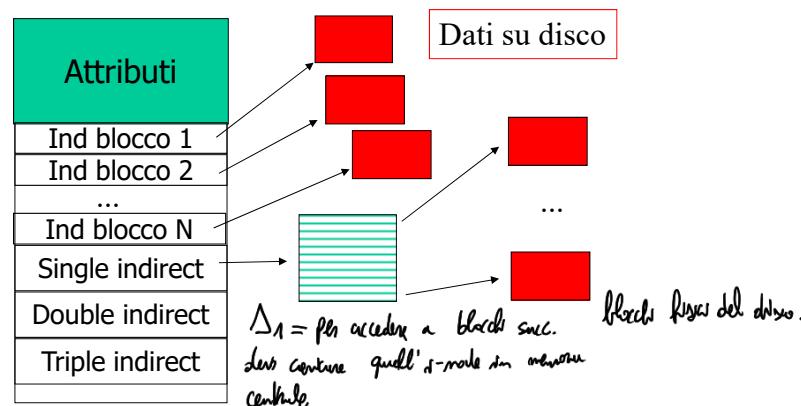
Ogni riga contiene un m di blocchi fisi che dipende da quanti sono i blocchi fisi.

Immagina disco fisso con 100 file che occupano tutti i blocks. Quanti -i- devi avere? 100. E nella tabella FAT mancano i numeri associati a dove è memorizzato il primo blocco logico nel file.
 Soluzione di windows: salvare indirizzo del primo blocco nella directory del file (e dovrà memorizzare anche gli attributi)

09/05/2017

Index-node (i-node)

- Es. *i-node* di un file Unix



25

Per Linux non ha struttura dati globale, ma ogni file ha la sua tabella, index-node, che ha le seguenti colonne: nome, indirizzo dei blocks su disco. Salvo una tabella per file, non una globale con tutti i file. NOTA: Gli si appunto per salvare le info del file. Salvo tutti i nome (permes., proprietario, data ultima modifica ecc.). Il nome lo dà nome all'interno delle directory, perché saranno le chiavi per accedere ai file. Alla tabella ci arriva da un numero in una directory.*

Index-node (i-node) (2)

- Vantaggi :
 - solo gli i-node dei file in uso devono risiedere in RAM
 - lo spazio è proporzionale al numero massimo di file aperti e non dipende dall'ampiezza del disco
 - l'efficienza decresce con l'ampiezza del file

26

* Problema: per rendere questa struttura facilmente accessibile devi avere dim. massima uguale per tutti i file. La lista può essere di lunghezza diversa per file differenti, che complica trovare l'index-node su disco. Soluzione: mette n primi x blocks logici sono punti di saltarne, però non posso continuare così. Dall'X+1, quel numero non è più n. blocco logico dato è min. X+1 blocks logici. Sarà sempre n. di blocco logico, che punta però ad un pezzo della tabella. $\Delta_1 \Rightarrow$ Voglio i-node tabella che siano sempre uguali in dimensione.

Directory è sempre una sequenza di byte. È un file. Ha una struttura regolare però: ha tante entry, quelli gli elementi nella directory (file e subdir). Per ogni entry, che deve memorizzare? Il nome in prima. Questi devono chiave chi ricerca nella struttura dell'os. 2^a campo = attributi (protezione, dati modifica, metadati in generale). Ultima informazione: punto blocco.
PROBLEMA: Quanti byte richiede il campo nome? Se ho superato il nome, posso spostare da modo più rapido tra i cartelle e in allie. FAT+directory ci dà tutte le info necessarie.

09/05/2017

Implementazione delle directory FAT

Le directory sono files, con contenuto strutturato in modo specifico: una tabella le cui entry contengono il nome del file/sottodirectory, attributi corrispondenti, informazioni sulla allocazione dei blocchi del file su disco.

Nome	Attributi	Allocazione file su disco
d1		inizio = Blocco 30
d2		inizio = Blocco 43
f1		inizio = Blocco 110

Ogni entry della tabella ha una dimensione fissa (limitazioni sulla lunghezza dei nomi)

Questa organizzazione pone alcuni problemi per l'implementazione dei link.

27

Unico attributo non un I-node è nome. Tante righe quindi sono i file e le directory, più directory corrette e generate.

I-NODE

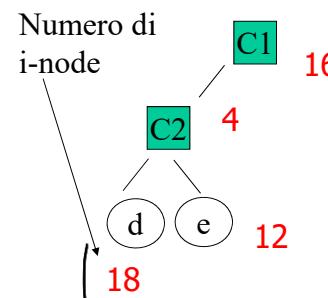
Implementazione delle directory

Una directory in un sistema con i-node

- es da Unix V7

4	.(punto)
16	..(punto punto)
12	e
18	d

Blocco dati relativo alla directory C2

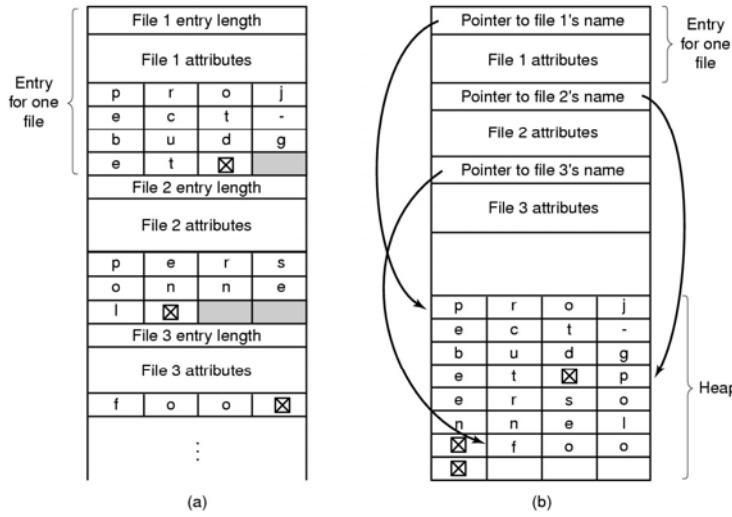


Molto più
Semplice: dove solo
avere chiavi di ricerca

28

Ho solo campo nome e numero dell'i-node, che deve consentire di riconoscere i-node da subito. Numero ci fa arrivare all'i-node perché i-node sono tutti uguali (dimensione circa 1K). Avrei bisogno i-node tutto uguali. Se so dove sta i-node attual e n-i-node si deve spostare.

Implementazione delle directory (nomi lunghi)



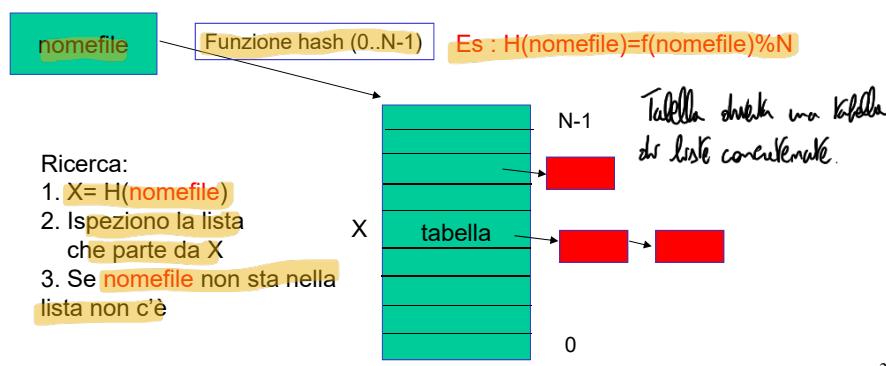
- Due modi di trattare i nomi di file “lunghi”
 - (a) In linea (b) In un heap

29

Gestire può essere complicato a seconda di come salviamo nome.

Implementazione delle directory (ricerca)

- Ricerca di un file X in una directory
 - tipicamente i nomi dei file non sono in ordine (ricerca lineare)
 - con directory con centinaia di file si possono usare hash table



30

Dallo punto punto dalla dir root come cello. Se file non c'è nella dir, vado di molla la directory.

$/d2/d4/f2 \Rightarrow$ cerco d2, se lo trovo, cerco d4, se lo trovo cerco f2 e trovo n. alt. n. n. al primo blocco. La ricerca deve essere lineare (o posso usare hash).

Gestione dello spazio disco (size blocco)

• Come scegliere l'ampiezza del blocco?

Bisogna trovare il giusto compromesso tra overhead dovuto a frammentazione interna (che richiederebbe blocchi più piccoli) e velocità di trasferimento dei dati, che richiede blocchi grandi (per ammortizzare il seek time).

Esempio: tempo per trasferire un blocco di n byte

$$10 + 4.165 + \frac{(n/131072) \times 8.33}{\text{seek rotazione}/2} \xrightarrow{\text{tempo giro completo}} \text{trasferimento di } n \text{ bytes (traccia di 128Kb)}$$

tempo di spostamento

Molto blocco occupato da file non posso usare con un altro file.

Blocchi piccoli per file = FAT + grande. Se ho file che 100KB, con blocc da 1KB e 258, 100 vs 50 blocchi. Se devo leggere tutto il file, una cosa è leggere 50 blocchi, un'altra 100.

31

NOTA: cercavo concatenico. Se so m. blocco dove scrivere su quella traccia. È tempo grande poiché \rightarrow (seek) richiede spostamento di organo meccanico. Per se sono sfortunati, dovrà fare molto spostamento molto giro di rotazione. Se impiego 8.33 sec. per un giro completo, dovrà far perdere 10 ms per ricaricarlo. Funzione minima è 1 byte $\Rightarrow \frac{1}{128}$ cycle. Avendo minimizzato alla larghezza contigua perde poco tempo. Se però ha la possibilità di allargare in modo contiguo lo file lo spazio sarà più grande e valendo anche riduzione.

* 128K

NOTA! Altra problematica: supponiamo che il file non sia composto da blocchi liberi e blocchi occupati.

Gestione dello spazio disco (blocchi liberi)

• Come tenere traccia dei blocchi liberi su disco?

– Free list

- lista concatenata di blocchi che contengono indirizzi di blocchi liberi

– Bitmap

- una mappa di bit con un bit per ogni blocco
- es. 0 blocco libero, 1 blocco in uso
- mantiene la contiguità dei blocchi (allocare un file su blocchi vicini diminuisce il tempo di seek in letture consecutive)

32

NOTA: Tutte le strutture dati che ci servono possono essere salvate in memoria RAM, ma dovrà essere preservata su disco. Possono essere in parti, ad esempio, un gestore.

• Ma serve sia quando sto creando nuovi file o lo modifio, sia quando elimino un file.

Gestione dello spazio disco (blocchi liberi)

Free disk blocks: 16, 17, 18

42
136
210
97
41
63
21
48
262
310
516

230
162
612
342
214
160
664
216
320
180
482

86
234
897
422
140
223
223
160
126
142
141

A 1 KB disk block can hold 256 32-bit disk block numbers

(a)

1001101101101100
0110110111110111
1010110110110110
0110110110111011
1110111011101111
1101101010001111
0000111011010111
1011101101101111
1100100011101111
0111011101110111
1101111101110111

A bit map

(b)

Free list

Bitmap

33

Tanti bit quanto ai blocchi non cui è dovuta partizione su disco. O libero. Vedo subito in fase di allocazione che da 0110110111110111 sono così allocare blocchi consecutivi per un file. E' la più economica scelta.
SOLUZIONE ALTERNATIVA: lista che rappresenta solo blocchi liberi. Salvate sempre su disco. Ogni blocco contiene un certo numero di blocchi liberi, almeno a my parola non bisogna

del prossimo blocco da gestire.

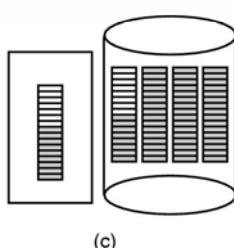
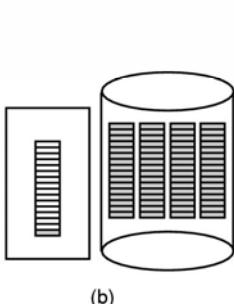
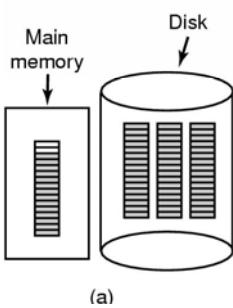
1. Bitmap grande m. dell'os
a m. blocchi. 2. Salvo solo
info su blocchi vuoti (memoria)
m. numero e non un bit.
Se parto, quando piena
controllare lista, altrimenti
bitmap.

NOTA: 2) ha vantaggi: mem
deve portare sempre tutto
in memoria. Magari solo
il primo pizz.

All'OS serve:

1. ho bisogno di blocchi
liberi e li prendo da
questa lista. (non metto
bene a prendere blocchi consecutivi)
2. Se cancello file posso aggiornare lista con blocchi appena liberati

Gestione dello spazio disco (blocchi liberi)



(a) blocco di puntatori ai blocchi liberi quasi pieno (RAM)

- tre blocchi di puntatori su disco

(b) situazione dopo aver liberato un file di 3 blocchi

(c) strategia alternativa per ridurre gli accessi al disco

- gli elementi in grigio puntano a blocchi di disco liberi

34

Consistenza di un File System

- Problema:

- tipicamente i file system leggono un blocco, lo modificano e scrivono la copia aggiornata più tardi
- se avviene un crash prima della scrittura della copia aggiornata il FS può trovarsi in uno stato inconsistente
- il problema è ancora più preoccupante se si tratta di i-node, informazioni di una directory o informazioni sui blocchi liberi

- Ci sono utility che

- controllano la consistenza di un file system (si possono servire di tabelle)
- lo riportano in uno stato consistente (eventualmente con la perdita di dati)
- es: scandisk (Windows) fsck (Unix)

35

Consistenza di un File System

Numero di blocco															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(a)

Numero di blocco															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1

(b)

Numero di blocco															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	0	0	1	1	1	0	0	0
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1

(c)

Numero di blocco															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(d)

- Lo hanno 2 volte nella lista libera
blocco grande ma va bene
- (a) consistente (consistent)
(b) blocco mancante (missing block)
(c) blocco duplicato nella lista libera (duplicate free block)
(d) blocco dati duplicato (duplicate data block)

Non occupato da nessun file ma non lo trovo nella lista libri liberi
blocco grande

Somma libri dati per 2 file!
Nella migliore delle ipotesi ho un file che ha perso parte

Se sono fortunato recupererò 1 file.

36

Tutti gli controllo su i-mode: devo recuperare due file da libri libri e b. libri. Metto nello array sotto (1 blocco su disco per libri) 1 se ho blocco corrispondente libri libri. Blocchi occupati a 0. Poi cerco di recuperare da libri di i-mode i blocchi utilizzati da file e faccio stessa cosa. Metto 1 se blocco occupato su i-mode.