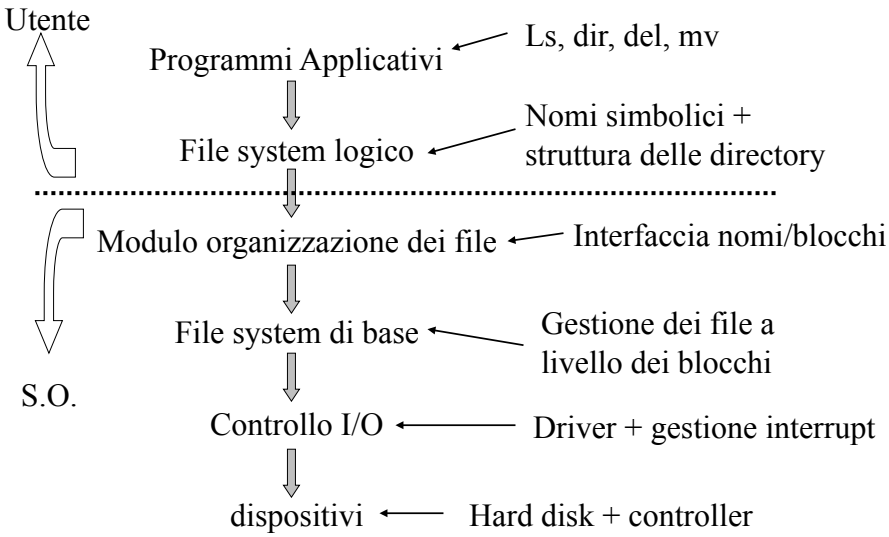
1

11 Realizzazione del File System

- Metodi di allocazione
- Allocazione contigua
- Allocazione concatenata e varianti
- Allocazione indicizzata e varianti
- Gestione dello spazio libero

2

11.1.1 Struttura a livelli (fig. 11.1)



11.4 Allocazione dei file

- Un HD è un supporto di memoria permanente in cui lo spazio è logicamente suddiviso in blocchi, di solito di 512 byte (ma altri valori sono possibili, ad esempio 256 o 1024 byte)
- Ogni blocco è numerato a partire da 0, per cui l'HD può essere visto come un enorme array in cui ogni entry è una sequenza di (ad esempio) 512 byte
- ogni blocco può essere acceduto (letto, scritto) direttamente semplicemente comunicando al controller del disco il numero del blocco interessato, e le operazioni di lettura e scrittura su HD avvengono sempre in unità di blocchi.

11.4 Allocazione dei file

- Il SO deve memorizzare i file all'interno dei blocchi dell'HD, suddividendoli tra più blocchi se la loro dimensione supera quella di un singolo blocco.
- Esistono fondamentalmente tre modi di base di allocare spazio sull'HD per i file:
  - allocazione **contigua**
  - allocazione **concatenata**
  - allocazione **indicizzata**

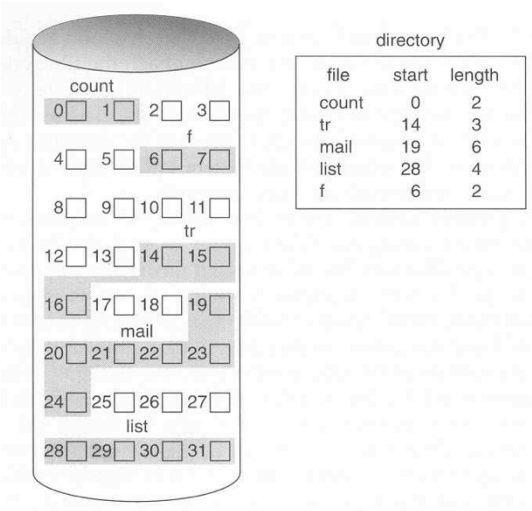
11.4.1 Allocazione contigua

5

- Ogni file è allocato in un insieme di blocchi dell'HD **contigui**
- Per recuperare i dati del file basta memorizzare, tra gli attributi del file:
  - il numero del primo blocco del file
  - quanti blocchi contigui sono occupati dal file (questo non è nemmeno strettamente necessario, dato che viene anche sempre registrata la dimensione del file)
- Questo metodo era usato nei sistemi IBM VM/CMS

11.4.1 Allocazione contigua (esempio, fig. 11.5)

6



directory

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

11.4.1 Allocazione contigua: vantaggi

- L'accesso ai vari blocchi del file è veloce e semplice ed è possibile sia l'accesso diretto che l'accesso sequenziale
- Poche informazioni sulla posizione del file sono sufficienti a registrare completamente quale parte dell'HD è occupata dal file

11.4.1 Allocazione contigua: problemi

- Per allocare un file è necessario trovargli uno spazio libero sul disco che sia **contiguo** (conosciamo già questa situazione...)
- È necessaria una strategia di allocazione dello spazio (come al solito, first/best/worst fit)
- e si verifica una **frammentazione esterna** del disco
- e può essere necessaria una **ricompattazione** periodica del disco (molto più costosa della ricompattazione della MP)

11.4.1 Allocazione contigua: problemi

- Ma se anche troviamo spazio sufficiente per allocare un file, che succede se la sua dimensione aumenta?
- siamo costretti a:
  - **riallocarlo** (spostarlo), ma è costoso
  - **sovradimensionarlo**, ma allora si aggrava il problema della frammentazione interna (comunque presente a causa dell'ultimo blocco)
- In definitiva, si presentano dei problemi simili a quelli visti con l'allocazione contigua in MP

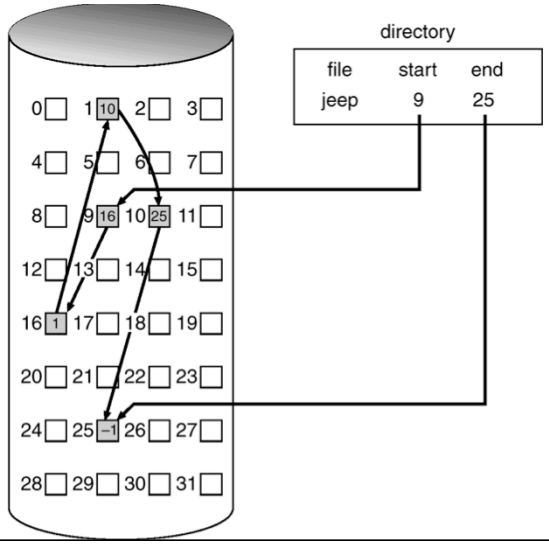
11.4.2 Allocazione concatenata

- Dato che anche in MS l'allocazione contigua dà problemi, dobbiamo ricorrere a sistemi simili a quelli visti per l'allocazione dello spazio in MP
- Nella **allocazione concatenata**, ogni blocco contiene (negli ultimi byte) un puntatore al blocco successivo del file (in sostanza, il numero del blocco in cui prosegue il file)
- Per sapere quali blocchi contengono il file basta memorizzare, tra gli attributi del file, il numero del blocco iniziale

11.4.2 Allocazione concatenata

- Opzionalmente, si può memorizzare anche il numero di blocchi usati e/o il numero del blocco finale
- Nell'ultimo blocco, nello spazio riservato per memorizzare il numero del blocco successivo viene scritto un numero negativo (o anche il valore 0), appunto per segnalare al sistema che quello è l'ultimo blocco del file

11.4.2 Allocazione concatenata (esempio, fig. 11.6)



11.4.2 Allocazione concatenata: vantaggi

13

- Non sono necessari blocchi contigui, per cui:
  - non c'è bisogno di ricompattare il disco
  - non si verifica frammentazione esterna
- Qualsiasi blocco libero può essere utilizzato per memorizzare un pezzo del file

11.4.2 Allocazione concatenata: problemi

14

- In ogni blocco, gli ultimi byte sono utilizzati per memorizzare il puntatore al (ossia il numero del) blocco successivo
- Nel caso di blocchi da 512 byte, con 4 byte per puntatore, circa lo 0,78% di un blocco non può essere utilizzato per memorizzare dati del file

11.4.2 Allocazione concatenata: problemi

- **l'accesso diretto ad una parte del file** (ad esempio al decimo blocco del file) **è altamente inefficiente:**
- in generale sono necessari n accessi al disco per leggere il blocco n-esimo, perché occorre leggere il primo blocco per sapere dove si trova il secondo, leggere il secondo per sapere dove si trova il terzo, e così via fino all'n-esimo
- **Il sistema di allocazione è poco affidabile:**
- se un blocco del file viene danneggiato, si perde tutta la parte di file a partire dal blocco danneggiato

11.4.2 Allocazione concatenata: Possibili soluzioni ai problemi

- **usare liste doppiamente concatenate:** se si danneggia un blocco possiamo recuperare i successivi ripercorrendo la catena all'indietro
- **memorizzare in ogni blocco il nome del file e la posizione del blocco all'interno della catena:** scandendo tutti i blocchi del disco, possiamo recuperare quelli persi

11.4.2 Allocazione concatenata. Possibili soluzioni ai problemi

17

- Non usare blocchi singoli, ma **considerare tutto l'HD formato da cluster di blocchi adiacenti**
- Ad esempio, ciascun cluster puo' essere composto da 4 blocchi del disco adiacenti, e avere quindi una dimensione di 2 o 4 Kbyte.
- in pratica è come lavorare con blocchi di dimensione più grossa, per cui:
  - i tempi di accesso al file diminuiscono, perchè dobbiamo riposizionare meno volte la testina di lettura
  - meno spazio viene sprecato per i puntatori, però:
  - aumenta la frammentazione interna

11.4.2 Allocazione concatenata: la variante della FAT

18

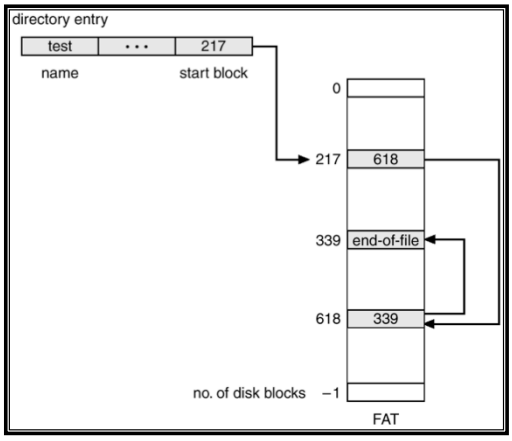
- Sebbene l'allocazione concatenata sia ampiamente insoddifacente, una sua variante molto efficiente era comunemente adottata dei sistemi ms-dos, OS2 ed è disponibile nei sistemi Windows come alternativa a NTFS
- **File Allocation Table (FAT):** area all'inizio del disco (in pratica è un array) in cui ogni entry corrisponde ad un blocco, e contiene il numero di un blocco
- Le entry che contengono uno zero corrispondono a blocchi del disco liberi

11.4.2 Allocazione concatenata: la variante della FAT

- La FAT registra lo stato di allocazione di tutti i blocchi dell'Hard Disk, e riproduce la lista concatenata di blocchi di ogni file
- In generale, se il blocco I punta al blocco N, allora nella FAT l'I-esima entry contiene il numero N
- Se l'ultimo blocco del file è il numero J, la J-esima entry della FAT contiene un marker speciale di fine file

11.4.2 un esempio di FAT

- Tra gli attributi del file "test" viene memorizzato il numero del primo blocco che contiene i dati del file (217).
- Per sapere quali altri blocchi contengono i dati del file "test" basta usare il numero del primo blocco come indice nella FAT, e poi seguire la catena di puntatori (fig. 11.7)



11.4.2 FAT: vantaggi

- Per poter usare in maniera efficiente la FAT, questa deve essere costantemente tenuta in RAM
- L'accesso diretto ai file migliora, perchè invece di seguire la catena concatenata dei blocchi su disco la si può percorrere nella FAT, che sta in RAM
- Il sistema è più sicuro in caso di perdita di un blocco
- La gestione dei blocchi liberi è automatica

11.4.2 FAT: svantaggi

- La FAT occupa una gran quantità di spazio in MP: è un array con un numero di entry uguale al numero di blocchi sul disco. Inoltre, ogni entry deve poter contenere il numero di un blocco
- Quindi l'uso della FAT diventa tanto più problematico quanto più aumentano le dimensioni medie degli HD (l'uso di cluster può mitigare parzialmente questo problema)
- Se la FAT viene persa non c'è più modo di accedere ai dati dei vari file, per cui deve essere frequentemente salvata sull'hard disk

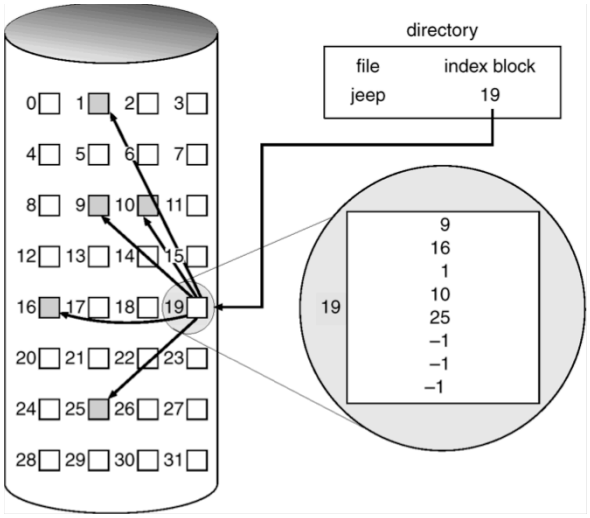
l'overhead di memoria della FAT

- Provate a calcolare quanto spazio è necessario per memorizza la FAT di un hard disk con queste caratteristiche:
  - capacità dell'hard disk: 20 giga byte
  - dimensione di un blocco dell'hard disk: 1 kilo byte
- Qual è la percentuale di spazio del disco "sprecata" per memorizzare la FAT?
- Per risolvere l'esercizio, di quanti byte abbiamo bisogno per memorizzare il numero di un blocco dell'hard disk?

11.4.3 Allocazione indicizzata

- Teniamo direttamente traccia di tutti i blocchi in cui è contenuto un file scrivendo il loro numero in un altro blocco del disco (detto **blocco indice**).
- Per recuperare i dati del file basta memorizzare, tra gli attributi del file, il numero del suo blocco indice
- osservate che questo approccio è simile alla gestione della memoria primaria nei sistemi paginati

11.4.3 Allocazione indicizzata (esempio, fig. 11.8)



11.4.3 Allocazione indicizzata: vantaggi e svantaggi

- Non sono necessari blocchi contigui, per cui non si crea frammentazione esterna
- L'accesso diretto e l'accesso sequenziale sono efficienti
- un blocco indice viene sempre sprecato per memorizzare il numero degli altri blocchi: se il file è piccolo il blocco indice è quasi tutto vuoto...

11.4.3 Allocazione indicizzata: problemi

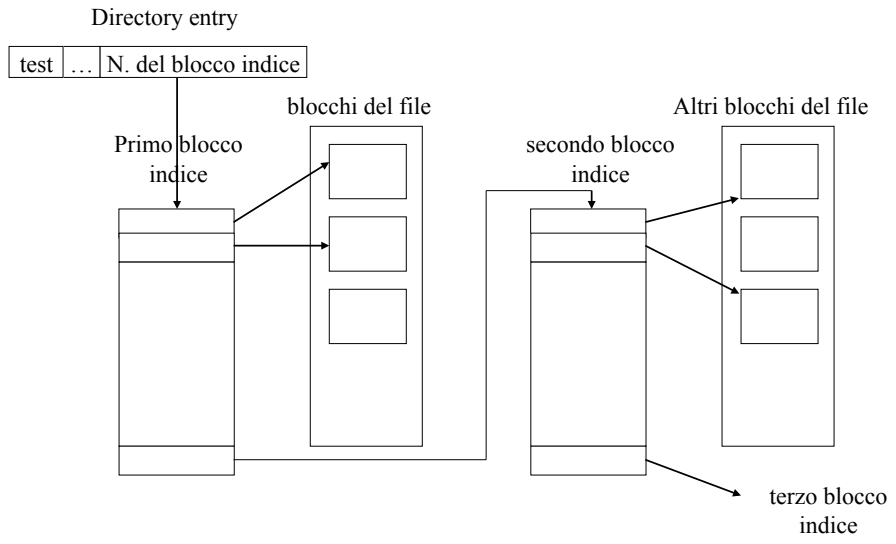
27

- E’ comunque necessario risolvere un problema: che succede se un blocco indice non è sufficiente per memorizzare i numeri di tutti i blocchi dati del file?
- Ci sono due soluzioni possibili di base:

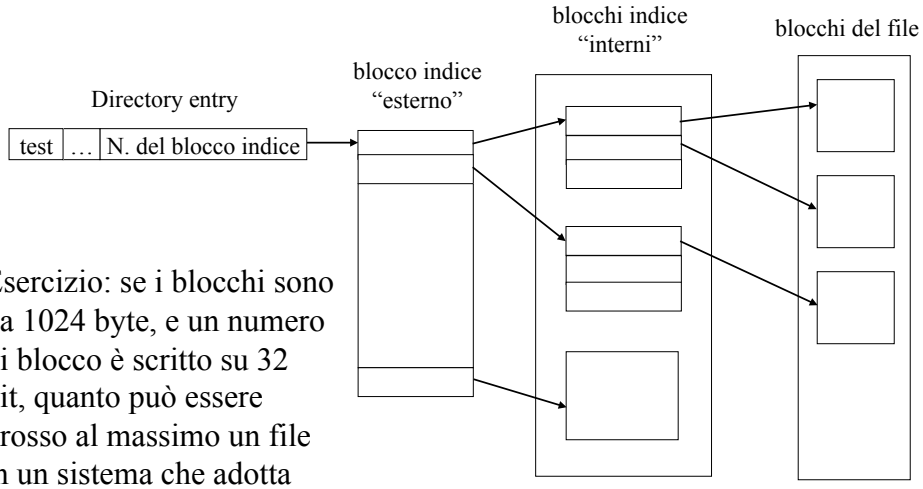
1. l’ultima entry del blocco indice punta ad un secondo blocco indice (**schema concatenato**)
2. il blocco indice contiene solo puntatori ad altri blocchi indice (**schema a più livelli**)

11.4.3 Allocazione indicizzata: lo schema concatenato

28



11.4.3 Allocazione indicizzata: lo schema a più livelli



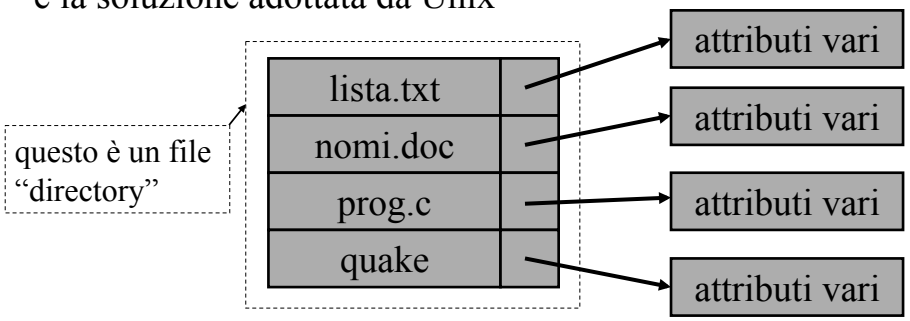
Esercizio: se i blocchi sono da 1024 byte, e un numero di blocco è scritto su 32 bit, quanto può essere grosso al massimo un file in un sistema che adotta questo schema?

11.4.3 Gli i-node Unix

- In Unix, ad ogni file è associato un i-node (index-node) che contiene gli attributi del file, e che fa anche da blocco indice.
- Gli i-node sono gestiti direttamente dal SO e sono memorizzati permanentemente sull'hard disk in una porzione riservata al SO, di solito nella prima parte dell'hard disk stesso.
- Semplificando un po', possiamo pensare che i primi blocchi dell'hard disk vengano appunto usati per contenere ciascuno l'i-node di un file. Il numero del blocco indice (ossia dell'i-node) di un file viene scritto a fianco del nome del file nella directory che lo "contiene".

10.3 Le Directory

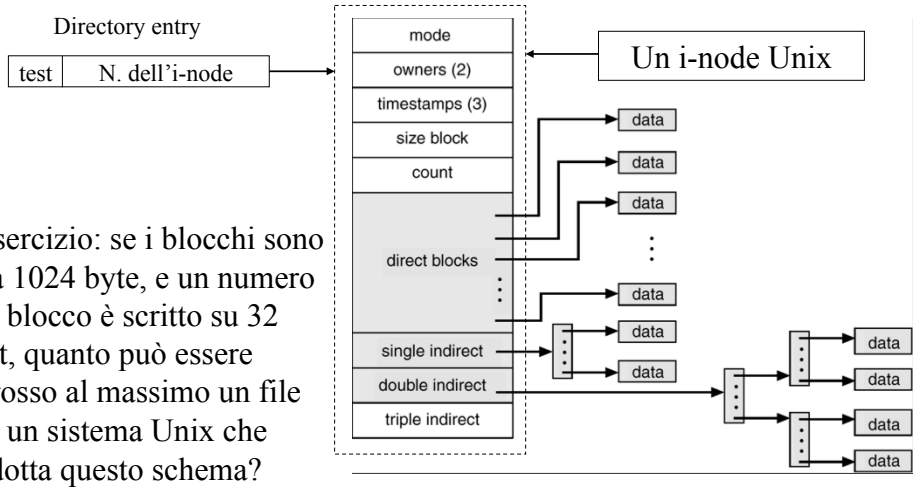
- (Ricordate questo lucido?) Alternativamente, possiamo inserire, a fianco del nome di ogni file solo un puntatore ad una struttura interna, anch'essa memorizzata in modo permanente sull'hard disk, e gestita direttamente dal SO, in cui sono contenute tutte le informazioni su quel file: questa è la soluzione adottata da Unix



11.4.3 Gli i-node Unix

- Per indirizzare i blocchi di dati del file, ogni i-node contiene:
  - 10 puntatori **diretti** a blocchi di dati del file,
  - un puntatore **single indirect** che punterà ad un blocco indice che conterrà puntatori a blocchi di dati file
  - un puntatore **double indirect** che punterà ad un blocco indice che conterrà puntatori a blocchi indice ognuno dei quali conterrà puntatori a blocchi di dati del file
  - Un puntatore **triple indirect** che punterà ad un blocco indice che conterrà puntatori a blocchi indice ognuno dei quali conterrà puntatori a blocchi indice, ognuno dei quali conterrà puntatori a blocchi di dati del file

11.4.3 Allocazione indicizzata: la soluzione degli i-node Unix (fig. 11.9)



Esercizio: se i blocchi sono da 1024 byte, e un numero di blocco è scritto su 32 bit, quanto può essere grosso al massimo un file in un sistema Unix che adotta questo schema?

11.4.3 NTFS

- Un approccio non molto diverso è usato nel file system adottato da Windows (XP e successivi): NTFS (New Technology File System).
- Ogni file è descritto da un **elemento**. Gli elementi di un file system NTFS hanno dimensione fissa (configurabile da 1 a 4 Kbyte all'atto della creazione del file system) e sono numerati consecutivamente.
- Tutti gli elementi di un file system sono contenuti in una **master file table** (MFT): un file memorizzato nei primi blocchi del disco e gestito esclusivamente dal SO.
- Il numero dell'elemento è associato al nome del file nella directory che "contiene" quel file, ed è detto **file reference**

11.4.3 NTFS

- Un elemento contiene tutti gli attributi del file corrispondente, come il proprietario del file, permessi di accesso, dimensioni correnti, date di creazione, accesso, modifica, ecc.
- Se il file è molto piccolo, anche i dati del file possono essere direttamente contenuti nel corrispondente elemento, in questo modo l'accesso al file risulta molto efficiente (con la sola lettura dell'elemento possiamo accedere sia agli attributi che ai dati del file)
- Altrimenti, l'elemento contiene più puntatori a cluster del disco che contengono dati del file, ed eventualmente almeno un puntatore ad un altro blocco di puntatori (si usa di fatto una allocazione indicizzata a schema concatenato)

11.5 Gestione dello spazio libero

- Il SO deve tenere anche traccia di tutti i blocchi liberi del disco.
- Una opportuna struttura dati del SO memorizzata permanentemente sull'HD permette di risalire ai blocchi liberi del disco (in Unix, queste informazioni sono memorizzate nel **superblocco**, contenuto in alcuni blocchi iniziali dell'HD, a partire dal blocco numero 1. In Windows XP queste informazioni stanno nella MFT.)
- Per creare o estendere un file, si cercano blocchi liberi nell'elenco e si allocano al file
- Quando un file viene cancellato i suoi blocchi sono reinseriti nell'elenco dei blocchi liberi

11.5.1 Vettore di bit

- usiamo un bit per ogni blocco. Il valore del bit indica se il blocco è occupato o libero

bit[*i*] = 1 ⇒ *i*-esimo blocco libero
0 ⇒ *i*-esimo blocco occupato

- E' il sistema usato da MAC/OS

11.5.1 Vettore di bit

- Naturalmente il vettore di bit richiede spazio aggiuntivo. Ad esempio:

  block size = 2^9 bytes

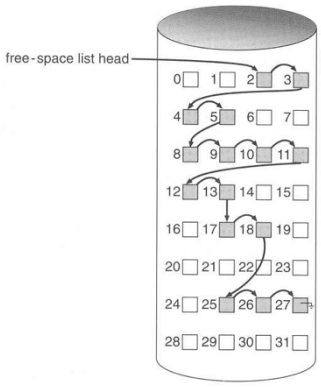
  disk size = 2^{35} bytes (32 gigabyte)

  $n = 2^{35}/2^9 = 2^{26}$ bit (ossia 8 megabyte)

- Perchè la gestione sia efficiente, l'intero vettore deve essere sempre tenuto in MP (e periodicamente salvato in MS)
- L'uso di cluster diminuisce lo spazio occupato: è sufficiente usare un bit per ogni cluser.

11.5.2 Lista concatenata

- Formare con i blocchi liberi una lista: ciascun blocco punta al successivo blocco libero
- Nessuno spreco di spazio
- la FAT può essere usata anche per gestire la lista dei blocchi liberi, altrimenti la ricerca di molti blocchi liberi sarebbe troppo inefficiente (fig. 11.10)



Altri metodi

- 11.5.3 Raggruppamento
  - Raggruppare in un blocco più puntatori a blocchi liberi. L'ultimo puntatore punta ad un altro blocco di blocchi liberi, e così via
  - Con questo metodo è facile trovare in fretta molti blocchi liberi
- 11.5.4 Conteggio
  - mantenere il numero di un blocco e quanti blocchi consecutivi liberi lo seguono
  - è un metodo simile alla lista ma con meno entry

11.6 Prestazioni ed efficienza

- A parte le osservazioni già fatte sulle prestazioni offerte dai vai metodi di allocazione dello spazio, l'efficienza nell'uso del FS dipende pesantemente dal fatto che il SO implementi un sistema di caching in MP dei file (e dei relativi attributi) usati più di frequente e di recente, ed in particolare di tutti i file aperti.
- In questo modo, tutte le operazioni sui file e l'aggiornamento dei loro attributi viene fatto accedendo solo alla copia in RAM dei dati del file e dei suoi attributi.
- Sarà il SO ad occuparsi di mantenere la consistenza con le corrispondenti informazioni in Memoria Secondaria, salvando periodicamente le informazioni in RAM sul disco