SISTEMI OPERATIVI

ESERCIZIO 3 (9 punti)

a) In un sistema i blocchi su disco occupano 512 byte, e un puntatore a blocco è scritto su 4 byte. Il sistema operativo adotta l'allocazione indicizzata. Quanti accessi al disco sono necessari per leggere il byte numero 100.000, assumendo che gli attributi del file in questione siano già in memoria primaria? (motivate la vostra risposta, specificando le eventuali assunzioni che fate).

3. In un blocco indice possiamo scrivere 512/4 = 128 puntatori, indirizzando così 64Kbyte di dati, per cui un solo blocco indice non è sufficiente ad indirizzare il blocco contenente il byte specificato. Sia usando l'allocazione indicizzata gerarchica che l'allocazione indicizzata concatenata, un secondo blocco indice è sufficiente per indirizzare il blocco contenente il byte specificato. Sono quindi necessari due accessi al disco per leggere i due blocchi indice + un accesso per leggere il blocco contenente il byte numero 100.000.

b) Quali sono i vantaggi e gli svantaggi dell'allocazione indicizzata?

Non c'è bisogno di usare blocchi adiacenti del disco, accesso diretto efficiente, un intero blocco indice quasi completamente "sprecato" nel caso di file piccoli

c) In un dato momento, lo spazio occupato sul disco da una cartella MYDIR è di X byte. Dopo un po' di tempo, lo spazio occupato dalla cartella MYDIR aumenta, ma la quantità totale di spazio occupato sul disco è diminuita. Come è possibile spiegare questa situazione? (si assuma per semplicità che vi sia un solo utente nel sistema, e che l'utente stia operando solo nella cartella MYDIR)

L'utente ha cancellato pochi file molto grossi da MYDIR, e ha inserito in MYDIR molti file piccoli la cui dimensione totale è inferiore alla dimensione dei file cancellati.

d) Cosa succede quando si esegue la *open* di un file, in un generico sistema operativo?

Si comunica al sistema che si vuole usare quel file: il SO, dopo aver controllato i diritti di accesso, recupera gli attributi del file e li copia in RAM, nella open file table. Successivi accessi agli attributi del file da parte del programma che ha chiamato la open useranno la copia degli attributi in RAM, anziché la copia su disco (ad accesso molto più lento). Di solito, anche una parte del file stesso viene copiata in RAM, in modo da velocizzare l'accesso ai dati del file. La open restituisce un *file pointer* che verrà usato dal programma per accedere ai vari dati/attributi del file disponibili in RAM.

ESERCIZIO 3 (4 punti)

Un hard disk ha la capienza di 2^{38} byte, ed è formattato in blocchi da 512 byte.

a) Quanti accessi al disco sono necessari per leggere l'ultimo blocco di un file A della dimensione di 4096 byte, assumendo che sia già in RAM il numero del primo blocco del file stesso e che venga adottata una allocazione concatenata dello spazio su disco? (motivate la vostra risposta)

9. Ogni blocco infatti memorizza 508 byte di dati più 4 byte di puntatore al blocco successivo (infatti, $2^{38}/2^9 = 2^{29}$), per cui sono necessari 9 blocchi per memorizzare l'intero file.

b) Qual è lo spreco di memoria dovuto alla frammentazione interna nella memorizzazione di A (motivate la risposta)?

L'hard disk è suddiviso in $2^{38}/2^9 = 2^{29}$ blocchi, sono necessari 4 byte per memorizzare un puntatore al blocco successivo, e ogni blocco contiene 508 byte di dati. Il nono blocco memorizzerà quindi 32 byte del file, e la frammentazione interna corrisponde a 512 – 32 = 480 byte (476 se si considerano non sprecati i 4 byte del nono blocco che contengono il puntatore, non utilizzato, al blocco successivo)

c) Se si adottasse una allocazione indicizzata dello spazio su disco, quanti accessi al disco sarebbero necessari per leggere l'ultimo byte di un file B grande 100k byte (specificate quali assunzioni fate nel rispondere a questa domanda e motivate la vostra risposta)?

Poiché sono necessari 4 byte per scrivere il numero di un blocco, in un blocco indice possono essere memorizzati 128 puntatori a blocco, e con un blocco indice possiamo indirizzare in tutto $2^7 * 2^9 = 2^{16}$ = 64k byte. Un solo blocco indice non è quindi sufficiente a memorizzare B. Assumendo una allocazione indicizzata concatenata (ma si ottengono gli stessi risultati con una allocazione indicizzata gerarchica) usando un secondo blocco indice è possibile memorizzare l'intero file. Se il numero del primo blocco indice è già in RAM, per leggere l'ultimo carattere del file sono necessari 3 accessi al disco: lettura del primo blocco indice, lettura del secondo blocco indice, lettura dell'ultimo blocco del file.

ESERCIZIO 3 (4 punti)

a) In quale caso l'uso dell'allocazione indicizzata dello spazio su disco risulta particolarmente svantaggiosa in termini di spazio su disco sprecato?

Nel caso di file piccoli, poiché quasi tutto il blocco indice viene sprecato.

b) L'uso di *cluster di blocchi adiacenti* (un po' come visto nel caso dell'allocazione concatenata) acuisce o rende meno grave il problema di cui si parla al punto a)?

Lo acuisce, in quanto il sistema lavora in sostanza con blocchi più grandi, e quindi lo spreco di spazio per memorizzare i file piccoli è proporzionalmente maggiore.

c) Utilizzando opportuni disegni descrivete le diverse forme di allocazione indicizzata viste a lezione: indicizzata *a schema concatenato*, indicizzata *a più livelli*, *variante adottata nei sistemi unix*.

Si vedano i lucidi della sezione 14.4.3

d) Se in un sistema unix i blocchi sono da 1024 byte e il numero di un blocco è scritto su 4 byte, qual è la dimensione massima di un file memorizzabile nel sistema (è sufficiente riportare l'espressione da usare per il calcolo)

Un blocco da 1024 byte può contenere 1024/4 = 256 puntatori a blocco. Si ha quindi:
(10 * 1024) + (256 * 1024) + (256 ^{2} * 1024) + (256 ^{3} * 1024) byte

e) se in un systema Unix l'hard disk fosse grande 2^40 byte, e suddiviso in blocchi da 2^9 byte, quale sarebbe la dimensione massima di un file memorizzabile nel sistema?

l'hard disk è suddiviso in 2^40/2^9 = 2^31 blocchi, e sono necessari 4 byte per scrivere il numero di un blocco. In un blocco indice possono quindi essere ospitati 512/4 = 128 puntatori a blocco, e la dimensione massima di un file memorizzabile nel sistema sarà quindi:

(10 * 512) + (128 * 512) + (128 ^{2} * 512) + (128 ^{3} * 512) byte

e se i blocchi dell'hard disk fossero invece da 4096 byte?

2^40/^12 = 2^28 blocchi. Usiamo 4 byte per scrivere in numero di un blocco, e in un blocco indice ci stanno 4096/4 = 1024 puntatori a blocco. E quindi:

(10 * 4096) + (1024 * 4096) + (1024 ^{2} * 4096) + (1024 ^{3} * 4096) byte

ESERCIZIO 3 (9 punti)

In un sistema si sa che la quasi totalità dei file (diciamo il 99,99% dei file) hanno una dimensione molto superiore a quella di un blocco dell'hard disk, e devono poter essere acceduti in modo diretto

a) Quali metodi di allocazione dello spazio su disco non verranno sicuramente adottati per quel particolare sistema, e perché?

L'allocazione contigua, per la difficoltà di trovare spazio contiguo sufficiente, e l'allocazione concatenata, a causa del tempo di accesso proporzionale alla lunghezza del file.

b) Si supponga che il sistema usi una allocazione dello spazio su disco simile a quella dei sistemi unix, usando blocchi da 1024 byte e 32 bit per scrivere il numero di un blocco. Quanti accessi su disco sono necessari per leggere il byte numero 200.000 di un file, supponendo già presente in memoria principale il numero dell'i-node associato al file? (motivate la vostra risposta)

3. Innanzitutto, un accesso in lettura è necessario per leggere l'i-node del file. Poiché il file è lungo almeno 200.000 byte, i primi 10 puntatori a blocco dell'i-node non sono sufficienti ad indirizzare tutti i blocchi del file, ed è necessario usare il puntatore di indirezione semplice. Infatti un blocco può contenere 256 puntatori a blocco, tra i quali vi sarà anche il blocco che contiene il byte numero 200.000. Altre due letture sono quindi necessarie per leggere prima il blocco di indirezione semplice e poi il blocco contenente il byte 200.000.

c) spiegate la differenza tra un pathname relativo e il pathname assoluto di un file

Un pathname relativo è sempre relativo ad una directory diversa dalla radice del file system, e non inizia mai con il simbolo "/", mentre il pathame assoluto di un file inizia sempre dalla radice del file system.

d) commentate questa la seguente asserzione: *se viene perso il pathname assoluto di un file non è più possibile accedere ai dati del file*

L'asserzione è senza senso, in quanto normalmente il pathname di un file non è memorizzato da nessuna parte

e) Dite quale configurazione RAID è opportuno scegliere per le seguenti situazioni:

a. Il sistema deve essere estremamente veloce nell'accesso ai dati e sfruttare al massimo lo spazio di memorizzazione disponibile, mentre l'affidabilità non è particolarmente importante

b. Il sistema deve garantire una ragionevole affidabilità ed efficienza nell'accesso ai dati, sfruttando al meglio lo spazio disponibile:

c. Il sistema deve garantire la massima affidabilità

a: RAID 0; **b: RAID 5;** **c: RAID 01 (o 10)**

ESERCIZIO 3 (9 punti)

a) Descrivete con un semplice esempio la variante FAT dell'allocazione concatenata.

si può ipotizzare un hard disk giocattolo fatto, ad esempio da 12 blocchi, e un file memorizzato, nell'ordine, nei blocchi 7, 3, 11, 6. La FAT sarebbe allora la seguente: (0 = blocco non utilizzato, -1 = blocco di fine file)

0	0	0	11	0	0	-1	3	0	0	0	6
0	1	2	3	4	5	6	7	8	9	10	11

b) Che svantaggi ha l'uso della FAT?

Per essere efficiente la FAT va tenuta in RAM, sottraendo così spazio ai processi. Periodicamente deve essere salvata sull'hard disk. Se viene persa la FAT si perdono in un colpo solo tutti i file memorizzati sull'hard disk.

c) Se un hard disk è grande 16 Giga byte con blocchi da 1 K byte, quanto spazio occupa la FAT di quel disco? (esplicitate i calcoli che fate)

Il disco è grande 2^{34} byte, per cui la FAT deve avere 2^{34} / 2^{10} = 2^{24} entry. Per memorizzare il numero di un blocco in una entry sono quindi necessari 3 byte, e la FAT occupa 3 * 2^{24} = 48 megabyte

ESERCIZIO 3 (6 PUNTI)

a) nel contesto del sistema operativo Unix, che cosa è un link counter, e che cosa ci dice il suo valore?

è un campo nell'index-node di un file che conta il numero di hard link al file correntemente presenti nel file system in cui risiede il file.

b) Indicate almeno un comando Unix e una system call unix che possono modificare il valore di un link counter. Cosa succede quando un link counter arriva al valore 0 (zero)?

possibili comandi che modificano il valore corrente di un link counter sono "ln", "rm", possibili system call sono "link" e "unlink". Viene rimosso l'index node e il file con tutti i suoi dati.

c) Sia A un link fisico al file X, e sia B un link simbolico al file X. È più veloce l'accesso ai dati di X usando A o usando B? Perché? Quale dei due link produce maggior occupazione di spazio sull'hard disk, e perché?

è più veloce l'accesso ai dati di X usando A, ossia il link fisico, perché usando il link simbolico B è necessario leggere un index-node in più quello usato per implementare il link simbolico (il che costringerà, nel caso generale, a una lettura in più sull'hard disk per leggere l'index node del link simbolico B).

Il link simbolico produce una maggior occupazione di spazio, proprio perché richiede l'allocazione di un index-node in più, quello che contiene le informazioni sul link simbolico B.

ESERCIZIO 3 (6 punti)

Sia dato un sistema operativo Unix, in cui viene eseguito correttamente il comando: "mkdir /users/gunetti/myfiles/pippo" (dove "pippo" non esisteva prima dell'esecuzione del comando)

a) cosa succede dal punto di vista degli index-node coinvolti nel comando?

il link counter dell'index-node associato all'hard link "myfiles" viene incrementato di 1. Il link counter dell'index-node appena allocato e associato all'hard link "pippo" viene inizializzato a 2.