Sistemi Operativi
AA 2022/23
Esercitazione

May 10, 2023

Esercizio 1

Si consideri un file consistente di 100 blocchi e che il suo *File Control Block* (FCB) sia gia' in memoria. Siano dati due file-systems gestiti rispettivamente tramite allocazione a lista concatenata (*Linked List Allocation*) e allocazione indicizzata (*Indexed Allocation*). Si assuma che nel caso indicizzato il FCB sia in grado di contenere i primi 200 blocchi del file.

Domanda Si calcolino le operazioni di I/O su disco necessarie per eseguire le seguenti azioni in entrambi i file-systems:

1. Aggiunta di un blocco all'inizio del file
2. Aggiunta di un blocco a meta' del file
3. Aggiunta di un blocco alla fine del file

Soluzione Nel caso di file-system con *Indexed Allocation*, ogni file conterra' un index block, ovvero un blocco contenente i puntatori a tutti gli altri blocchi componenti il file. Cio' implica che per raggiungere un dato blocco bastera' effettuare una semplice indicizzazione. Nell'implementazione con *Linked List Allocation* invece, ogni blocco contiene il riferimento al precedente ed al successivo. In questo caso, per effettuare operazioni che non siano all'inizio del file bisognera' scorrere la lista fino al blocco desiderato ed in seguito eseguire l'operazione necessaria.

Supponendo che il FileControlBlock sia già stato caricato in memoria, avremo:

1. Linked Allocation: 1 `IO-ops`; Indexed Allocation: 1 `IO-ops`
2. Linked Allocation: 52 `IO-ops`; Indexed Allocation: 1 `IO-ops`
3. Linked Allocation: 3 `IO-ops`; Indexed Allocation: 1 `IO-ops`

Per quanto riguarda Indexed Allocation, abbiamo soltanto una operazione di IO per scrivere il blocco che aggiungiamo. Per quanto riguarda Linked Allocation, quando il blocco è all'inizio del file dobbiamo fare una operazione di IO per accedervi in scrittura. Quando il blocco da aggiungere è a metà del file, 50 operazioni di IO vengono effettuate per leggere la prima metà dei blocchi del file, un'altra operazione è necessaria per accedere in scrittura al cinquantesimo blocco di cui cambiamo il successore, un'ultima viene fatta per scrivere il blocco aggiunto: il totale consta quindi di 52 operazioni di IO. Quando il blocco da aggiungere è alla fine del file, ammesso di avere il puntatore all'ultimo elemento, vi accediamo in lettura e in scrittura per modificare il puntatore al prossimo, quindi effettuiamo un'altra operazione di IO per scrivere il blocco aggiunto, totalizzando così 3 operazioni di IO.

Se invece assumiamo che il FileControlBlock non sia in memoria, allora occorre considerare due operazioni di IO in aggiunta a quelle dello schema precedente: una per caricarlo in memoria, una per scriverlo con la size aggiornata. In questo caso avremo quindi:

1. Linked Allocation: 3 `IO-ops`; Indexed Allocation: 3 `IO-ops`

2. Linked Allocation: 54 `IO-ops`; Indexed Allocation: 3 `IO-ops`

3. Linked Allocation: 5 `IO-ops`; Indexed Allocation: 3 `IO-ops`

A prescindere dal dettaglio del calcolo, che come abbiamo visto può dipendere dalle assunzioni fatte, è importante che sia chiaro il vantaggio della implementazione Indexed Allocation in caso di accessi scattered a blocchi di un file. Il numero di operazioni di IO è in questo caso indipendente dalla posizione del blocco all'interno del file, dal momento che non c'è bisogno di scorrerlo per intero fino al punto desiderato.

Esercizio 2

Che relazione c'è tra un *File Descriptor* ed una entry nella tabella globale dei file aperti del file system?

Soluzione Per ogni istanza del file aperta da un processo e identificata dal descrittore del file, una struct del tipo OpenFileRef viene aggiunta ad una lista accessibile nel PCB. Structs associate allo stesso file puntano alla stessa entry nella tabella globale dei file aperti, in cui abbiamo una struct del tipo OpenFileInfo per ogni file aperto.

Riassumendo, mentre possono esistere più descrittori di file per lo stesso file aperto, una e una sola entry ad esso associata esiste nella lista globale dei file aperti, alla quale tutte le OpenFileRef nella lista presente all'interno del PCB puntano.

Esercizio 3

Cos'e' un *File Control Block* (FCB)? Cosa contiene al suo interno?

Soluzione Il FCB e' una struttura dati che contiene tutte le informazioni relative al file a cui e' associato. Esempi di informazioni possono essere: permessi, dimensione, data di creazione, numero di inode (se esiste), ecc. . Inoltre il FCB contiene informazione sulla locazione sul disco dei dati del file - ad esempio in un FS con allocazione concatenata il puntatore al primo blocco del file. In Figura 1 e' riportata una illustrazione di tale struttura.

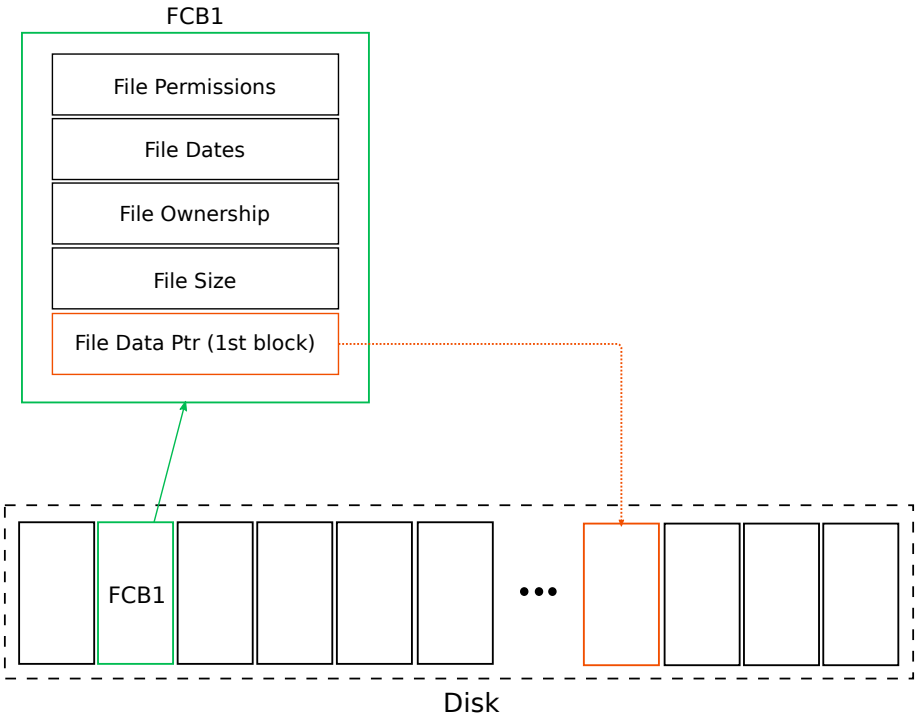

Figure 1: Esempio di FCB.La struttura contiene tutti gli attributi del file compresa la locazione dei dati - qui rappresentato dal puntatore al primo blocco della lista contente i dati, supponendo un FS con linked allocation.

Esercizio 4

Si consideri l'implementazione di un file system con allocazione concatenata (Linked Allocation) ed un file system che invece utilizzi una allocazione ad indice (Indexed Allocation).

Domanda Illustrare brevemente i vantaggi dell'uno e dell'altro nell'eseguire le seguenti operazioni:

1. accesso sequenziale
2. accesso indicizzato
3. operazioni su file di testo

Soluzione

1. **Accesso Sequenziale**: in questo caso, il file system che usa la *lista concatenata* sarà favorito, garantendo una maggiore velocità dell'operazione. Ciò poiché non bisogna effettuare alcuna ricerca per trovare il blocco successivo, poiché esso sarà semplicemente il blocco `next` nella lista.
2. **Accesso Indicizzato**: questa operazione - contrariamente alla precedente - risulta essere molto onerosa per il file system che usa la *linked list*. Infatti, per ogni accesso, bisognerà scorrere tutta la lista finchè non viene trovato il blocco desiderato. La ricerca tramite `inode` risulterà molto più efficiente.
3. **Accesso su file di testo**: per la natura del tipo di file (testo), la *linked list* risulterà più efficiente ancora una volta. Questo poiché i file di testo sono memorizzati in maniera sequenziale sul disco, riportandoci al caso 1.