SAPIENZA
UNIVERSITÀ DI ROMA

REPORT PROGETTO SISTEMI OPERATIVI

Progetto scelto:

Simple file-system

Componenti gruppo:

Manuel Ivagnes 1698903
Valerio Coretti 1635747

Anno di corso
2017/2018

Premessa

Il progetto scelto pone come obiettivo la realizzazione di un semplice file-system implementato tramite l'utilizzo di un file binario che sostituisce il disco.
Trattandosi di un piccolo progetto universitario con solo scopo di apprendimento, invece di optare per una mera divisione dei compiti, l'intero lavoro è stato svolto cercando un confronto continuo tra i componenti del gruppo per lo sviluppo di ogni elemento.
Questo ci ha portato alla scrittura di un codice più esteso ma facilmente leggibile anche grazie all'utilizzo di numerosi commenti.
Abbiamo inoltre cercato di riportare quanto più possibile quanto appreso durante il corso di laurea riutilizzando ed estendendo metodi precedentemente acquisiti e non limitandoci esclusivamente a questo corso.

Per il controllo del corretto funzionamento sono stati realizzati test per ogni componente ed è stata realizzata una semplice Shell di prova che fornisce una comoda interfaccia interattiva arricchita dall'utilizzo di un format di colori personalizzati.

Il progetto

Il file-system è un componente del sistema operativo che fornisce i meccanismi di accesso e memorizzazione dei dati, consentendo una visione logica uniforme di tutte le informazioni memorizzate in modo permanente sul dispositivo.
Permette inoltre di astrarre dalle caratteristiche fisiche dei supporti di memorizzazione.

In questa semplice implementazione si utilizza una struttura ad albero:

1. Come primo elemento si ha una directory root nella quale sono contenuti file e/o altre directory.
2. Ogni directory può contenere a sua volta altri file e/o altre directory.

Il disco sottostante è simulato tramite un file diviso in blocchi di dimensione fissa gestiti da una bitmap, ovvero un vettore di byte nel quale ogni bit indica se il blocco corrispondente è già allocato (1) oppure no (0).

La divisione del progetto permette comunque modularità alle funzioni che possono essere facilmente riadattate per l'utilizzo su un disco reale.

La realizzazione

Come prima cosa abbiamo realizzato le funzioni per il controllo ed il set dei bit nella bitmap, il processore infatti non può elaborarli direttamente ma necessita di una manipolazione dei byte per l'accesso ad essi.
Le funzioni sono realizzate tramite un sistema di shift e maschere che permettono appunto la trasformazione del byte e l'estrazione del solo bit necessario.

Nella get, una volta preso il byte richiesto, questo viene passato a getBit che lo shifta fino ad avere come bit meno significativo il corrispondente all'offset desiderato, il resto viene occultato da una maschera e si ha quindi 1 oppure 0.

Nella set si crea una maschera shiftando un bit 1 fino all'offset richiesto, poi:
- Nel caso in cui si voglia impostare il bit ad 1, la maschera viene messa in OR con il byte, quindi sia che il bit sia 1 o 0 il bit risulterà impostato ad 1
- Nel caso in cui si voglia 0 si fa un AND con !1 che restituisce sempre 0

Dopodiché ci siamo dedicati al disck_driver che simula il disco tramite un file binario, ad ogni inizializzazione del file-system si richiama DisckDriver_init che permette di aprire un file già esistente o crearne uno nuovo andando a inizializzare l'header del disco e la bitmap.
L'header contiene le informazioni principali sullo stato generale dei blocchi nel disco e di conseguenza sulla bitmap.

In questo caso inoltre abbiamo deciso di segnalare gli errori hardware e quelli generati dall'utilizzo di parametri sbagliati nel file system tramite un Error_helper, che invece di ritornare un semplice errore chiude il programma tramite exit e relativo messaggio d'errore.

Per le funzioni read_block e write_block abbiamo deciso di usare le syscall read e write, inoltre, per evitare errori dati da interrupt durante la scrittura (o lettura) abbiamo deciso di inserirle in cicli di controllo, che catturano le eccezioni e permettono di continuare le operazioni normalmente.

L'offset delle seek è impostato in modo da calcolare implicitamente anche la grandezza dell'header e della bitmap, in questo modo il blocco 0 corrisponde sempre alla root e non bisogna aggiungere altri calcoli in fs.

Abbiamo inoltre aggiunto due funzioni supplementari:
I. updateBlock che permette di semplificare la scrittura del fs rimuovendo la necessità di usare la free prima di ogni write su blocchi da sovrascrivere. Questa infatti, al contrario della write, non controlla o esegue side-effect sulla bitmap.
II. extremeFreeBlock che oltre a modificare la bitmap inizializza tutto il blocco rendendo impossibile il recupero dei dati

Per semplicità nella scrittura del file system abbiamo accorpato la flush, la quale però è comunque disponibile.

Il file-system in se viene realizzato utilizzando una serie di catene di blocchi collegati, ad ogni elemento viene associato un numero di blocco corrispondente tramite il quale è possibile recuperarlo.
Gli elementi possono essere di 4 tipi:

- firstDirectoryBlock => è il primo blocco della directory, contiene header, fileControlBlock, il numero di file/directory presenti e la prima parte dei numeri corrispondenti ai first-blocchi dei file/directory che contiene
- directoryBlock => altri blocchi, contengono header e altri file/directory
- firstFileBlock => è il primo blocco del file, contiene header, fileControlBlock e la prima parte dei numeri corrispondenti ai blocchi aventi il continuo del file
- fileBlock => altri blocchi, contengono header ed altri dati

Ogni blocco ha quindi un header che contiene le informazioni sulla posizione del file/directory nella catena corrispondente.
Se si ha firstDirectoryBlock o firstFileBlock allora si ha anche il fileControlBlock, ovvero una struttura contente tutte le informazioni sul file/directory, quali blocco corrispondente, nome, … .

Per indicare la non presenza di un blocco collegato abbiamo usato il valore -1.
Inoltre nelle funzioni che controllano la presenza di file/directory il valore è considerato valido solo se >0 per evitare conflitti con la root.
Sia directory che file vengono infatti sempre inizializzati a 0 durante la creazione per evitare collegamenti errati dati da memoria sporca.

Anche in questo caso abbiamo aggiunto una nuova funzione per la formattazione completa, ExtremeFormat, che che oltre a modificare la bitmap inizializza tutti i blocchi rendendo impossibile il recupero dei dati.

Cercando una gestione più efficace dei file ripartiti su più blocchi abbiamo inserito nel fcb la variabile written_bytes[1] che mantiene l'ultima posizione scritta e permette di ottimizzare il controllo degli errori sulle size di write, read e seek.

Non avendoli mai utilizzati nelle funzioni abbiamo deciso di rimuovere i valori pos_in_dir e current_block nel DirectoryHandle ed i valori size_in_bytes e size_in_block nel fileControlBlock, ottenendo una ottimizzazione degli spazi.

Per la spiegazione delle singole funzioni è possibile visionare i commenti.
Per semplificare la compilazione dei test abbiamo aggiunto un makefile con le seguenti opzioni:

- shell
- simplefs_test
- test_disk_driver
- test_bitmap

[1] Permette inoltre di mantenere il valore anche dopo la chiusura e riapertura della shell