

**PROGETTO SISTEMI OPERATIVI 2017-2018**

**FILE SYSTEM**

**ALUNNI:**

Simone Staffa 1710426

Alessio Cesaretti 1712254

**Il progetto**

Un file system è l’insieme delle strutture dati e metodi che permettono la registrazione e l’accesso ai dati e programmi presenti in un sistema di calcolo, facilitando quindi la gestione dei file e delle cartelle.

I file system possono essere rappresentati sia testualmente che graficamente tramite browser di file e Shell, nel nostro caso trattiamo la prima tipologia di rappresentazione.

Questa implementazione utilizza una struttura gerarchica ad albero in cui abbiamo una *root directory* (“/”), che contiene tutti gli altri eventuali file e directory. A sua volta ogni directory può contenere altri file e/o altre directory.

Il disco sottostante è simulato e gestito attraverso un file diviso in blocchi gestiti tramite una *bitmap*. Una *bitmap* è un vettore di byte nel quale viene mantenuta traccia dello stato dei blocchi, settando il bit corrispondente a 0 o 1: “0” se il blocco è libero, “1” se il blocco è occupato.

**Lo sviluppo**

Nella realizzazione del progetto abbiamo ritenuto opportuno lavorare in coppia e consultarci continuamente, piuttosto che suddividerci i compiti, in modo da poter correggere più facilmente gli eventuali errori. Per mantenere un certo ordine logico e per facilitare la leggibilità del codice, ogni funzione è stata commentata.

L’implementazione dei file è stata svolta secondo il seguente ordine per facilitare il progredire del progetto:

1. *bitmap.c*
2. *disk\_driver.c*
3. *simplefs.c*
4. *file\_system\_test.c*

Come prima cosa abbiamo realizzato le funzioni per la gestione dei bit all’interno della *bitmap*. Dovendo lavorare bit a bit dentro un singolo byte, abbiamo utilizzato maschere e shift. Siccome non è possibile lavorare sui bit singolarmente abbiamo fatto ricorso all’utilizzo di variabili char, spostando i bit con l’operatore di shift e usando delle maschere per effettuare le operazioni di manipolazione.

Le semplici funzioni get e set ci sono state utili poi per implementare il disco simulato con cui lavorare. Con la funzione *DiskDriver\_init* infatti viene creato il file binario di supporto ( se non già esistente) inizializzando l’header del disco e la sua bitmap. Le funzioni di lettura e scrittura dei blocchi sono state realizzate mediante le syscall “*read”* e “*write”*, all’interno di cicli di controllo per gestire sia il numero di byte da leggere/scrivere, sia gli errori e in caso opportuno decidere di interrompere l’operazione o proseguire. Per ultimo abbiamo implementato il file system vero e proprio, appoggiandoci alle funzioni del disco per applicare le operazioni sul nostro file di testo.

Ogni blocco (file o directory che sia) viene identificato con un **FileControlBlock** che mantiene informazioni di stato quali nome, tipo, dimensioni (in bytes e in numero di blocchi) e posizione sul disco.

Il file system è implementato come una serie di catene di blocchi collegati.

Per facilitarne la gestione, distinguiamo 4 tipi di elementi all’interno di una catena:

* ***FirstDirectoryBlock***: è il primo blocco della directory e contiene: un header, utile per tenere traccia della posizione del blocco e dei suoi successivi; un *FileControlBlock*; un intero indicante il numero di blocchi (file/directory) presenti; un array che contiene la prima parte dei blocchi;
* ***FirstFileBlock***: è il primo blocco del file, contiene le stesse strutture del primo blocco delle directory.
* ***DirectoryBlock***: gli altri blocchi successivi al FirstDirectoryBlock, contengono header e dati dei blocchi.
* ***FileBlock***: gli altri blocchi successivi al FirstFileBlock, contengono header e dati del file.

Inoltre per tenere traccia, all’interno del file system, dei file e delle directory aperte, vengono utilizzate rispettivamente due strutture: **FileHandle** e **DirectoryHandle**.

Al termine di ogni file relativi alla bitmap/disk\_driver/simplefs, prima di procedere con il successivo, abbiamo creato dei test per validare il lavoro svolto così da avanzare senza problemi. Per ogni file quindi vi è un relativo file di test, che prova il corretto funzionamento di ogni singola funzione.

**Problematiche**

Le principali problematiche riscontrate sono state:

* L’implementazione delle funzioni della bitmap, poiché la manipolazione bit a bit non è nativamente supportata, se non mediante l’utilizzo di maschere e operazioni di shift.
* La comprensione delle varie struct e relativo utilizzo.
* La comprensione del meccanismo di astrazione dal disco.

**Risultato finale**

E’ stato prodotto anche un apposito *makefile* con il quale viene semplificata la compilazione, basterà quindi digitare da terminale il comando “*make fs”* e verrà generato un eseguibile “*file\_system”*. E’ disponibile anche un comando di reset “*make clean”* per cancellare l’eseguibile e il file binario associato al file system.

L’utilizzo finale del progetto, per provarne il corretto funzionamento è facilitato da un’interfaccia su shell che permette di gestire un vero e proprio file system, da terminale, salvando le modifiche sul file di testo relativo al disco simulato. In particolare possiamo distinguere 8 operazioni effettuabili:

1. **Creazione di un file:** inserendo da tastiera un nome con cui distinguerlo dagli altri
2. **Scrittura su file:** inserendo da tastiera il file su cui si desidera scrivere, per poi inserire il testo da applicare
3. **Lettura file:** inserendo da tastiera il file che si desidera leggere
4. **Eliminazione file/directory**: inserendo da tastiera il file/directory che si desidera eliminare
5. **Creazione directory:** inserendo da tastiera un nome per la nuova directory
6. **Visualizzazione contenuto directory**: per visualizzare i file/directory (opportunamente distinti) presenti nel percorso corrente
7. **Cambio directory:** inserendo da tastiera tra le directory disponibili o “..” per salire di un livello nell’albero

0. **Esci:** interrompe il processo