SAPIENZA
UNIVERSITÀ DI ROMA

PROGETTO SISTEMI OPERATIVI 2017-2018

FILE SYSTEM

ALUNNO:

Simone Silvestri 1699926

Il progetto:

Il progetto consiste nello sviluppo di un File System, cioè l'insieme di metodi e strutture dati necessarie al collezionamento e all'accesso ai dati e programmi su un sistema di calcolo, facilitando all'utente l'interazione con file e cartelle.

Il File System sviluppato è di tipo testuale, cioè l'interazione con l'utente avviene attraverso l'utilizzo della **shell** (terminale).

L'implementazione effettuata utilizza uno struttura dati ad albero in cui abbiamo una **root directory** ("/") che contiene tutti gli eventuali file o directory presenti nel File System.

Il disco che è alla base del File System è simulato e sostituito da un file .txt mappato con una **BitMap**, essa è un vettore di Byte nel quale viene mantenuta traccia dei blocchi tramite il flag corrispondente: se li blocco è libero il flag corrispondente è settato a '1', differentemente se il blocco è occupato il flag corrispondente sarà settato a '0'.

Lo sviluppo:

Lavorando da solo ed essendo un progetto complesso, lo sviluppo è stato finalizzato con burst ad intervalli regolari (principalmente inizio e conclusione). Per aiutarmi nella comprensione del codice ho utilizzato molto i commenti, oltre alla libreria di linux per alcune syscall utilizzate durante la scrittura delle funzioni. L'implementazione passo passo delle funzioni ha seguito un certo ordine logico, necessario per arrivare al risultato finale.

L'ordine con cui ho completato l'implementazione dei file è il seguente:
- *bitmap.c*
- *disk_driver.c*
- *simplefs.c*
- *simplefs_test.c*
- *file_system.c*

La prima implementazione appunto è stata quella della BitMap e le funzioni per la gestione dei bit al suo interno. Per poter "sporcare" il bit (segnare se il bit è libero o occupato) all'interno della BitMap, sono state utilizzati **shift logic**i e **maschere**. Questo è stato possibile utilizzando variabili di tipo char, spostando i bit con gli shift e manipolandoli con le maschere opportune. Utilizzando queste tecniche, sono state implementate le funzione BitMap_get e BitMap_set utili a simulare il disco attraverso il file .txt .
Fatto questo, è stato possibile simulare il disco attraverso l'operazione Disk_Driver_init, che crea il file binario di supporto (se non già esistente)

inizializzando la sua BitMap e l'header del disco. Per riuscire a scrivere e leggere su disco sono state utilizzate le syscall "read" e "write", in modo da poter tenere traccia sia dei byte in lettura/scrittura, sia gli eventuali errori ed operazioni di controllo. Infine, è stato implementato il file system vero e proprio, inoltre sono state testate tutte quante le funzioni necessarie di *bitmap.c* e *disk_driver.c* .

Ogni blocco (sia file sia directory) all'interno del nostro File System viene identificato come *FileControlBlock*, una struttura dati che tiene conto di tutte le informazioni di stato relative al blocco (quali nome, dimensioni ecc.). Il File System vero proprio quindi è una serie a catena di blocchi collegati; possiamo distinguere i blocchi in 4 tipi:

- **FirstDirectoryBlock:** è il primo blocco della directory e contiene: un header con all'interno tutte le informazioni necessarie per tenere traccia del blocco dei suoi successivi; un array che contiene la prima parte dei blocchi; un *FileControlBlock;* un intero con il numero di entry all'interno della directory.

- **DirectoryBlock:** contiene i blocchi successivi contenuti nel *FirstDirectoryBlock*, compresi di header e dati associati.

- **FirstFileBlock:** è il primo blocco del file, contiene al suo interno le stesse strutture del *FirstDirectoryBlock.*

- **FileBlock:** contiene i blocchi successivi contenuti nel *FirstFileBlock,* compresi di header e dati associati.

Infine, per tenere traccia di ogni file o directory aperte all'interno del File System, sono utilizzate le struct **FileHandle** e **DirectoryHandle.**
Durante lo sviluppo, mi è stato indispensabile testare le funzioni prima di utilizzarle nel File System, dunque ho preparato un file *simplefs_test.c* con al suo interno i test delle funzioni in *bitmap.c* e *disk_driver.c* . Le funzioni presenti nel file *simplefs.c* invece, sono testate direttamente all'interno del file per la shell, *file_system.c* .

Problemi riscontrati:

Lavorando in singolo, le problematiche riscontrate sono state le seguenti:
- L'utilizzo di alcune **syscall** (*lseek*, *posix_fallocate)* e la manipolazione dei bit sul disco simulato, attraverso le operazioni di shift e l'utilizzo di maschere.
- La comprensione delle struct e il collegamento tra le funzioni, lavorando da solo questo ha richiesto un apporto decisamente maggiore.

Risultato Finale:

Il risultato finale è un eseguibile ottenuto dalla compilazione di tutti i file .c denominato **“fs”** che nelle prime istruzione inizializza il disco creando un *disk.txt* e una directory **root.**

L’utilizzo finale del progetto consiste in un’interfaccia a terminale in cui all’utente è chiesta l’interazione attraverso un intero da inserire a schermo (tramite *scanf)* con cui può selezionare le seguenti operazioni:

1. **Creazione di un file:** viene richiesto all’utente il nome del file desiderato, dopodichè il file viene creato e viene creata una risposta a schermo.
2. **Lettura su file:** inserendo il nome del file viene letto il contenuto a schermo.
3. **Scrittura su file:** inserendo il nome del file, è possibile inserire un contenuto testuale all’interno del file.
4. **Creazione Directory:** si può creare una directory inserendo un nome.
5. **Eliminazione File o Directory:** si può eliminare un file o directory inserendone il nome.
6. **Stampa contenuto directory (ls):** stampa il contenuto della directory corrente.
7. **Cambia Directory:** inserendo il nome della directory, è possibile cambiare directory corrente.