Progetto - Simple File System with Inode

Il progetto ha lo scopo di realizzare un file-system "semplificato" utilizzando un file per simulare il reale comportamento di un disco di memoria.

Sono state realizzate le funzioni base di un file-system:

- Creazione di file e cartelle
- Eliminazione di file e cartelle
- Scrittura e lettura di un file
- Navigazione all'interno delle cartelle

Il progetto è stato sviluppato in maniera sequenziale: scrittura del codice della bitmap per la gestione della memoria, del file per simulare il disco sottostante, delle operazioni di alto livello del file-system e di una shell.

Bitmap.c

La bitmap viene utilizzata per tenere traccia dei blocchi liberi ed occupati in memoria: ogni bit rappresenta un blocco del disco, se il bit è 1 il blocco è occupato, se 0 è libero.
Sono state implementate le operazioni di conversione dell'indice della bitmap e dell'indice dell'array (contente la bitmap), come anche di prelevazione e settaggio dei bit e dei byte.
Tramite delle operazioni di shift e maschere si setta il bit. Per prima cosa viene convertito l'indice in modo da avere l'offset del byte: si shifta un bit 1 fino all'offset richiesto e si ottiene la maschera. Per impostare il bit a 1 il byte del vettore viene messo in OR con la maschera, per impostare il bit a 0, invece, il byte viene messo in AND con la maschera negata.
Sono state implementate anche delle funzioni di stampa della bitmap, sia dei byte che dei singoli bit.

DiskDriver.c

Il disco viene simulato tramite un file, la init andrà a creare, o aprire il file, inizializzando l'header e la bitmap. In questa sezione vengono implementate tutte le operazioni di lettura e scrittura del disco utilizzando le syscall read e write. La scrittura di un blocco verrà permessa solo se libero, occupandosi dell'aggiornamento della bitmap andando a settare il bit a 1 e aggiornando il primo blocco libero.
Si è voluta implementare una nuova funzione chiamata DiskDriver_rewrite, la quale va a sovrascrivere un blocco già utilizzato, omettendo le operazioni di controllo della bitmap e della ricerca del prossimo blocco libero.

Simplefs.c

Per sviluppare il file-system si allocano e si inizializzano le strutture generali tramite le operazioni init e format. Si crea la root, la cartella principale, che conterrà tutti i file e le cartelle presenti nel nostro sistema.
Ci sono due tipi di blocchi: i blocchi indice e i blocchi data.
Ogni file avrà come primo blocco un FirstDirectoryBlock o un FirsFileBlock, contenenti l'header, il FileControlBlock e i vettori degli indici dei blocchi.
L'header ha informazioni quali la natura del blocco e la sua posizione nel disco, il FileControlBlock contiene tutte le informazioni del file come la grandezza in byte, in blocchi, il blocco nel quale si trovano, il blocco della cartella genitore ed altre informazioni base.
I FirstBlock (directory o file) contengono un primo vettore (direct_blocks) con gli indici o dei Data_Block, o dei FirstDirectoryBlock, o dei FirstFileBlock (settati a -1 nel caso in cui fossero vuoti), e un secondo vettore (indirect blocks) composto da 3 elementi, i quali contengono gli indici rispettivamente del single_block, del direct_block e del triple_block. Se il direct_block non è sufficiente a indirizzare tutti i blocchi del file, viene usato il single_block che non contiene l'indice di un data block, ma quello di un altro blocco indice, che a sua volta punterà ai dati. In maniera analoga, se anche questi indici non fossero sufficienti, verranno utilizzati anche il double e il triple block, che conterranno rispettivamente un blocco indice di doppia e di tripla indirezione.
I blocchi data invece contengono l'header, e la struttura contenente i dati del disco.
Sono state implementate delle funzioni ausiliarie per rendere più modulare e gestibile il codice, per la ricerca e il controllo di un file, la scrittura dei dati su disco, la ricerca di un blocco conoscendo l'offset, la stampa di informazioni come l'index block, ecc.

Shell.c

Come ultima cosa è stata implementata una semplice interfaccia grafica, una shell, la quale permette l'interazione con l'utente. In questo caso è stata utilizzata una libreria esterna, linenoise, che permette l'utilizzo della cronologia delle operazioni, l'auto-completamento delle operazioni disponibili e il suggerimento dei parametri in input.
Per rendere ancora più client-friendly la shell, sono stati inseriti dei suggerimenti per la scrittura, la lettura e la seek del file.
Ogni funzione viene descritta nei vari file .h e con dei commenti aggiuntivi tra le principali operazioni per rendere chiaro ogni passaggio.

È stato implementato anche un MakeFile: in base all'argomento passato vengono eseguiti i test di ogni sezione.

Per eseguire la shell

- make shell
    - ./shell

Nella cartella test :

- make test_bitmap
    - ./test_bitmap
- make test_disk_driver
    - ./test_disk_driver
- make test_simpleFS
    - ./test_simpleFS