Documentazione del progetto di Sistemi Operativi

# Implementazione File System

## BitMap

La BitMap è stata implementata per mappare blocchi liberi e occupati.

Per l’implementazione della BitMap sono state utilizzate due struct.

La prima struct si occupa della BitMap vera e propria e i campi che la caratterizzano. Sono importanti infatti il numero dei bit, il numero dei blocchi e le entries.

La seconda struct caratterizza le entry della bitmap, sono importanti il numero dell’entry e il numero del bit.

Per una corretta esecuzione della BitMap sono state implementate le seguenti funzioni:

- void BitMap\_init(BitMap\* b, int bitmap\_blocks, int disk\_blocks, int occupation);

Inizializza una nuova BitMap vuota.

È una funzione void, quindi non avrà nessun valore di ritorno ma inizializzerà tutti i campi della struct a dei valori di default che indicano che il disco è completamente vuoto.

- BitMapEntryKey BitMap\_blockToIndex(int num);

Prende come parametro un intero che indica il numero dei blocchi e restituisce un BitMapEntryKey popolata con rispettivamente il primo campo della struct che indica l’entry della BitMap che contiene l’informazione sul blocco e il secondo campo indica lo spiazzamento all’interno del blocco per individuare il bit di cui abbiamo bisogno.

- int BitMap\_indexToBlock(int entry, uint8\_t bit\_num);

Funzione elementare che converte un bit in un indirizzo lineare.

- int BitMap\_get(BitMap\* bmap, int start, int status);

Funzione che restituisce l'indice del primo bit con stato status nella BitMap bmap e inizia a cercare dalla posizione start in poi. Restituisce -1 se non viene trovato.

- int BitMap\_set(BitMap\* bmap, int pos, int status);

Funzione che setta il bit con indice pos a status. Ha come valore di ritorno status in caso di successo.

- int BitMap\_test(BitMap\* bmap, int pos);

Funzione che ritorna lo status del bit con indice pos all’interno della BitMap bmap.

## Common

Il file common.h contiene la macro ﻿CHECK\_ERR molto importante poiché permette di verificare se una funzione è andata a buon fine, altrimenti stampa un messaggio di errore e chiude il programma.

Al suo interno contiene le variabili globali che si utilizzano all’interno dell’intero progetto SimpleFS.

## Diskdriver

Per una corretta e lineare esecuzione del DiskDriver sono state utilizzate due struct.

La prima struct serve per caratterizzare il DiskDriver attraverso 3 parametri che corrispondono a un puntatore ad una struttura DiskHeader, un puntatore ad una BitMap e un file descriptor riguardante il file su cui si crea il filesystem. La seconda struttura caratterizzare DiskHeader attraverso i campi che identificano: numero dei blocchi, numero dei blocchi usati, numero dei blocchi liberi e l’indice del primo blocco libero.

-void DiskDriver\_init(DiskDriver\* disk, const char\* filename, int num\_blocks);

Funzione che crea il file che rappresenta il disco, allocando lo spazio necessario per il disco, attraverso un calcolo delle dimensioni più grandi che può assumere la bitmap.

-int DiskDriver\_load(DiskDriver\* disk, const char\* filename);

Funzione che si occupa del caricamento di un disco già esistente.

-int DiskDriver\_readBlock(DiskDriver\* disk, void\* dest, int block\_num);

Funzione che legge il blocco in posizione block\_num. Ha come valore di ritorno -1 se il blocco risulta libero nella BitMap e quindi non può essere letto, 0 altrimenti.

-int DiskDriver\_writeBlock(DiskDriver\* disk, void\* src, int block\_num);

Funzione che scrive il blocco in posizione block\_num. Ha come valore di ritorno -1 se l’operazione non è consentita, 0 altrimenti. Se il blocco in posizione block\_num è libero nella BitMap, esso viene scritto, altrimenti viene modificato.

-int DiskDriver\_freeBlock(DiskDriver\* disk, int block\_num);

Funzione che libera il blocco all’interno del disco in posizione block\_num. Ritorna -1 se l’operazione non è possibile.

-int DiskDriver\_getFreeBlock(DiskDriver\* disk, int start);

Funzione che ritorna l’indice del primo bocco libero all’interno del disco, blocchi liberi cercati in maniera circolare.

-int DiskDriver\_flush(DiskDriver\* disk);

Funzione che obbliga il sistema a scrivere le modifiche attualmente salvate nei buffer di sistema.

-void DiskDriver\_shutdown(DiskDriver\* disk);

Funzione che si occupa dello svuotamento dei buffer riguardanti il disco e della chiusura dei file relativi al DiskDriver.

## Simplefs

-DirectoryHandle\* SimpleFS\_init(SimpleFS\* fs, DiskDriver\* disk);

Funzione che inizializza il file system su un disco già esistente e restituisce un handle alla directory di livello superiore memorizzata nel primo blocco.

-void SimpleFS\_format(SimpleFS\* fs);

Funzione che serve per formattare e reinizializzare un file system già esistente. Attraverso questa funzione verrà resettata anche la BitMap dei blocchi occupati sul disco.

-FileHandle\* SimpleFS\_createFile(DirectoryHandle\* d, const char\* filename);

Funzione che crea un file vuoto nella directory d e restituisce NULL in caso di errore o altrimenti

un handle ad un file vuoto.

-int SimpleFS\_readDir(char\*\* names, DirectoryHandle\* d);

Funzione che legge il nome di tutti i file e le directory figli della directory passata come parametro.

-FileHandle\* SimpleFS\_openFile(DirectoryHandle\* d, const char\* filename);

Funzione che apre un file già esistente nella directory d, allocando e restituendo un FileHandle.

-void SimpleFS\_close(FileHandle\* f);

Funzione che chiude un file già esistente dellocando il FileHandle fornito e le strutture conenute al suo interno.

-int SimpleFS\_write(FileHandle\* f, void\* data, int size);

Funzione che restituisce il numero di byte scritti, che sovrascrive e alloca nuovo spazio se necessario per scrivere nel file.

-int SimpleFS\_read(FileHandle\* f, void\* data, int size)**;**

Funzione che restituisce il numero di byte letti,che sovrascrive e alloca nuovo spazio se necessario per leggere il file.

-int SimpleFS\_seek(FileHandle\* f, int pos);

Funzione che restituisce il numero di byte letti spostando il puntatore corrente in pos, che sarà restituito in caso di successo, -1 in caso di errore (file troppo corto).

-int SimpleFS\_changeDir(DirectoryHandle\* d, const char\* dirname);

Funzione che cerca una directory in d. Se dirname è uguale a "..", sale di un livello. La funzione restituisce 0 in caso di successo, valore negativo in caso di errore. La funzione ha modifica il DirectoryHandle fornito in input.

-int SimpleFS\_mkDir(DirectoryHandle\* d, const char\* dirname);

Funzione che crea una nuova directory figlia della directory passata come parametro. La funzione restituisce 0 in caso di successo, -1 in caso di errore.

-int SimpleFS\_remove(DirectoryHandle\* d, const char\* filename);

Funzione che rimuove il file nella directory corrente. La funzione restituisce 0 in caso di successo,-1 in caso di errore. Se viene rimossa una directory, saranno eliminati tutti i file e le directory contenuti in essa.

## Shell

Parte integrante del nostro progetto è anche una shell che permette agli utenti di interagire con il sistema operativo. Essa funziona attraverso funzioni da noi implementate e permette l’utilizzo delle seguenti operazioni :

* quit

Per chiudere la nostra shell.

* ls

per mostrare l’elenco dei file contenuti nella working directory.

* cd

per cambiare directory di lavoro.

* mkdir

per creare una directory.

* rm

Per rimuovere un file o una directory.

* touch

per creare un nuovo file vuoto.

* cat

per concatenare i file e scriverli sullo schermo.

* cp

per copiare un file da o verso SimpleFs.

* info

per avere informazioni sul disco.

* info -bmap

per avere info sulla bitmap del disco.

* echo

per replicare un messaggio o per scriverlo su un file.

**Fortunato Tocci**

**Gabriele Sellani**

**Mattia Nicolella**