Gebze Technical University

CSE 312 /CSE 504 Operating Systems

Midterm Exam Project Report

Mehmet Hafif 171044042

31 May 2020

Abstract

Design and implementation of a simplified UNIX like file system in C.

Content

1 Introduction

In this project I implemented a simple UNIX-like file system in C language that helped me understand the hierarchical directory and inode structures of file systems. Simple operations can be done over this simulated file system.

1.1 Constraints

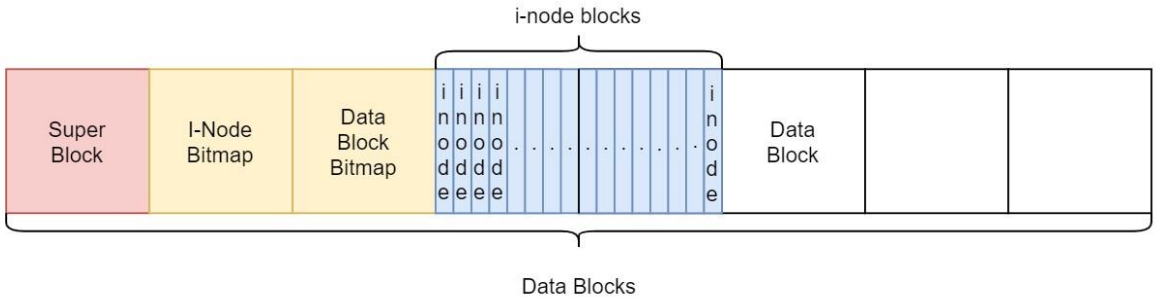
In this design there is one constraint due to my design decisions it is:

File name size: In order to prevent extra work of dynamic allocation of file names I decided to give filenames a limit of 20 character. This is valid only for filenames not the full path of the file.

2 Design of the File System

2.1 File System Structure

File system consist of same size data blocks first several block is used to keep some important information and those structures generally used on memory. Below image is a representation of how file system is structured around data blocks.



2.2 Superblock Structure

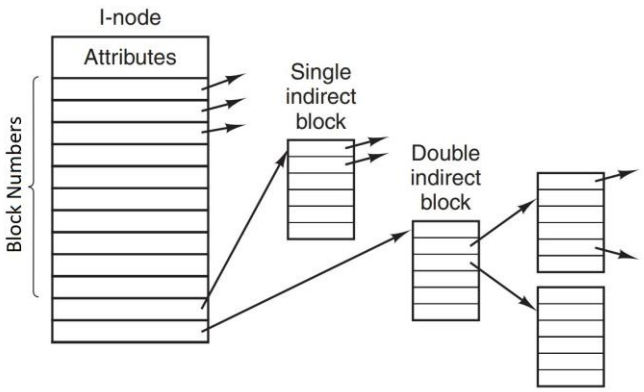
```
typedef struct{
    int root_dir_inode;
    int free_block_count;
    int free_inode_count;
    int block_size;
    int inode_count;
} SuperBlock;
```

Superblock resides in the first block of the file system, and it keeps such informations: inode number of root "/" directory, free block count, free inode count, block size and inode count.

2.3 I-Node Structure

I-Nodes reside in inode blocks that starts from 4^{th} block and finish depending on I-node count. Inodes keeps those informations: File type, last modification time, size, number of block it's data occupys, 10 direct block number, one indirect block number and one double indirect block number.

```
typedef struct{
    file_t type;
    struct timeval last_modified;
    int size;
    int block_count;
    int direct_block[10];
    int indirect_block;
    int double_indirect_block;
} Inode;
```



Triple indirect blocks are not implemented because they are actually not needed since our simulated file system has exact size of 1MB and minimum block size is 1KB, in this design one double indirect block can hold more than 1MB of data even with minimum block size. So, keeping in mind the file system size limit of 1MB, in this design a file can hold up to <1MB of data and since first several block is always occupied, tests are made with file sizes of 900KB and works perfectly.

2.4 Directory Structure

```
typedef struct{
    int inode;
    char name[MAX_FILENAME];
} DirEntry;

typedef struct{
    int num_entry;
    DirEntry *dir_entry;
} Directory;
```

In this design directories are structured data blocks that holds the directory entry list, a directory entry is file name and it's inode number. Directory entry list is dynamicly allocated therefore it depends on the block size. Due to this this struct is serialized when it is written to the desired data block.

2.5 Bitmap Structure

There are 2 bitmap located in the second and third blocks of the file system, those are in design array of bits, stating whether a block/inode is free or not. Since bit arrays are not a valid data type char array is used and bit operations are made to index bits of every char.

3 File Operations

3.1 List

Lists the contents of the given directory path.

Function name in source code: *int list_ dir(char *name)*

How it works: It gets the inode no of the given path then reads the first direct block that holds the directory structure. Then in a loop prints the directory entires that holds a file or directory. It prints the size, modification time, type and filename.

3.2 Mkdir

Makes a directory at given path if possible.

Function name in source code: *int make_ dir(char *name)*

How it works: It creates new inode and initialize the values. Later it finds the inode of the parent directory then reads its first direct block as a directory, then loops over the directory entries, finds the first empty spot and adds new entry. After this operation it creates a new directory structure initialize it and write it to the first direct block of new created inode. Additionally in inode map and block map occupied bits are set to 1.

3.3 Rmdir

Removes a directory at given path if possible.

Function name in source code: *int remove_ dir(char *name)*

How it works: In order to remove the directory parent directory inode is found then its first direct block is read into a directory structure, later on by looping over the directory entries desired entry is found and its name is deleted then it's inode and first direct block is freed on bitmaps.

3.4 Dumpe2fs

Gives information about the file system.

Function name in source code: *int simple_ dumpe2fs()*

How it works: It displays some important information at first by reading the superblock content like block size, block count, inode count, free block count, free inode count then it iterates through all inodes checks if it is free or used, if used then displays its inode no, type and number of block that file occupies.

3.5 Write

Creates file and writes data to the file.

Function name in source code:

*int write_ file(char *new_ name, char *from_ name)*

How it works: It first creates the file by another function call, there it is checked whether the file previously exist or not. If it exist then it's previous blocks are freed if not it is created and new inode no is gathered, then new inodes needed blocks are assigned by calling another function by looking at the file size it assigns the amount of blocks that is needed so prevenst allocating unnecessary blocks for a file. After that the external file's content is read then written to the data blocks in an order.

3.6 Read

Reads data from the file.

Function name in source code:

*int read_ file(char *file_ name, char *to_ name)*

How it works: The inode number of the file that will be read is gathered then by calling block number finder function data blocks are found with respect to their index and they are read into a buffer. At the end buffer is written to the external file.

3.7 Del

Deletes file from the path.

Function name in source code: *int delete_ file(char *name)*

How it works: In order to delete a file inode number of desired file's parent is retrieved then its first direct block is read into a directory structure and it's entry is deleted. Also data blocks occupied by that inode is also cleaned.