CSE 312 HOMEWORK 3

MYFILESYSTEM

SIMPLIFED FAT LIKE FILE SYSTEM

Çağrı Çaycı

1901042629

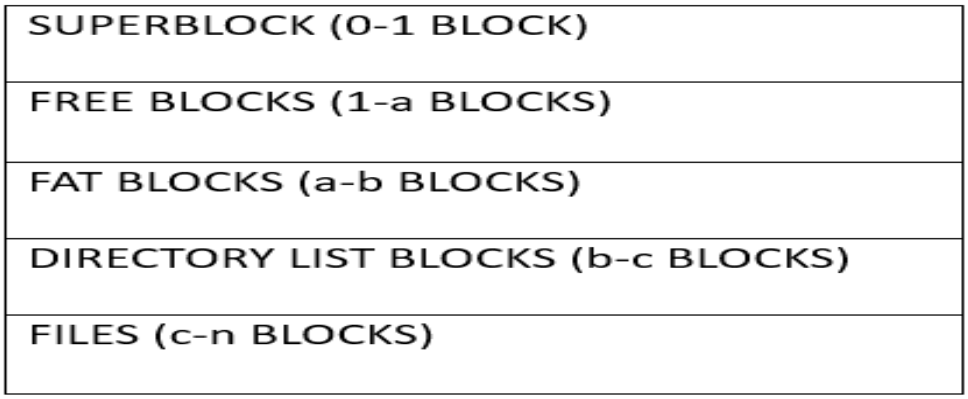
OVERVIEW

In this project, simplified fat like file system is implemented. A file system is a method or structure used by operating systems to organize, store, and retrieve files on a storage device such as a hard disk drive, solid-state drive, or flash drive. It provides a logical framework for managing files, directories (also known as folders), and metadata associated with them.

File systems are responsible for how data is stored, accessed, and managed on a storage device. They define the rules and structures for naming files, organizing them into directories, and keeping track of their locations on the storage medium. Additionally, file systems handle tasks such as file creation, deletion, modification, and access control.

FILE SYSTEM STRUCTURE

File systems consist of sections. Therefore, to implement a file system these sections must be managed. In this project, the file system consists of 5 sections. These are Superblock, Free Table, FAT Table, Directory List and Files section.



Superblock: Superblock contains necessary information about file system such as total number of blocks, number of free blocks, free blocks, fat blocks, directory list blocks.

```
typedef struct SuperBlock{
    int block_size;
    int number_of_blocks;
    int free_blocks;
    int fat_blocks;
    int directory_blocks;
    SuperBlock(int block_size, int number_of_blocks, int free_blocks, int fat_blocks, int directory_blocks);
    SuperBlock();
}SuperBlock;
```

Free Table: Free table is just an integer array which keeps 0s for full, 1s for free blocks. It keeps track of which block is free.

FAT Table: FAT table is just an integer array which keeps -1 for EOF or index of next block. It keeps track of how files are stored.

Directory List: Directory list is a structure to keep metadata of files and directories. The first entry of this list is root directory.

```
typedef struct DirectoryEntry{
    char filename[128];
    char parent[128];
    time_t last_modification;
    int size;
    int first_block;
    int directory;
    int exist = 0;
    DirectoryEntry(char filename[], char parent[], time_t last_modification, int size, int first_block, int directory);
    DirectoryEntry();
}DirectoryEntry;
```

Files: This section contains files' content block by block.

CREATING FILE SYSTEM

At the creation of file system, some settings must be done. Firstly, the number of fat blocks, free blocks, directory list blocks and the total number of blocks in the file system must be calculated according to maximum file system size and size of a block. Then, space must be allocated in the file system for sections. Also, the root directory must be added to the directory list section.

```
int block_size = 1024 * atoi(argv[1]); /* BLOCK SIZE IS CALCULATED. */

int number_of_blocks = MAXSIZE / block_size; /* NUMBER OF BLOCKS IS CALCULATED. */

int fat_blocks = ((number_of_blocks * sizeof(int)) + block_size - 1) / block_size; /* NUMBER OF FAT BLOCKS IS CALCULATED. */

int directory_blocks = ((MAXNUMBEROFFILE * sizeof(DirectoryEntry)) + block_size - 1) / block_size; /* NUMBER OF DIRECTORY BLOCKS IS CALCULATED. */

int free_blocks = ((number_of_blocks * sizeof(int)) + block_size - 1) / block_size; /* NUMBER OF FREE BLOCKS IS CALCULATED. */
```

```
FILE * file = fopen(filename, "wb");
if(file == NULL){
    perror("fopen");
    exit(-1);
}
SuperBlock superBlock(block_size, number_of_blocks, free_blocks, fat_blocks, directory_blocks);

DirectoryEntry rootDir(rootName, rootParent, 0, 0, 0, 1);

fwrite(&superBlock, sizeof(SuperBlock), 1, file);

int free_table[number_of_blocks];

for(int i = 0; i < number_of_blocks; i++){
    if(i < 1 + free_blocks + fat_blocks + directory_blocks)
        free_table[i] = 0; // NOT FREE //
    else
        free_table[i] = 1; // FREE //
    fwrite(&free_table[i], sizeof(int), 1, file);
}


int fat_table[number_of_blocks];

for(int i = 0; i < number_of_blocks; i++){
    fat_table[i] = -1;
    fwrite(&fat_table[i], sizeof(int), 1, file);
}

fwrite(&rootDir, sizeof(DirectoryEntry), 1, file);

fseek(file, (directory_blocks - 1) * block_size, SEEK_SET);

char buffer[block_size];

for(int i = 0; i < number_of_blocks - fat_blocks - directory_blocks - free_blocks - 1; i++){
    fwrite(buffer, sizeof(char), block_size, file);
}

fclose(file);
```

MANAGING FILE SYSTEM

After the file system is created, it is possible to make some operations on the file system. These operations are dir, mkdir, rmdir, write, del, read, and dumpe2fd. Firstly, getting necessary information about the file system is essential to provide the operations for the user. This operation can be done by reading the first block (Superblock) of the file system. After that, all the other sections except the files section are read.

```
FILE * file = fopen(fileSystem, "rb");
if(file == NULL){
    perror("fopen");
    exit(-1);
}

SuperBlock superBlock;

fread(&superBlock, sizeof(SuperBlock), 1, file);

int free_table[superBlock.number_of_blocks];

fread(&free_table, sizeof(int), superBlock.number_of_blocks, file);

int fat_table[superBlock.number_of_blocks];

fread(&fat_table, sizeof(int), superBlock.number_of_blocks, file);

fseek(file, sizeof(SuperBlock) + (superBlock.fat_blocks + superBlock.free_blocks) * superBlock.block_size, SEEK_SET);

DirectoryEntry directoryEntries[MAXNUMBEROFFILE];

fread(directoryEntries, sizeof(DirectoryEntry), MAXNUMBEROFFILE, file);

fclose(file);
```

LISTING CONTENT OF DIRECTORIES

To list the content of a directory, the path must be checked whether it is directory or a file. If the path is the path of a file, the filename must be printed. If the path is the path of a directory, all the entries of directory list must be checked, if their parent's name is equal to directory name, their filenames must be printed.

```
void dir(char * parent, char * child, DirectoryEntry directoryEntries[]){
    int fileType = entryType(parent, child, directoryEntries);
    if(fileType == 0){ // IT IS A FILE
        printf("%s", child);
    }else{
        int number_of_files = 0;
        for(int i = 0; i < MAXNUMBEROFFILE; i++){
            if(strcmp(directoryEntries[i].parent, child) == 0){
                printf("%s\t", directoryEntries[i].filename);
                number_of_files++;
            }
        }
        if(number_of_files > 0)
            printf("\n");
    }
}
```

CREATING A DIRECTORY

To create a directory, it must be checked that there is a file or directory with the same name in the same directory. If this is the case, an error message must be printed. Otherwise, the directory must be added to the first empty place in the directory list.

```
void mkdir(char * parent, char * child, DirectoryEntry directoryEntries[]){
    int fileType = entryType(parent, child, directoryEntries);
    if(fileType == -1){
        for(int i = 0; i < MAXNUMBEROFFILE; i++){
            if(directoryEntries[i].exist != 999){
                strcpy(directoryEntries[i].filename, child);
                strcpy(directoryEntries[i].parent, parent);
                time(&directoryEntries[i].last_modification);
                directoryEntries[i].directory = 1;
                directoryEntries[i].size = 0;
                directoryEntriesChanged = 1;
                directoryEntries[i].exist = 999;
                return;
            }
        }
    }else if(fileType == 0){ // THERE IS A FILE ALREADY WITH GIVEN NAME //

    }else if(fileType == 1){
        printf("CANNOT CREATE DIRECTORY \"%s\": IT EXISTS!\n", child);
    }
}
```

REMOVING A DIRECTORY

To remove a directory, it must be checked that the directory list contains the directory, and it is not a file. If this is the case, an error message must be printed. Otherwise, the directory and contents of the directory must be removed from the directory list.

```
void rmdir(char * parent, char * child, DirectoryEntry directoryEntries[]){
    int fileType = entryType(parent, child, directoryEntries);
    if(fileType == 0){ // IT IS A FILE
        printf("FAILED TO REMOVE \"%s\": NOT A DIRECTORY!\n", child);
    }else if(fileType == -1){
        printf("FAILED TO REMOVE \"%s\": NO SUCH FILE OR DIRECTORY!\n", child);
    }else{
        for(int i = 0; i < MAXNUMBEROFFILE; i++){
            if(strcmp(directoryEntries[i].parent, child) == 0 || strcmp(directoryEntries[i].filename, child) == 0){
                strcpy(directoryEntries[i].filename, "");
                strcpy(directoryEntries[i].parent, "");
                directoryEntries[i].last_modification = 0;
                directoryEntries[i].directory = 0;
                directoryEntries[i].size = 0;
                directoryEntriesChanged = 1;
                directoryEntries[i].exist = 0;
                break;
            }
        }
    }
}
```

CREATING A FILE

To create a file, the content of the given file must be added to the files section block by block sequentially. The block is put in the first free block in the free table. During this operation, the block number of each block must be put to the corresponding entry of FAT table and corresponding entry of free table must be set as 0 (full). Also, the metadata of the file must be added to the directory list.

```
int counter = 0;

FILE * fptr = fopen(fileSystem, "rb+");
if(fptr == NULL){
    perror("fopen");
    exit(-1);
}
int next = -1;
for(int i = 0; i < superBlock.number_of_blocks && counter < file_blocks; i++){
    if(free_table[i] == 1){ // FREE BLOCK IS FOUND. //
        fat_table[i] = next;
        free_table[i] = 0;
        next = i;
        fseek(fptr, sizeof(SuperBlock) + i * superBlock.block_size, SEEK_SET);
        fwrite(fileArray[file_blocks - counter - 1], sizeof(char), superBlock.block_size, fptr);
        rewind(fptr);
        counter++;
    }
}

if(counter != file_blocks){
    printf("SOME PARTS OF THE FILE IS NOT WRITTEN TO \"%s\": MEMORY IS FULL!\n", child);
}

fclose(file);
fclose(fptr);

for(int i = 0; i < MAXNUMBEROFFILE; i++){
    if(directoryEntries[i].exist != 999){
        strcpy(directoryEntries[i].filename, child);
        strcpy(directoryEntries[i].parent, parent);
        time(&directoryEntries[i].last_modification);
        directoryEntries[i].directory = 0;
        directoryEntries[i].size = size;
        directoryEntries[i].first_block = next;
        directoryEntries[i].exist = 999;
        break;
        // UPDATE PARENTS SIZE //
    }
}
```

READING A FILE

To read a file, the name of that file is searched in the directory list and its metadata is obtained. The first block number is taken from this metadata. Starting from the first block until EOF is encountered, the data of all blocks are transferred to the given file. During this operation, next block number is got from the FAT table.

```
void read(char * parent, char * child, char * filename, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[]){
    int fileType = entryType(parent, child, directoryEntries);
    if(fileType == 1){
        printf("\"%s\": IS A DIRECTORY!\n", child);
    }else if(fileType == -1){
        printf("\"%s\": NO SUCH FILE OR DIR!\n", child);
    }else{
        FILE * file = fopen(filename, "w");
        FILE * fptr = fopen(fileSystem, "rb+");
        for(int i = 0; i < MAXNUMBEROFFILE; i++){
            if(strcmp(directoryEntries[i].filename, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0){
                int start = directoryEntries[i].first_block;

                while(start != -1){
                    char buffer[superBlock.block_size];

                    fseek(fptr, sizeof(SuperBlock) + start * superBlock.block_size, SEEK_SET);

                    fread(buffer, sizeof(char), superBlock.block_size, fptr);

                    rewind(fptr);

                    fwrite(buffer, sizeof(char), strlen(buffer), file);

                    memset(buffer, 0, sizeof(buffer));

                    start = fat_table[start];
                }
                break;
            }
        }
        fclose(file);
        fclose(fptr);
    }
}
```

DELETING A FILE

To delete a file, the content of the given file must be removed from the files section block by block sequentially. The first block number is got from the corresponding entry of the directory list. During this operation, the block number of each block must be got from the corresponding entry of FAT table. Also, corresponding entry of free table must be set as 1 (free).

```
void del(char * parent, char * child, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], int free_table[]){
    int fileType = entryType(parent, child, directoryEntries);
    if(fileType == 1){
        printf("\"%s\": IS A DIRECTORY!\n", child);
    }else if(fileType == -1){
        printf("\"%s\": NO SUCH FILE OR DIR!\n", child);
    }else{
        for(int i = 0; i < MAXNUMBEROFFILE; i++){
            if(strcmp(directoryEntries[i].filename, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0){
                int current = directoryEntries[i].first_block;
                int next;
                do{
                    next = fat_table[current];
                    free_table[current] = 1;
                    fat_table[current] = -1;
                    current = next;

                }while(current != -1);
                strcpy(directoryEntries[i].filename, "");
                strcpy(directoryEntries[i].parent, "");
                directoryEntries[i].last_modification = 0;
                directoryEntries[i].directory = 0;
                directoryEntries[i].size = 0;
                directoryEntries[i].exist = 0;
                break;
            }
        }
        freeTableChanged = 1;
        fatTableChanged = 1;
        directoryEntriesChanged = 1;
    }
}
```

TESTS

```
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ make
g++ lib.cpp makeFileSystem.cpp -o makeFileSystem
g++ lib.cpp fileSystemOper.cpp -o fileSystemOper
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./makeFileSystem 4 mySystem.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data mkdir /usr
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data mkdir /usr/ysa
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data mkdir /bin/ysa
NO SUCH FILE OR DIR!
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data write /usr/ysa/file1 linuxFile.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data write /usr/file2 linuxFile.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data write /file3 linuxFile.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data dir /
usr     file3
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data del /usr/ysa/file1
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data dumpe2fs
BLOCK SIZE: 4096
NUMBER OF BLOCKS: 4096
FREE BLOCKS NUMBER: 4048
NUMBER OF FILES: 2
NUMBER OF DIRECTORIES: 3
OCCUPIED BLOCKS FOR file2: 47 46
OCCUPIED BLOCKS FOR file3: 49 48
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data read /usr/file2 linuxFile2.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ cmp linuxFile2.data linuxFile.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$
```

(GENERAL TEST)

```
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ make
g++ lib.cpp makeFileSystem.cpp -o makeFileSystem
g++ lib.cpp fileSystemOper.cpp -o fileSystemOper
^[[Acagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./makeFileSystem 4 mySystem.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data mkdir /usr
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data write /usr/file linuxFile.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data dir /usr
file
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data dir /
usr
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data dumpe2fs
BLOCK SIZE: 4096
NUMBER OF BLOCKS: 4096
FREE BLOCKS NUMBER: 4050
NUMBER OF FILES: 1
NUMBER OF DIRECTORIES: 2
OCCUPIED BLOCKS FOR file: 45 44
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data write /file2 linuxFile.data
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data dumpe2fs
BLOCK SIZE: 4096
NUMBER OF BLOCKS: 4096
FREE BLOCKS NUMBER: 4048
NUMBER OF FILES: 2
NUMBER OF DIRECTORIES: 2
OCCUPIED BLOCKS FOR file: 45 44
OCCUPIED BLOCKS FOR file2: 47 46
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data del /usr/file
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data dumpe2fs
BLOCK SIZE: 4096
NUMBER OF BLOCKS: 4096
FREE BLOCKS NUMBER: 4050
NUMBER OF FILES: 1
NUMBER OF DIRECTORIES: 2
OCCUPIED BLOCKS FOR file2: 47 46
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data write /file3 cayciii.txt
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$ ./fileSystemOper mySystem.data dumpe2fs
BLOCK SIZE: 4096
NUMBER OF BLOCKS: 4096
FREE BLOCKS NUMBER: 4046
NUMBER OF FILES: 2
NUMBER OF DIRECTORIES: 2
OCCUPIED BLOCKS FOR file3: 49 48 45 44
OCCUPIED BLOCKS FOR file2: 47 46
cagri@CSE344:~/Masaüstü/CSE 312 HW 3$
```

(GENERAL TEST)

ADDITIONAL NOTES

- THERE MIGHT BE SOME SMALL DIFFERENCES BETWEEN THE REPORT AND THE ACTUAL CODE.
- PLEASE BE LOYAL TO COMMAND LINE FORMAT. (DO NOT USE “\”, USE “/” INSTEAD.
- PLEASE USE 16KB, 8KB, 4KB, 2KB ETC. TOO LARGE OR TOO SMALL BLOCK SIZE MIGHT CAUSE UNEXPECTED BEHAVIOR.