HW2 report Eren Torlak 210104004090

Design Report

- *Gebze Technical University*
- *Department of Computer Engineering*
- *CSE 312 - Operating Systems*

NOTES:

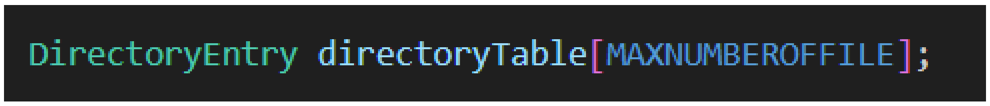
- Progam has part2.cpp and part3.cpp
- I used / for directory.

File System Design

In this project, we will design and implement a simplified FAT12-like file system. The FAT (File Allocation Table) file system is a robust and well-understood structure, providing a good foundation for learning about file systems. This report outlines the design decisions and structures used in creating our file system.

1. Directory Table and Directory Entries

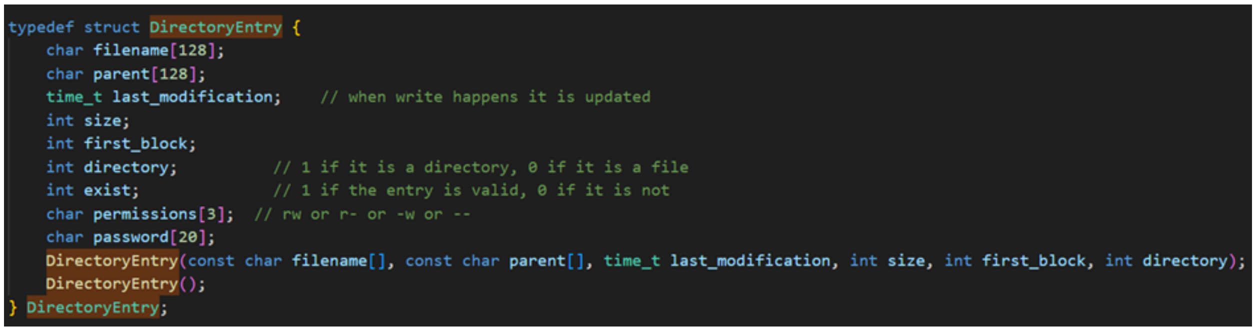
Directory Table:



```
DirectoryEntry directoryTable[MAXNUMBEROFFILE];
```

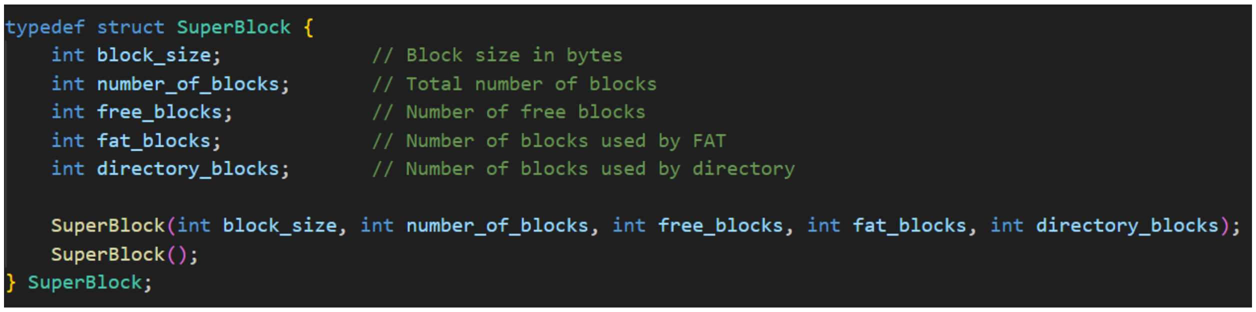
- The directory table is an array of `DirectoryEntry` structures. Each entry represents a file or directory in the file system. The table allows for up to 512 entries. It is a design choice, it can be changed.

Directory Entries:



```
typedef struct DirectoryEntry {
    char filename[128];
    char parent[128];
    time_t last_modification;    // when write happens it is updated
    int size;
    int first_block;
    int directory;          // 1 if it is a directory, 0 if it is a file
    int exist;              // 1 if the entry is valid, 0 if it is not
    char permissions[3];  // rw or r- or -w or --
    char password[20];
    DirectoryEntry(const char filename[], const char parent[], time_t last_modification, int size, int first_block, int directory);
    DirectoryEntry();
} DirectoryEntry;
```

- Each `DirectoryEntry` contains the following fields:
- `filename`: A character array with a maximum length of 128 bytes, allowing for descriptive file names.
- `parent`: A character array with a maximum length of 128 bytes, indicating the parent directory. The root directory's parent is set to "NOPARENT".
- `last_modification`: A `time_t` value indicating the last modification time, updated whenever the file is written.
- `size`: An integer representing the file size in bytes.
- `first_block`: An integer pointing to the first block of the file in the FAT.
- `directory`: An integer flag (1 for directory, 0 for file) indicating whether the entry is a directory.
- `exist`: An integer flag (1 for valid entry, 0 for invalid) indicating if the entry is currently used.
- `permissions`: A character array of length 3 (e.g., "rw-", "r--", "-w-", "---") indicating read and write permissions for the owner.
- `password`: A character array of length 20 for password protection, allowing basic file security.



```
typedef struct SuperBlock {
    int block_size;             // Block size in bytes
    int number_of_blocks;       // Total number of blocks
    int free_blocks;            // Number of free blocks
    int fat_blocks;             // Number of blocks used by FAT
    int directory_blocks;       // Number of blocks used by directory

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

2. Free Blocks Management

- *Free Blocks*:
- The free block management is handled using a free block table. This table is an array of integers, with each index representing a block in the file system:
- A value of 1 indicates that the block is free.
- A value of 0 indicates that the block is occupied.
- The free block table is stored immediately after the superblock in the file system data file.

3. Handling Permissions

- *Owner Permissions*:

It can handle: +r, -r, +w, -w or +rw, -rw

- Permissions are stored as a 3-character array within the `DirectoryEntry` structure, representing read (r) and write (w) permissions for the file owner:
- The first character is 'r' if read permission is granted, '-' otherwise.
- The second character is 'w' if write permission is granted, '-' otherwise.
- The permissions are manipulated using the `chmodFile` function, which can add or remove read and write permissions based on the provided parameters.

5. Handling Password Protection

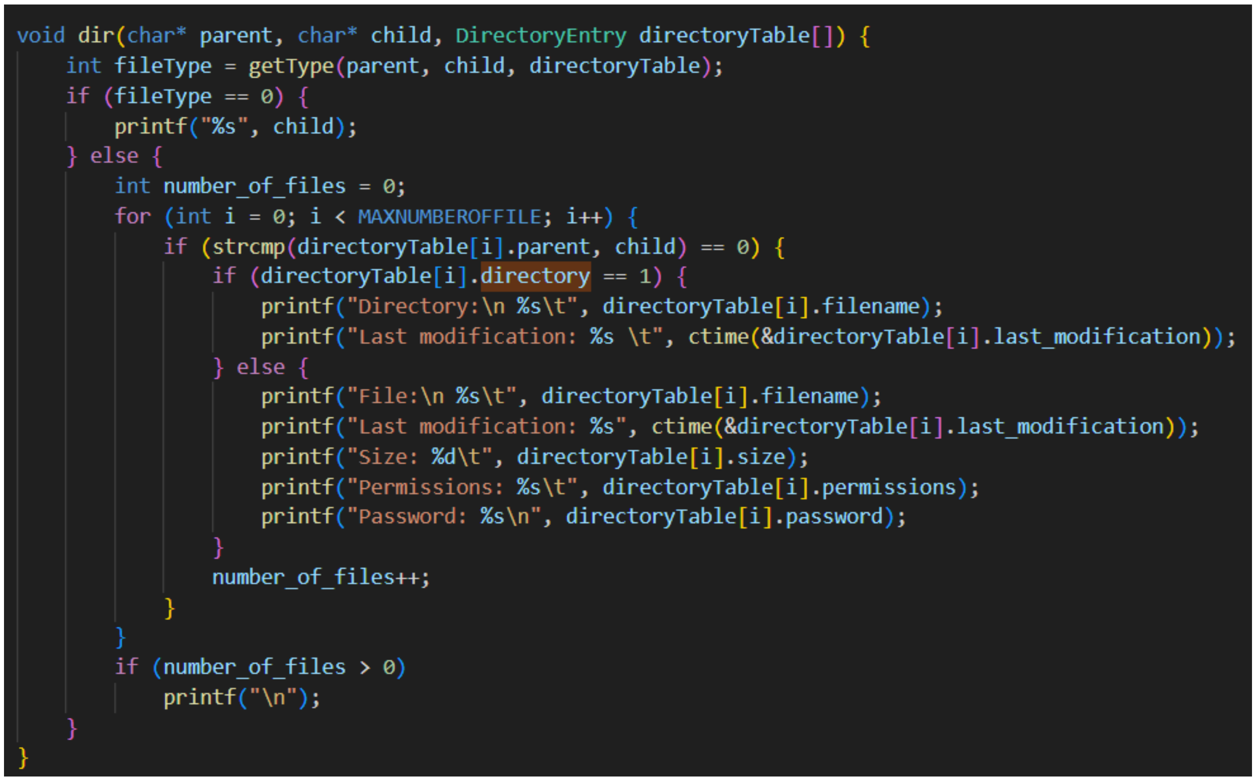
- *Password*:
- Password protection is implemented to add a layer of security to files. Each `DirectoryEntry` includes a fixed-length character array for storing a password.

- The password is managed using the `addpwFile` function, which sets or modifies the password for a given file.
- When performing operations on a password-protected file, the provided password is checked against the stored password. If they do not match, the operation is denied.

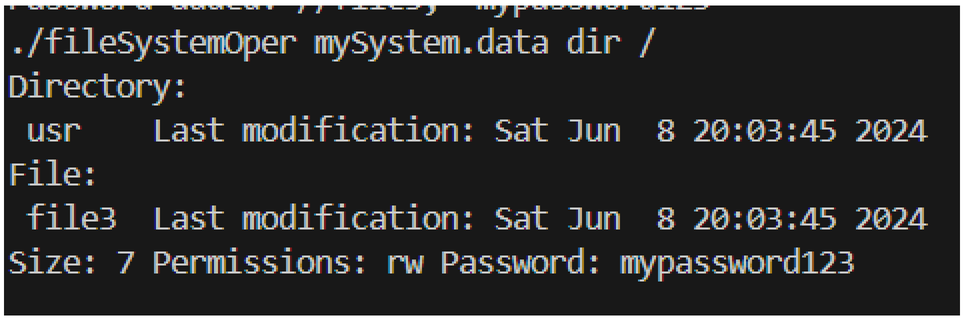
Source Code Functions for File System Operations

Directory Operations

dir(char parent, char child, DirectoryEntry directoryEntries[])**:



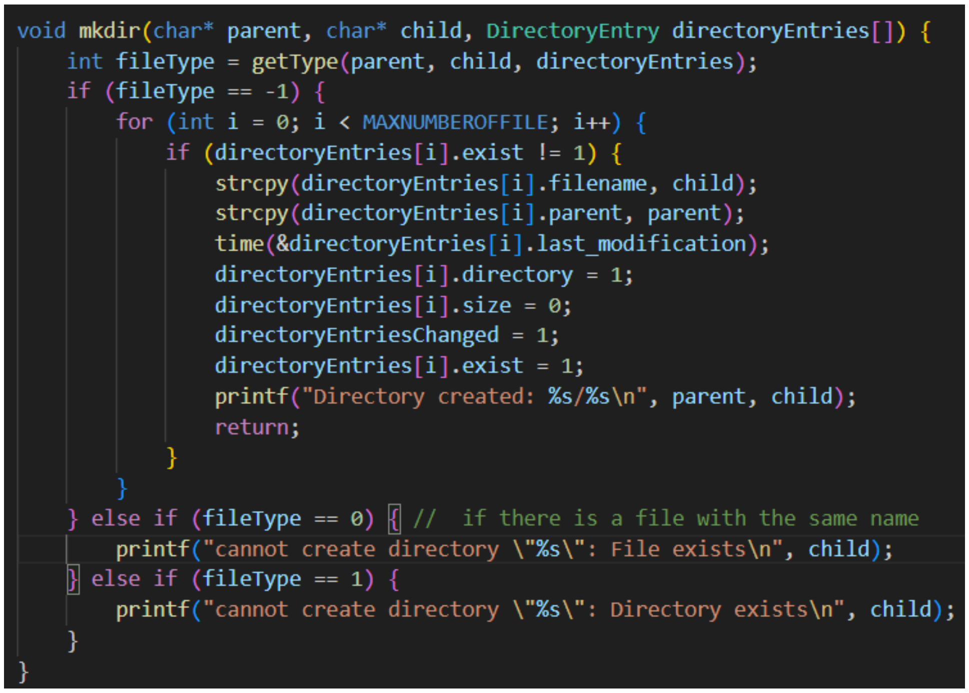
```c
void dir(char* parent, char* child, DirectoryEntry directoryTable[]) {
    int fileType = getType(parent, child, directoryTable);
    if (fileType == 0) {
        printf("%s", child);
    } else {
        int number_of_files = 0;
        for (int i = 0; i < MAXNUMBEROFFILE; i++) {
            if (strcmp(directoryTable[i].parent, child) == 0) {
                if (directoryTable[i].directory == 1) {
                    printf("Directory:\n %s\t", directoryTable[i].filename);
                    printf("Last modification: %s \t", ctime(&directoryTable[i].last_modification));
                } else {
                    printf("File:\n %s\t", directoryTable[i].filename);
                    printf("Last modification: %s", ctime(&directoryTable[i].last_modification));
                    printf("Size: %d\t", directoryTable[i].size);
                    printf("Permissions: %s\t", directoryTable[i].permissions);
                    printf("Password: %s\n", directoryTable[i].password);
                }
                number_of_files++;
            }
        }
        if (number_of_files > 0)
            printf("\n");
    }
}
```



```
./fileSystemOper mySystem.data dir /
Directory:
 usr    Last modification: Sat Jun  8 20:03:45 2024
File:
 file3  Last modification: Sat Jun  8 20:03:45 2024
Size: 7 Permissions: rw Password: mypassword123
```

- Lists the contents of the specified directory.
- If the specified path points to a file, it prints the file name.
- If the path points to a directory, it lists all the files and subdirectories within it.

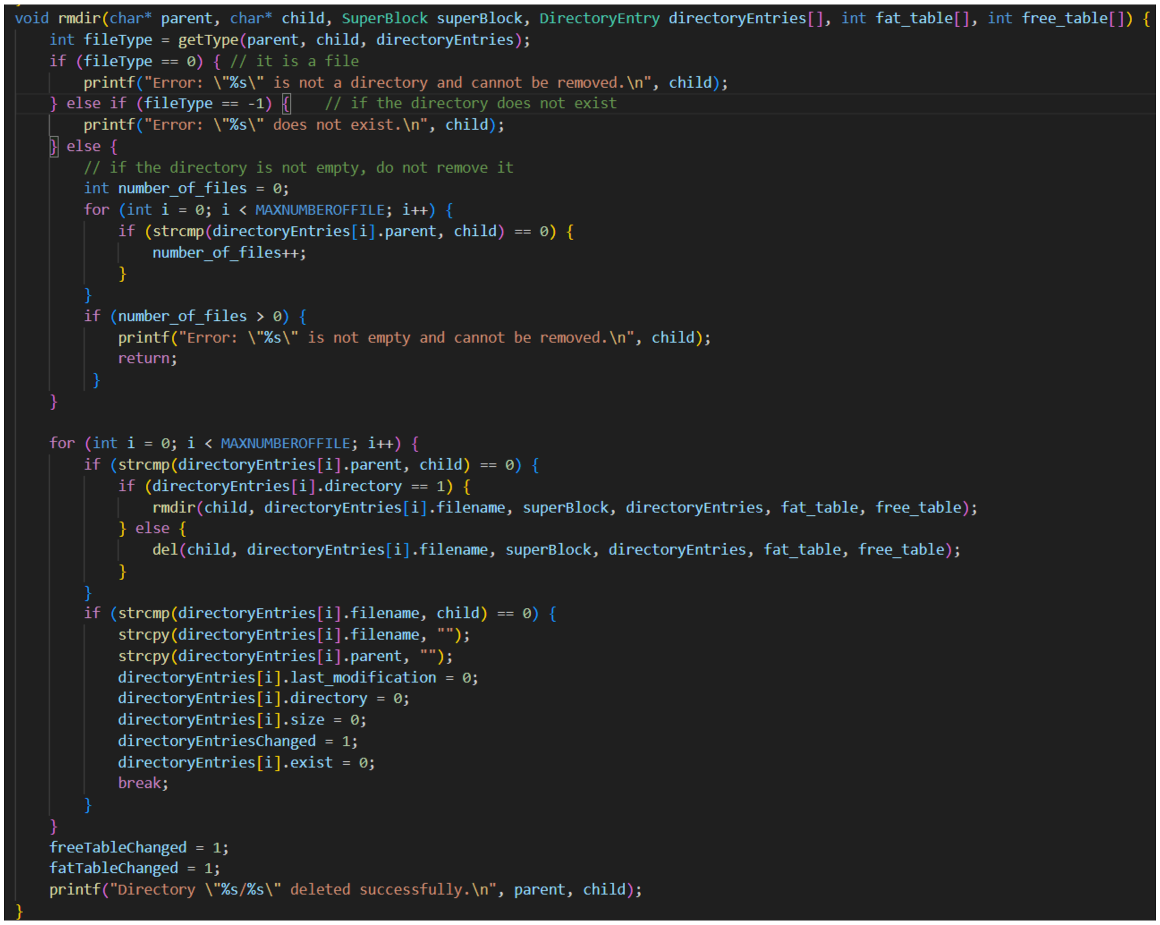
mkdir(char parent, char child, DirectoryEntry directoryEntries[])**:



```c
void mkdir(char* parent, char* child, DirectoryEntry directoryEntries[]) {
    int fileType = getType(parent, child, directoryEntries);
    if (fileType == -1) {
        for (int i = 0; i < MAXNUMBEROFFILE; i++) {
            if (directoryEntries[i].exist != 1) {
                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 = 1;
                printf("Directory created: %s/%s\n", parent, child);
                return;
            }
        }
    } else if (fileType == 0) { //  if there is a file with the same name
        printf("cannot create directory \"%s\": File exists\n", child);
    } else if (fileType == 1) {
        printf("cannot create directory \"%s\": Directory exists\n", child);
    }
}
```

- Creates a new directory under the specified parent directory.
- Checks if a directory or file with the same name already exists before creation.
- Updates the directory table with the new entry.

rmdir(char parent, char child, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], int free_table[])**:



```c
void rmdir(char* parent, char* child, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], int free_table[]) {
    int fileType = getType(parent, child, directoryEntries);
    if (fileType == 0) { // it is a file
        printf("Error: \"%s\" is not a directory and cannot be removed.\n", child);
    } else if (fileType == -1) {    // if the directory does not exist
        printf("Error: \"%s\" does not exist.\n", child);
    } else {
        // if the directory is not empty, do not remove it
        int number_of_files = 0;
        for (int i = 0; i < MAXNUMBEROFFILE; i++) {
            if (strcmp(directoryEntries[i].parent, child) == 0) {
                number_of_files++;
            }
        }
        if (number_of_files > 0) {
            printf("Error: \"%s\" is not empty and cannot be removed.\n", child);
            return;
        }
    }

    for (int i = 0; i < MAXNUMBEROFFILE; i++) {
        if (strcmp(directoryEntries[i].parent, child) == 0) {
            if (directoryEntries[i].directory == 1) {
                rmdir(child, directoryEntries[i].filename, superBlock, directoryEntries, fat_table, free_table);
            } else {
                del(child, directoryEntries[i].filename, superBlock, directoryEntries, fat_table, free_table);
            }
        }
        if (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;
        }
    }
    freeTableChanged = 1;
    fatTableChanged = 1;
    printf("Directory \"%s/%s\" deleted successfully.\n", parent, child);
}
```

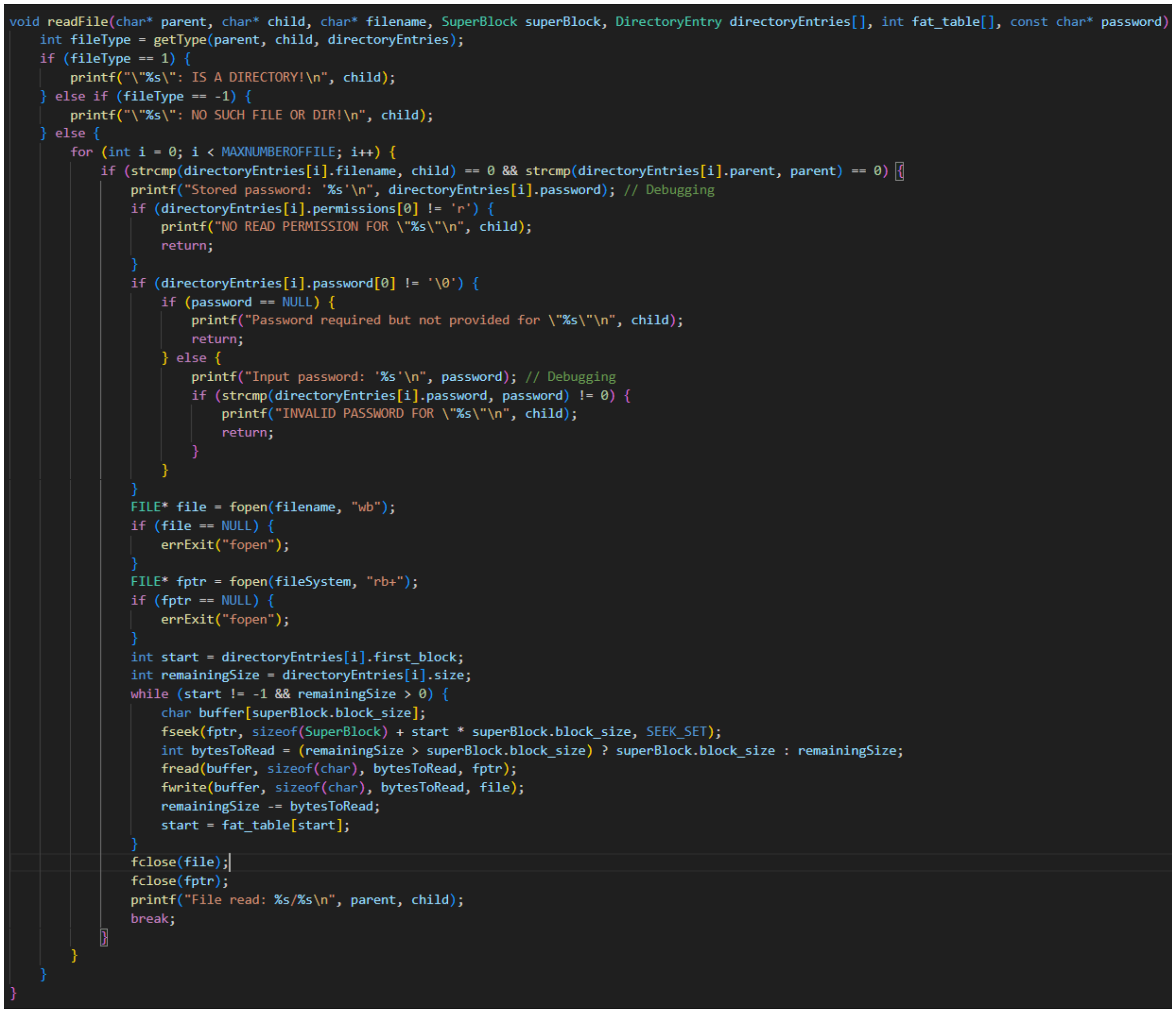
- Removes a directory and its contents.
- Ensures the directory is empty before removal.
- Updates the free block table and FAT accordingly.

File Operations

writeFile(char parent, char child, char* filename, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], int free_table[])**:

- Writes data from a specified Linux file to the file system.
- Allocates blocks in the FAT for the file data.
- Updates the directory entry with the file's size and block information.

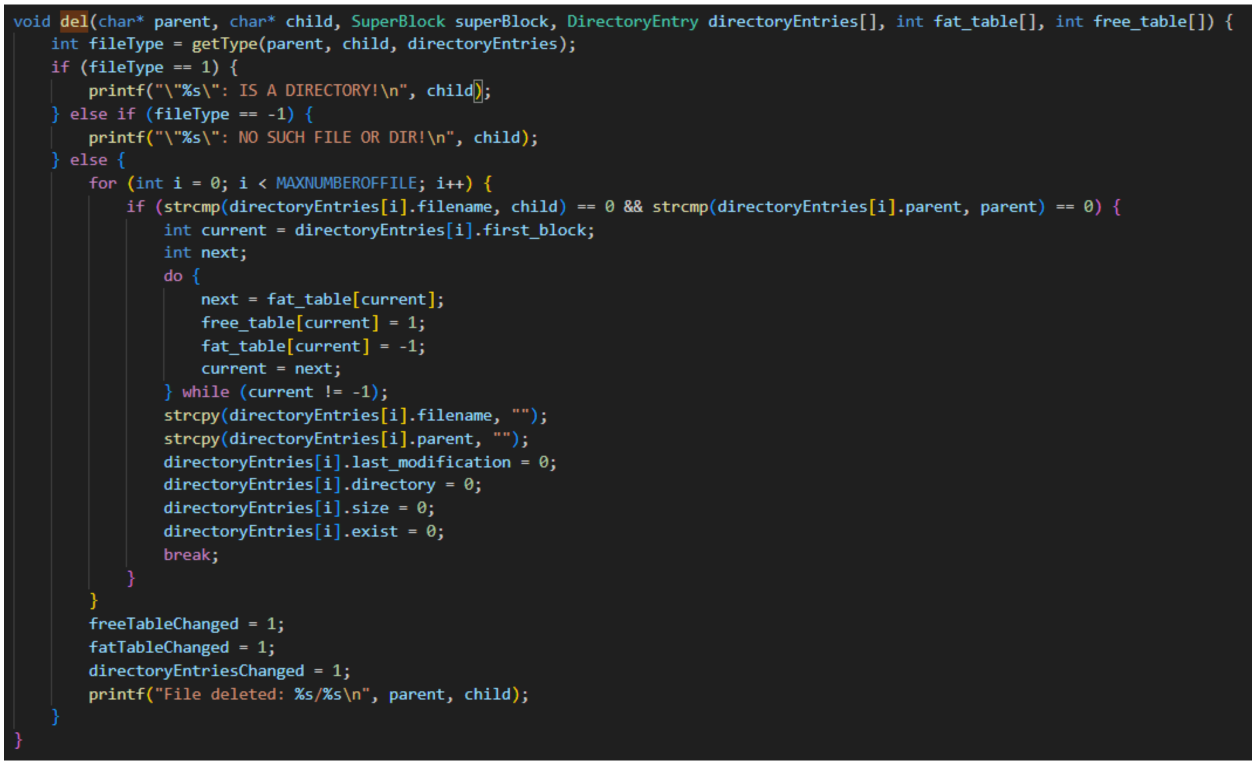
readFile(char parent, char child, char* filename, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], const char* password)**:



```c
void readFile(char* parent, char* child, char* filename, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], const char* password)
    int fileType = getType(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) {
                printf("Stored password: '%s'\n", directoryEntries[i].password); // Debugging
                if (directoryEntries[i].permissions[0] != 'r') {
                    printf("NO READ PERMISSION FOR \"%s\"\n", child);
                    return;
                }
                if (directoryEntries[i].password[0] != '\0') {
                    if (password == NULL) {
                        printf("Password required but not provided for \"%s\"\n", child);
                        return;
                    } else {
                        printf("Input password: '%s'\n", password); // Debugging
                        if (strcmp(directoryEntries[i].password, password) != 0) {
                            printf("INVALID PASSWORD FOR \"%s\"\n", child);
                            return;
                        }
                    }
                }
                FILE* file = fopen(filename, "wb");
                if (file == NULL) {
                    errExit("fopen");
                }
                FILE* fptr = fopen(fileSystem, "rb+");
                if (fptr == NULL) {
                    errExit("fopen");
                }
                int start = directoryEntries[i].first_block;
                int remainingSize = directoryEntries[i].size;
                while (start != -1 && remainingSize > 0) {
                    char buffer[superBlock.block_size];
                    fseek(fptr, sizeof(SuperBlock) + start * superBlock.block_size, SEEK_SET);
                    int bytesToRead = (remainingSize > superBlock.block_size) ? superBlock.block_size : remainingSize;
                    fread(buffer, sizeof(char), bytesToRead, fptr);
                    fwrite(buffer, sizeof(char), bytesToRead, file);
                    remainingSize -= bytesToRead;
                    start = fat_table[start];
                }
                fclose(file);
                fclose(fptr);
                printf("File read: %s/%s\n", parent, child);
                break;
            }
        }
    }
}
```

- Reads data from the file system and writes it to a specified Linux file.
- Checks for read permissions and password protection.
- Reads the file's blocks sequentially and writes the data to the Linux file.

del(char parent, char child, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], int free_table[])**:

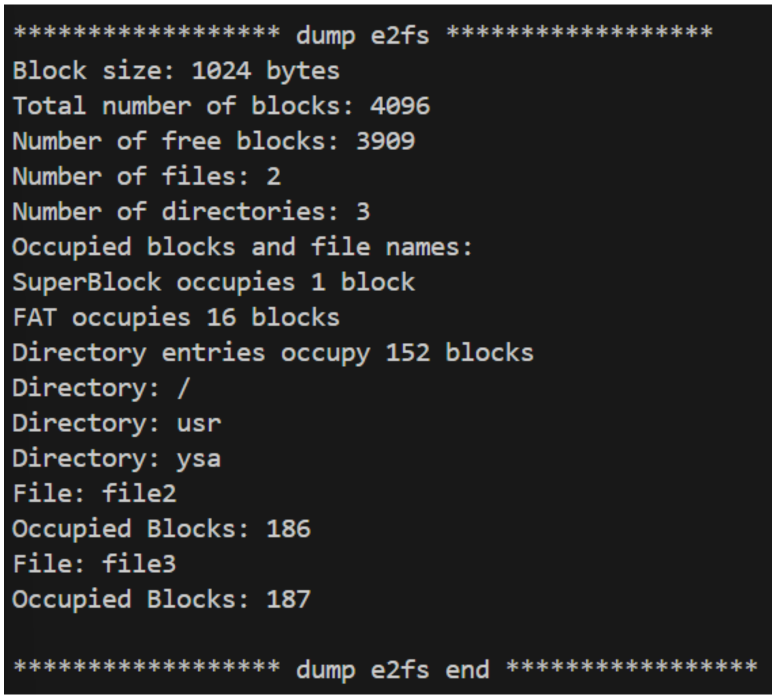


```c
void del(char* parent, char* child, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], int free_table[]) {
    int fileType = getType(parent, child, directoryEntries);
    if (fileType == 1) {
        printf("\"%s\": IS A DIRECTORY!\n", child);
    } else if (fileType == -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;
        printf("File deleted: %s/%s\n", parent, child);
    }
}
```

- Deletes a file from the file system.
- Frees the blocks allocated to the file in the FAT and updates the free block table.
- Clears the directory entry.

File System Information

dumpe2fs(SuperBlock superBlock, int free_table[], int fat_table[], DirectoryEntry directoryEntries[]):



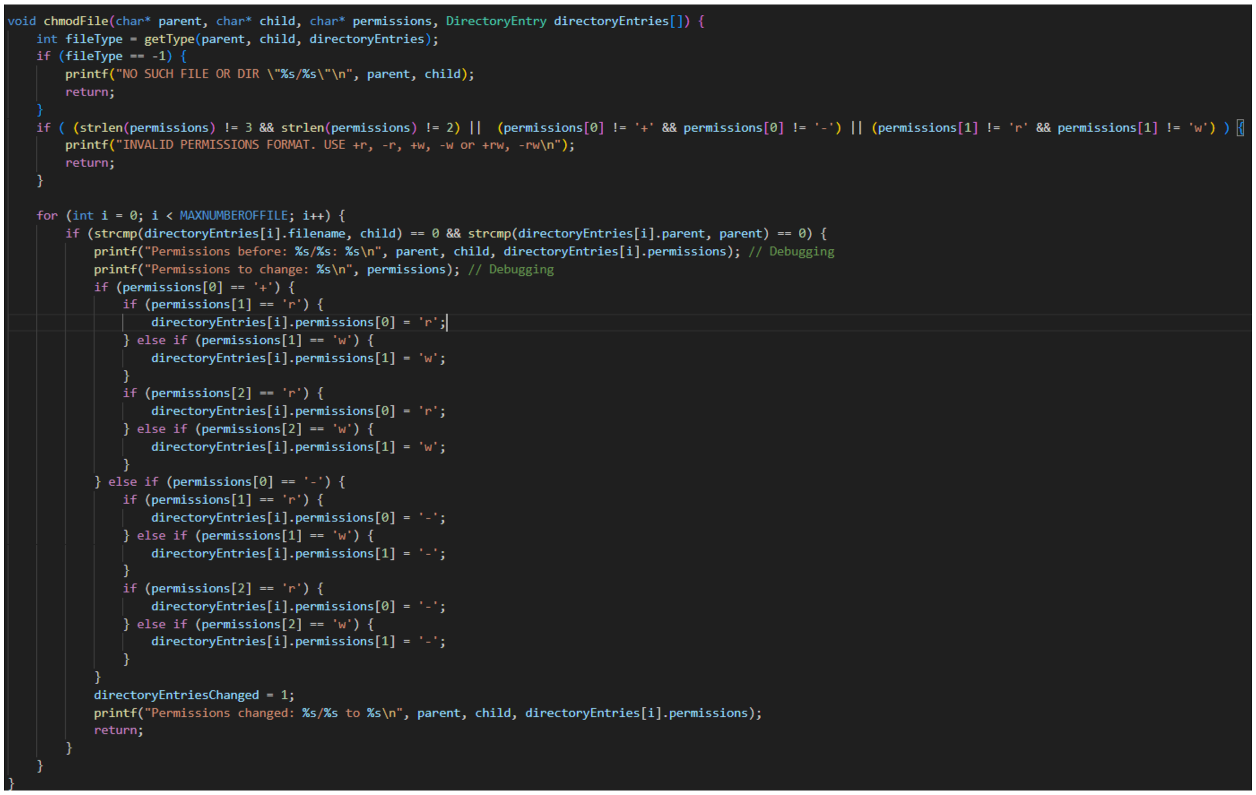
```
***************** dump e2fs *****************
Block size: 1024 bytes
Total number of blocks: 4096
Number of free blocks: 3909
Number of files: 2
Number of directories: 3
Occupied blocks and file names:
SuperBlock occupies 1 block
FAT occupies 16 blocks
Directory entries occupy 152 blocks
Directory: /
Directory: usr
Directory: ysa
File: file2
Occupied Blocks: 186
File: file3
Occupied Blocks: 187

***************** dump e2fs end ****************
```

- Displays information about the file system, including block count, free blocks, number of files and directories, and block size.
- Lists all occupied blocks and the file names associated with them.

Permission and Password Management

chmodFile(char parent, char child, char* permissions, DirectoryEntry directoryEntries[])**:

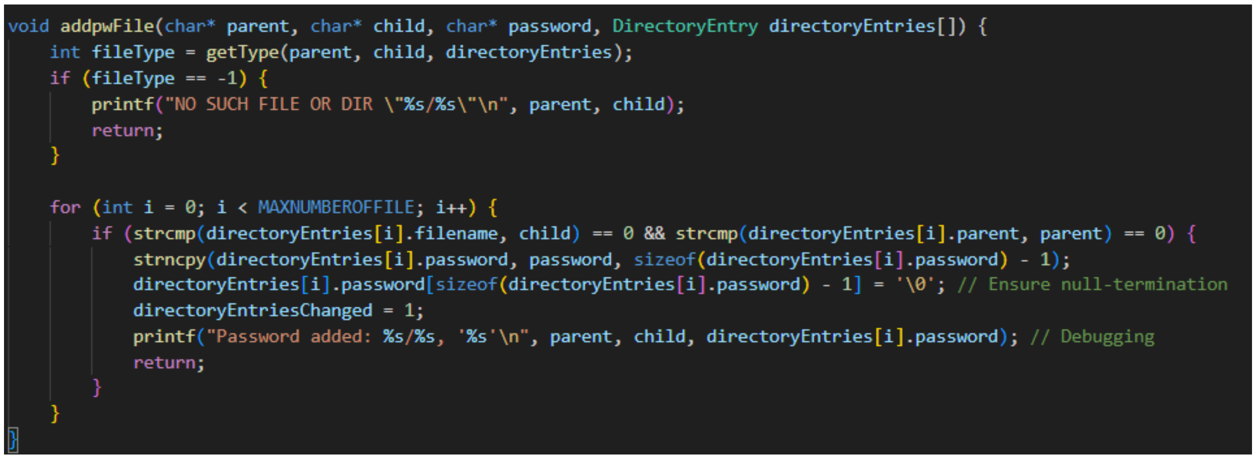


```c
void chmodFile(char* parent, char* child, char* permissions, DirectoryEntry directoryEntries[]) {
    int fileType = getType(parent, child, directoryEntries);
    if (fileType == -1) {
        printf("NO SUCH FILE OR DIR \"%s/%s\"\n", parent, child);
        return;
    }
    if ( (strlen(permissions) != 3 && strlen(permissions) != 2) ||  (permissions[0] != '+' && permissions[0] != '-') || (permissions[1] != 'r' && permissions[1] != 'w') ) {
        printf("INVALID PERMISSIONS FORMAT. USE +r, -r, +w, -w or +rw, -rw\n");
        return;
    }

    for (int i = 0; i < MAXNUMBEROFFILE; i++) {
        if (strcmp(directoryEntries[i].filename, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0) {
            printf("Permissions before: %s/%s: %s\n", parent, child, directoryEntries[i].permissions); // Debugging
            printf("Permissions to change: %s\n", permissions); // Debugging
            if (permissions[0] == '+') {
                if (permissions[1] == 'r') {
                    directoryEntries[i].permissions[0] = 'r';
                } else if (permissions[1] == 'w') {
                    directoryEntries[i].permissions[1] = 'w';
                }
                if (permissions[2] == 'r') {
                    directoryEntries[i].permissions[0] = 'r';
                } else if (permissions[2] == 'w') {
                    directoryEntries[i].permissions[1] = 'w';
                }
            } else if (permissions[0] == '-') {
                if (permissions[1] == 'r') {
                    directoryEntries[i].permissions[0] = '-';
                } else if (permissions[1] == 'w') {
                    directoryEntries[i].permissions[1] = '-';
                }
                if (permissions[2] == 'r') {
                    directoryEntries[i].permissions[0] = '-';
                } else if (permissions[2] == 'w') {
                    directoryEntries[i].permissions[1] = '-';
                }
            }
            directoryEntriesChanged = 1;
            printf("Permissions changed: %s/%s to %s\n", parent, child, directoryEntries[i].permissions);
            return;
        }
    }
}
```

- Changes the owner permissions of a specified file or directory.
- Validates the permission format before applying changes.

1. *addpwFile(char parent, char* child, char* password, DirectoryEntry directoryEntries[])**:



```c
void addpwFile(char* parent, char* child, char* password, DirectoryEntry directoryEntries[]) {
    int fileType = getType(parent, child, directoryEntries);
    if (fileType == -1) {
        printf("NO SUCH FILE OR DIR \"%s/%s\"\n", parent, child);
        return;
    }

    for (int i = 0; i < MAXNUMBEROFFILE; i++) {
        if (strcmp(directoryEntries[i].filename, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0) {
            strncpy(directoryEntries[i].password, password, sizeof(directoryEntries[i].password) - 1);
            directoryEntries[i].password[sizeof(directoryEntries[i].password) - 1] = '\0'; // Ensure null-termination
            directoryEntriesChanged = 1;
            printf("Password added: %s/%s, '%s'\n", parent, child, directoryEntries[i].password); // Debugging
            return;
        }
    }
}
```

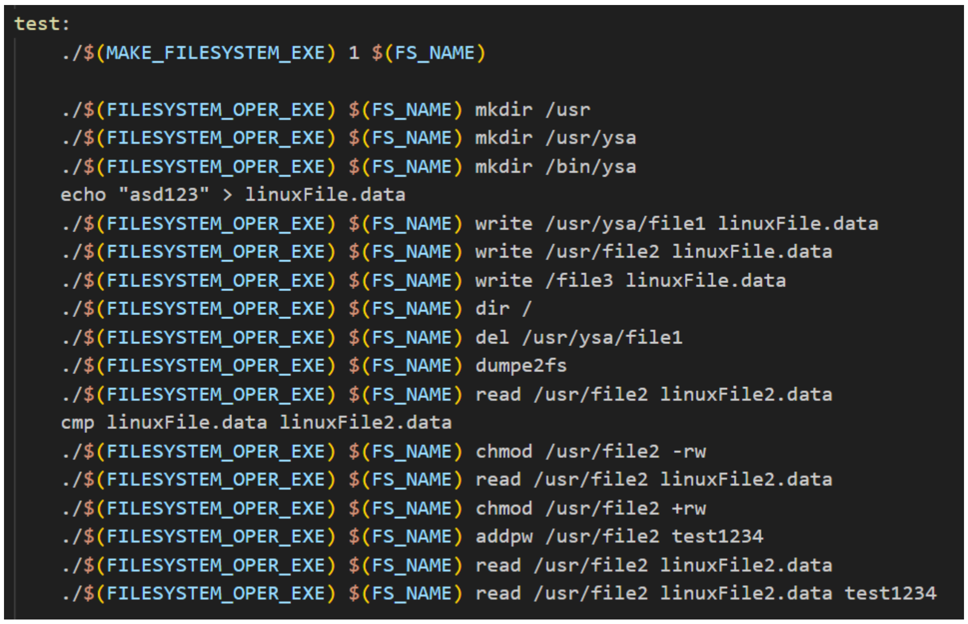
- Adds or modifies the password for a specified file.
- Ensures the password is securely stored in the directory entry.

Conclusion

This design report provides a detailed overview of the structures and functions used to implement a simplified FAT12-like file system. The system supports basic file and directory operations, manages permissions and password protection, and maintains information about the file system state. The implementation ensures that key concepts of file systems are covered, providing a solid foundation for understanding more advanced file system designs.

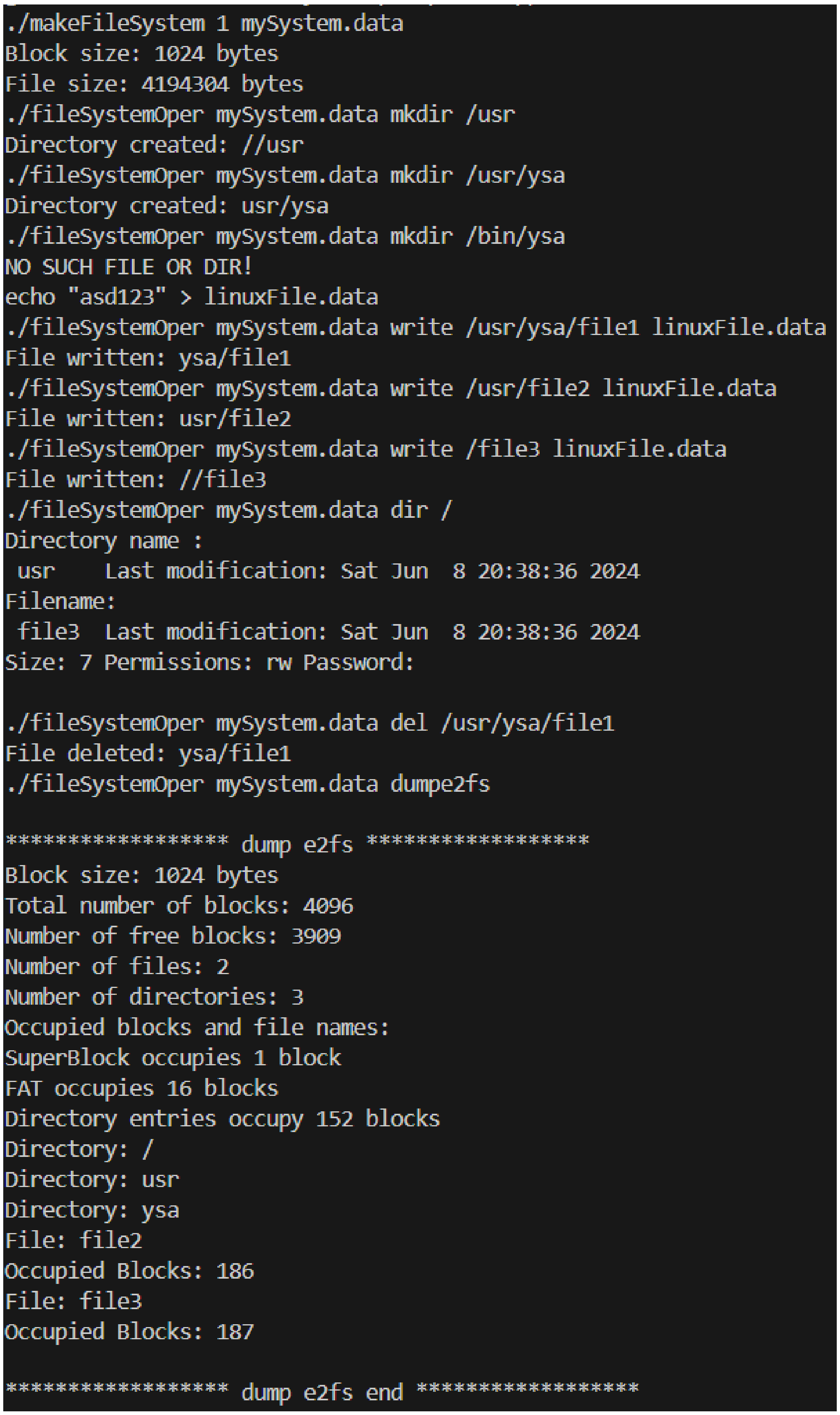
OUTPUT:

my makefile :



```
test:
	./$(MAKE_FILESYSTEM_EXE) 1 $(FS_NAME)

	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) mkdir /usr
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) mkdir /usr/ysa
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) mkdir /bin/ysa
	echo "asd123" > linuxFile.data
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) write /usr/ysa/file1 linuxFile.data
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) write /usr/file2 linuxFile.data
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) write /file3 linuxFile.data
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) dir /
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) del /usr/ysa/file1
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) dumpe2fs
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) read /usr/file2 linuxFile2.data
	cmp linuxFile.data linuxFile2.data
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) chmod /usr/file2 -rw
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) read /usr/file2 linuxFile2.data
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) chmod /usr/file2 +rw
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) addpw /usr/file2 test1234
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) read /usr/file2 linuxFile2.data
	./$(FILESYSTEM_OPER_EXE) $(FS_NAME) read /usr/file2 linuxFile2.data test1234
```

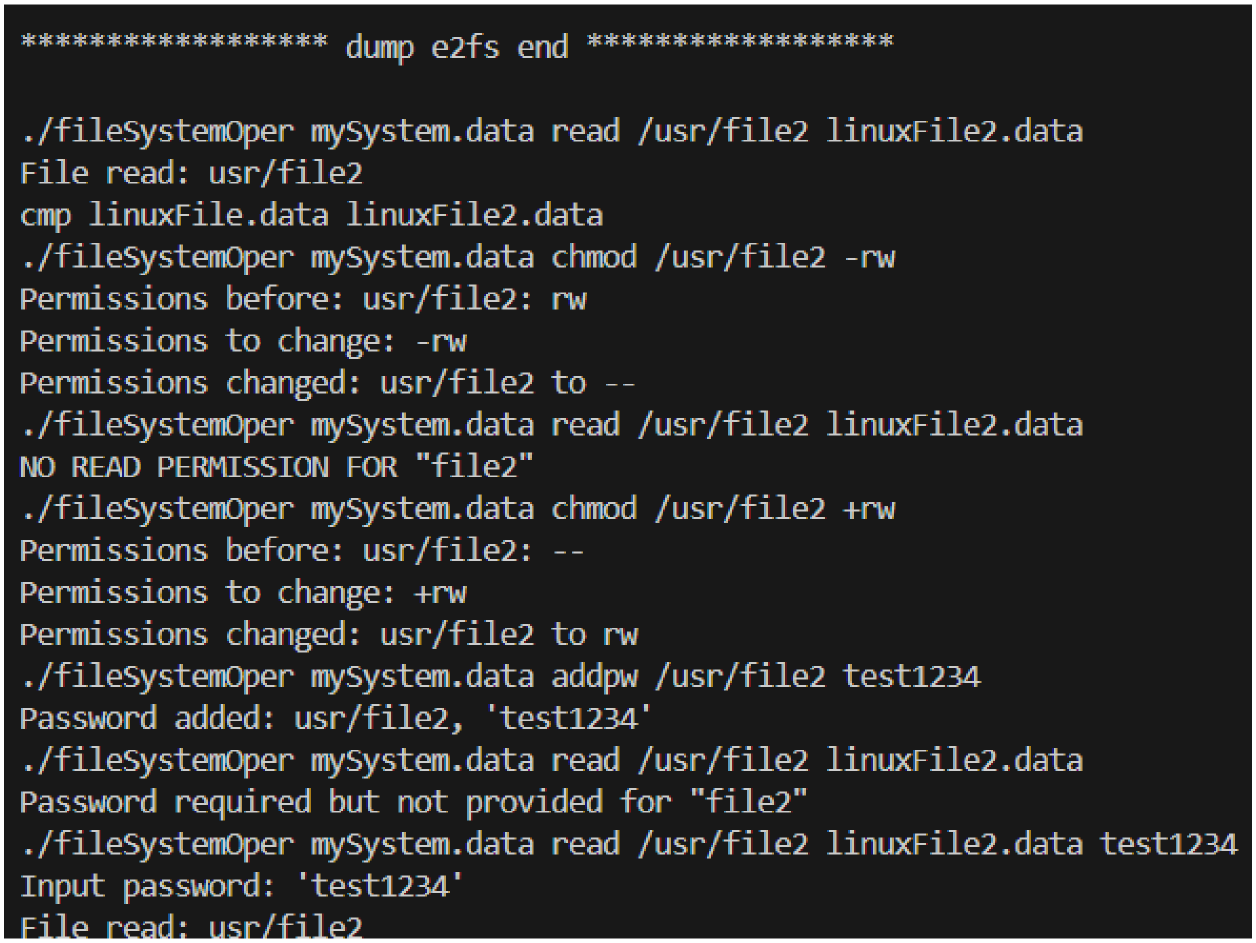


```
./makeFileSystem 1 mySystem.data
Block size: 1024 bytes
File size: 4194304 bytes
./fileSystemOper mySystem.data mkdir /usr
Directory created: //usr
./fileSystemOper mySystem.data mkdir /usr/ysa
Directory created: usr/ysa
./fileSystemOper mySystem.data mkdir /bin/ysa
NO SUCH FILE OR DIR!
echo "asd123" > linuxFile.data
./fileSystemOper mySystem.data write /usr/ysa/file1 linuxFile.data
File written: ysa/file1
./fileSystemOper mySystem.data write /usr/file2 linuxFile.data
File written: usr/file2
./fileSystemOper mySystem.data write /file3 linuxFile.data
File written: //file3
./fileSystemOper mySystem.data dir /
Directory name :
 usr    Last modification: Sat Jun  8 20:38:36 2024
Filename:
 file3  Last modification: Sat Jun  8 20:38:36 2024
Size: 7 Permissions: rw Password:

./fileSystemOper mySystem.data del /usr/ysa/file1
File deleted: ysa/file1
./fileSystemOper mySystem.data dumpe2fs

****************** dump e2fs ******************
Block size: 1024 bytes
Total number of blocks: 4096
Number of free blocks: 3909
Number of files: 2
Number of directories: 3
Occupied blocks and file names:
SuperBlock occupies 1 block
FAT occupies 16 blocks
Directory entries occupy 152 blocks
Directory: /
Directory: usr
Directory: ysa
File: file2
Occupied Blocks: 186
File: file3
Occupied Blocks: 187

****************** dump e2fs end ******************
```



```
****************** dump e2fs end ******************

./fileSystemOper mySystem.data read /usr/file2 linuxFile2.data
File read: usr/file2
cmp linuxFile.data linuxFile2.data
./fileSystemOper mySystem.data chmod /usr/file2 -rw
Permissions before: usr/file2: rw
Permissions to change: -rw
Permissions changed: usr/file2 to --
./fileSystemOper mySystem.data read /usr/file2 linuxFile2.data
NO READ PERMISSION FOR "file2"
./fileSystemOper mySystem.data chmod /usr/file2 +rw
Permissions before: usr/file2: --
Permissions to change: +rw
Permissions changed: usr/file2 to rw
./fileSystemOper mySystem.data addpw /usr/file2 test1234
Password added: usr/file2, 'test1234'
./fileSystemOper mySystem.data read /usr/file2 linuxFile2.data
Password required but not provided for "file2"
./fileSystemOper mySystem.data read /usr/file2 linuxFile2.data test1234
Input password: 'test1234'
File read: usr/file2
```