CSE312 OPERATING SYSTEMS

HOMEWORK 2

REPORT

Burak Demirkaya

210104004274

07/06/2024

1. Design Decisions

Directory Table and Directory Entries

The directory table is represented as an array of DirectoryEntry structures, with a maximum of MAXFILES entries. Each DirectoryEntry represents a file or directory and contains the following fields:

```
struct DirectoryEntry{
    char* name;
    char* parent;
    int size;
    char permission[2]; // r, w
    time_t lastModified;
    time_t created;
    char password[32];
    int isDirectory;
    int exist;
    int startBlock;
```

```
struct SuperBlock{
    int blockSize;      // Block size in bytes (0.5 KB or 1 KB)
    int totalBlocks;    // Total number of blocks in the file system
    int freeBlocks;     // Number of free blocks
    int fatBlocks;      // Number of blocks used by the FAT
    int directoryBlocks; // Number of blocks used by the directory
```

Free Blocks

Free blocks are kept in a free_table array, where each bit corresponds to a block in the file system. The bit is 1 if the block is free and 0 if it's not. This allows us to efficiently find free blocks and mark blocks as used or free.

Arbitrary Length of File Names

File names are stored as dynamically allocated strings in the name field of the DirectoryEntry structure. Handling of arbitrary length of file names is achieved through the following steps:

```
int nameLength = strlen(empty[i].name) + 1;
int parentLength = strlen(empty[i].parent) + 1;
fwrite(&nameLength, sizeof(int), 1, fileSystem);
fwrite(empty[i].name, nameLength, 1, fileSystem);
fwrite(&parentLength, sizeof(int), 1, fileSystem);
fwrite(empty[i].parent, parentLength, 1, fileSystem);
```

1. The length of the root directory's name is calculated using strlen(rootName) + 1.

2. This length is then written to the file system. This step is crucial as it allows the program to know how many characters to read for the root directory's name when the file system is read.
3. Finally, the actual name of the directory is written to the file system. The previously stored length (nameLength) is used to determine how many characters to write.

This approach allows for file names of arbitrary length, as the length of each name is stored before the name itself.

Permissions

Permissions are handled using the permission field of the DirectoryEntry structure. This field is an array of two characters, where the first character is 'R' if the file or directory entry is readable and the second character is 'W' if the file or directory entry is writable.

Password Protection

Password protection is handled using the password field of the DirectoryEntry structure. This field is a string that stores the password for the file or directory. If the password is an empty string, the file or directory is not password protected. To access a password protected file or directory, the user must enter the correct password.

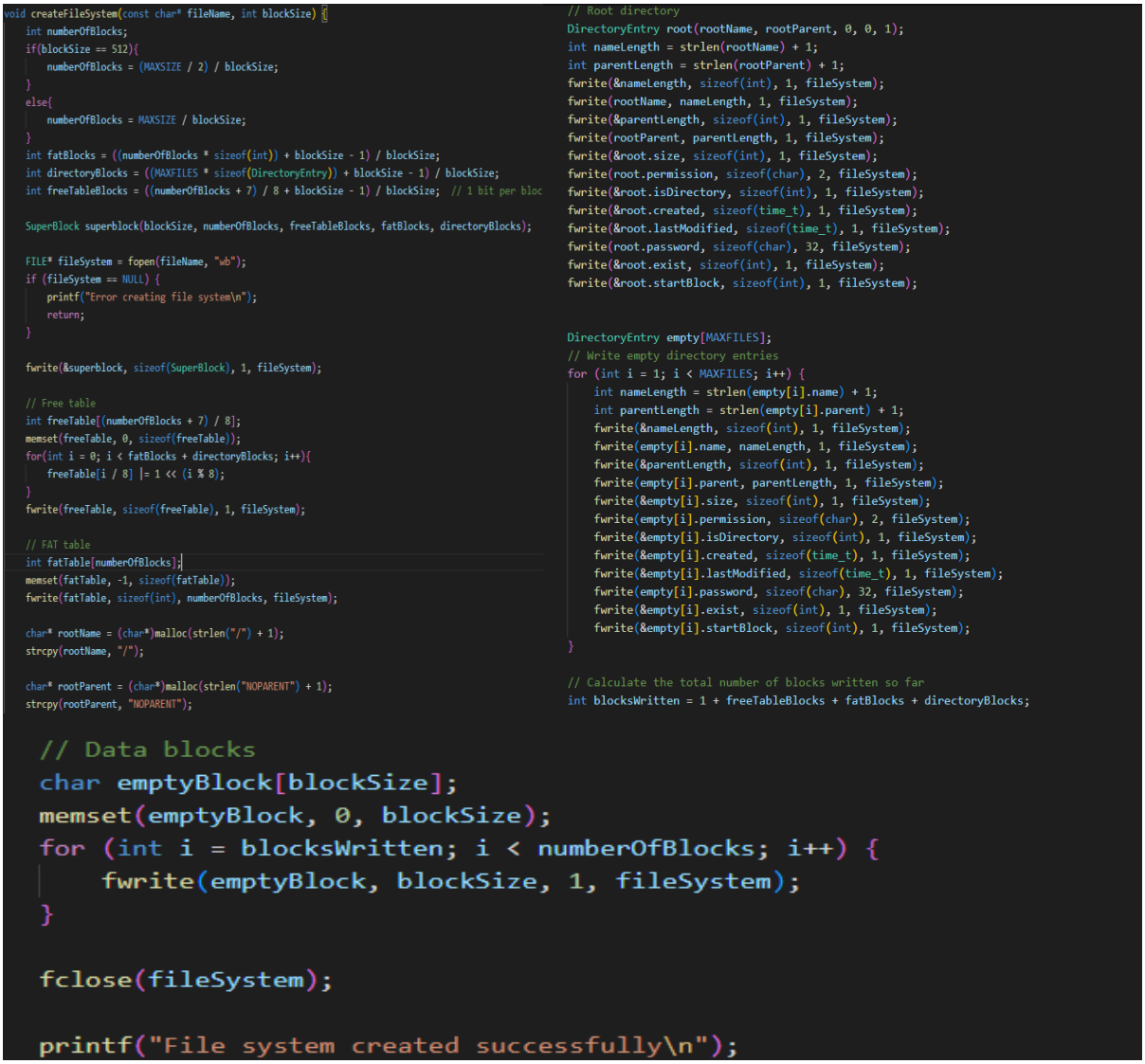
2. Creating FileSystem

A file system is created with a specified block size. The number of blocks is calculated based on the block size and the maximum size of the file system.

The function then creates a superblock, free table, FAT table, and root directory, and writes them to the file system. The superblock contains metadata about the file system, such as the block size, number of blocks, and the number of blocks allocated to the free table, FAT table, and directory entries.

The free table is a bitmap that keeps track of which blocks are free or occupied. The FAT table is an array that maps each block to the next block in the file, allowing for files to be split across non-contiguous blocks.

The root directory is created with the name "/" and no parent. The lengths of the root directory's name and its parent's name are calculated and written to the file system before the names themselves, allowing for names of arbitrary length. Finally, empty directory entries and data blocks are written to fill up the rest of the file system.



```
void createFileSystem(const char* fileName, int blockSize) {
    int numberOfBlocks;
    if(blockSize == 512){
        numberOfBlocks = (MAXSIZE / 2) / blockSize;
    }
    else{
        numberOfBlocks = MAXSIZE / blockSize;
    }
    int fatBlocks = ((numberOfBlocks * sizeof(int)) + blockSize - 1) / blockSize;
    int directoryBlocks = ((MAXFILES * sizeof(DirectoryEntry)) + blockSize - 1) / blockSize;
    int freeTableBlocks = ((numberOfBlocks + 7) / 8 + blockSize - 1) / blockSize;  // 1 bit per bloc

    SuperBlock superblock(blockSize, numberOfBlocks, freeTableBlocks, fatBlocks, directoryBlocks);

    FILE* fileSystem = fopen(fileName, "wb");
    if (fileSystem == NULL) {
        printf("Error creating file system\n");
        return;
    }

    fwrite(&superblock, sizeof(SuperBlock), 1, fileSystem);

    // Free table
    int freeTable[(numberOfBlocks + 7) / 8];
    memset(freeTable, 0, sizeof(freeTable));
    for(int i = 0; i < fatBlocks + directoryBlocks; i++){
        freeTable[i / 8] |= 1 << (i % 8);
    }
    fwrite(freeTable, sizeof(freeTable), 1, fileSystem);

    // FAT table
    int fatTable[numberOfBlocks];
    memset(fatTable, -1, sizeof(fatTable));
    fwrite(fatTable, sizeof(int), numberOfBlocks, fileSystem);

    char* rootName = (char*)malloc(strlen("/") + 1);
    strcpy(rootName, "/");

    char* rootParent = (char*)malloc(strlen("NOPARENT") + 1);
    strcpy(rootParent, "NOPARENT");

    // Root directory
    DirectoryEntry root(rootName, rootParent, 0, 0, 1);
    int nameLength = strlen(rootName) + 1;
    int parentLength = strlen(rootParent) + 1;
    fwrite(&nameLength, sizeof(int), 1, fileSystem);
    fwrite(rootName, nameLength, 1, fileSystem);
    fwrite(&parentLength, sizeof(int), 1, fileSystem);
    fwrite(rootParent, parentLength, 1, fileSystem);
    fwrite(&root.size, sizeof(int), 1, fileSystem);
    fwrite(root.permission, sizeof(char), 2, fileSystem);
    fwrite(&root.isDirectory, sizeof(int), 1, fileSystem);
    fwrite(&root.created, sizeof(time_t), 1, fileSystem);
    fwrite(&root.lastModified, sizeof(time_t), 1, fileSystem);
    fwrite(root.password, sizeof(char), 32, fileSystem);
    fwrite(&root.exist, sizeof(int), 1, fileSystem);
    fwrite(&root.startBlock, sizeof(int), 1, fileSystem);


    DirectoryEntry empty[MAXFILES];
    // Write empty directory entries
    for (int i = 1; i < MAXFILES; i++) {
        int nameLength = strlen(empty[i].name) + 1;
        int parentLength = strlen(empty[i].parent) + 1;
        fwrite(&nameLength, sizeof(int), 1, fileSystem);
        fwrite(empty[i].name, nameLength, 1, fileSystem);
        fwrite(&parentLength, sizeof(int), 1, fileSystem);
        fwrite(empty[i].parent, parentLength, 1, fileSystem);
        fwrite(&empty[i].size, sizeof(int), 1, fileSystem);
        fwrite(empty[i].permission, sizeof(char), 2, fileSystem);
        fwrite(&empty[i].isDirectory, sizeof(int), 1, fileSystem);
        fwrite(&empty[i].created, sizeof(time_t), 1, fileSystem);
        fwrite(&empty[i].lastModified, sizeof(time_t), 1, fileSystem);
        fwrite(empty[i].password, sizeof(char), 32, fileSystem);
        fwrite(&empty[i].exist, sizeof(int), 1, fileSystem);
        fwrite(&empty[i].startBlock, sizeof(int), 1, fileSystem);
    }

    // Calculate the total number of blocks written so far
    int blocksWritten = 1 + freeTableBlocks + fatBlocks + directoryBlocks;

    // Data blocks
    char emptyBlock[blockSize];
    memset(emptyBlock, 0, blockSize);
    for (int i = blocksWritten; i < numberOfBlocks; i++) {
        fwrite(emptyBlock, blockSize, 1, fileSystem);
    }

    fclose(fileSystem);

    printf("File system created successfully\n");
```

3. File System Operations

- dir

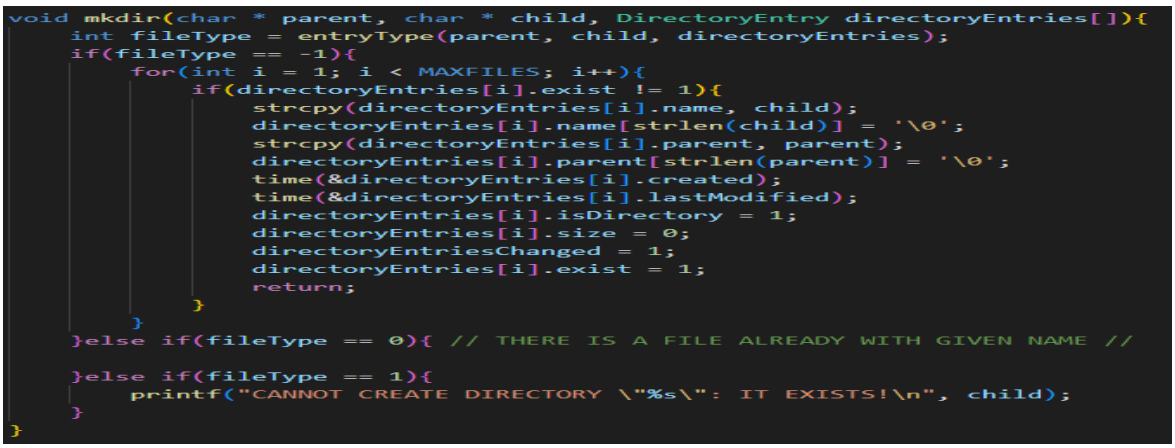
The dir function is used to display the details of a file or directory. It takes three parameters: parent, child, and directoryEntries. The function first determines the type of the child entry (file or directory). If it's a file, it prints the file name. If it's a directory, it prints the directory name and details of its contents.

```
void dir(char* parent, char* child, DirectoryEntry directoryEntries[]) {
    int fileType = entryType(parent, child, directoryEntries);
    if (fileType == 0) { // It is a file
        printf("File: %s\n", child);
    } else if (fileType == 1) { // It is a directory
        printf("Directory: %s\n", child);
        printf("-------------------------------------------------\n");
        printf("%-10s %-5s %-5s %-10s %-25s %-25s %-15s %-15s\n", "Type", "Read", "Write", "Size", "Last Modified", "Created", "Password", "Name");
        for (int i = 1; i < MAXFILES; i++) {
            if (directoryEntries[i].exist && strcmp(directoryEntries[i].parent, child) == 0) {
                char lastModified[26];
                char created[26];
                strncpy(lastModified, ctime(&directoryEntries[i].lastModified), 25);
                strncpy(created, ctime(&directoryEntries[i].created), 25);
                lastModified[24] = '\0'; // Remove newline character
                created[24] = '\0'; // Remove newline character

                printf("%-10c %-5c %-5c %-10d %-25s %-25s %-15s %-15s\n",
                    (directoryEntries[i].isDirectory == 1) ? 'd' : '-',
                    directoryEntries[i].permission[0], // Read permission
                    directoryEntries[i].permission[1], // Write permission
                    directoryEntries[i].size, // Size
                    lastModified,
                    created,
                    (directoryEntries[i].password[0] != '\0') ? "Yes" : "No",
                    directoryEntries[i].name);
            }
        }
        printf("-------------------------------------------------\n");
    } else {
        printf("Path not found: %s/%s\n", parent, child);
    }
}
```

- mkdir

The mkdir function creates a new directory. It first checks if a file or directory with the same name already exists. If not, it finds the first unused entry in the directory entries array and sets its properties to create the new directory. If a file or directory with the same name already exists, it does nothing in the case of a file, and prints an error message in the case of a directory.

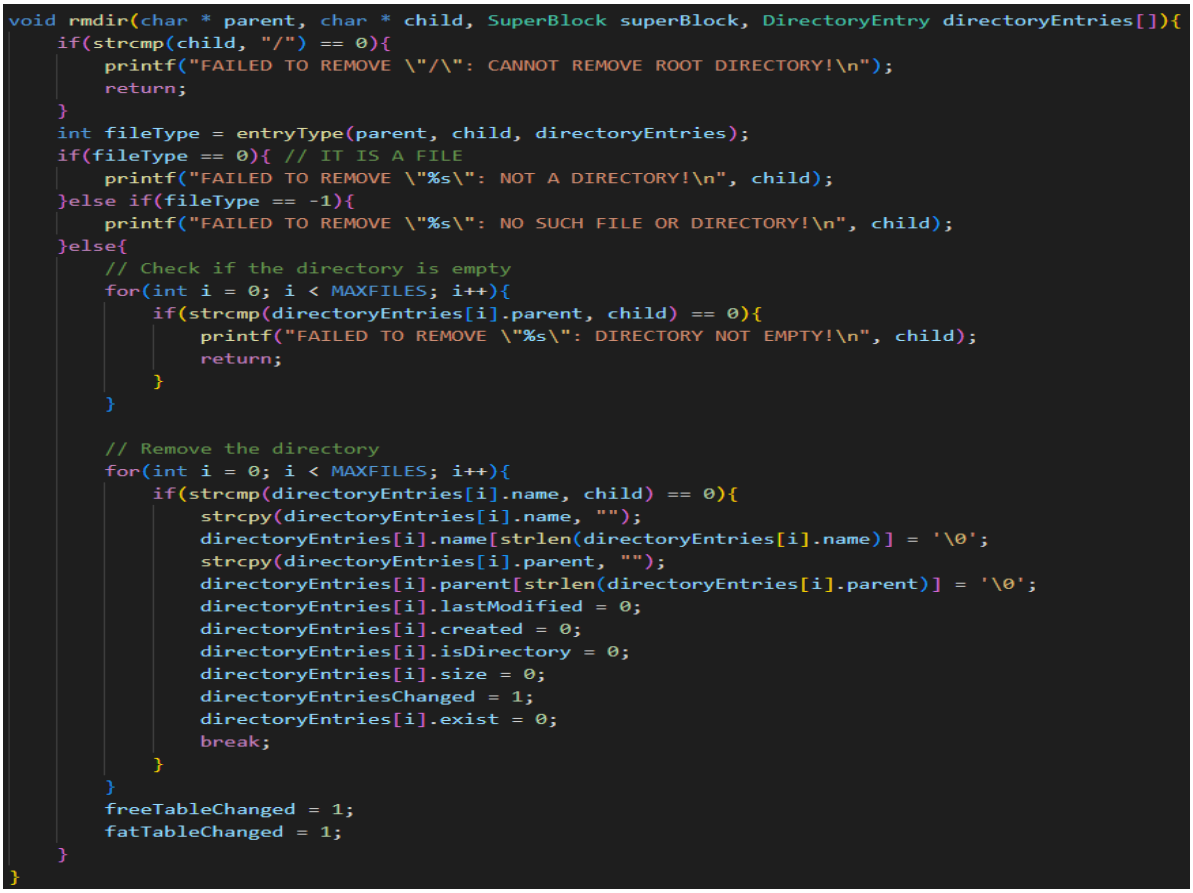


```
void mkdir(char * parent, char * child, DirectoryEntry directoryEntries[]){
    int fileType = entryType(parent, child, directoryEntries);
    if(fileType == -1){
        for(int i = 1; i < MAXFILES; i++){
            if(directoryEntries[i].exist != 1){
                strcpy(directoryEntries[i].name, child);
                directoryEntries[i].name[strlen(child)] = '\0';
                strcpy(directoryEntries[i].parent, parent);
                directoryEntries[i].parent[strlen(parent)] = '\0';
                time(&directoryEntries[i].created);
                time(&directoryEntries[i].lastModified);
                directoryEntries[i].isDirectory = 1;
                directoryEntries[i].size = 0;
                directoryEntriesChanged = 1;
                directoryEntries[i].exist = 1;
                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);
    }
}
```

- rmdir

The rmdir function removes a directory. It first checks if the directory to be removed is the root directory, a file, or does not exist, and if so, it prints an error message and returns. If the directory exists and is not the root directory, it checks if the directory is empty. If not, it prints an error message and returns. If the directory is empty, it finds the directory in the directory entries array and resets its properties to remove it.



```
void rmdir(char * parent, char * child, SuperBlock superBlock, DirectoryEntry directoryEntries[]){
    if(strcmp(child, "/") == 0){
        printf("FAILED TO REMOVE \"/\": CANNOT REMOVE ROOT DIRECTORY!\n");
        return;
    }
    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{
        // Check if the directory is empty
        for(int i = 0; i < MAXFILES; i++){
            if(strcmp(directoryEntries[i].parent, child) == 0){
                printf("FAILED TO REMOVE \"%s\": DIRECTORY NOT EMPTY!\n", child);
                return;
            }
        }

        // Remove the directory
        for(int i = 0; i < MAXFILES; i++){
            if(strcmp(directoryEntries[i].name, child) == 0){
                strcpy(directoryEntries[i].name, "");
                directoryEntries[i].name[strlen(directoryEntries[i].name)] = '\0';
                strcpy(directoryEntries[i].parent, "");
                directoryEntries[i].parent[strlen(directoryEntries[i].parent)] = '\0';
                directoryEntries[i].lastModified = 0;
                directoryEntries[i].created = 0;
                directoryEntries[i].isDirectory = 0;
                directoryEntries[i].size = 0;
                directoryEntriesChanged = 1;
                directoryEntries[i].exist = 0;
                break;
            }
        }
        freeTableChanged = 1;
        fatTableChanged = 1;
    }
}
```

- dumpe2fs

The dumpe2fs function summarizes the file system's state. It prints the block size, total blocks, and free blocks from the super block and free table. It counts and displays directories and files from the directory entries array. Lastly, it prints each file's name and block sequence from the file allocation table

```
void dumpe2fs(SuperBlock superBlock, int free_table[], int fat_table[], DirectoryEntry directoryEntries[]){
    printf("BLOCK SIZE: %d\nNUMBER OF BLOCKS: %d\n", superBlock.blockSize, superBlock.totalBlocks);
    int freeBlocks = 0;
    int fileNumber = 0;
    int directoryNumber = 0;
    for(int i = 0; i < superBlock.totalBlocks; i++){
        if ((free_table[i / 8] & (1 << (i % 8))) != 0) { // Check if the block is free
            freeBlocks++;
        }
    }
    printf("FREE BLOCKS NUMBER: %d\n", freeBlocks);

    for(int i = 0; i < MAXFILES; i++){
        if(directoryEntries[i].exist == 1){
            if(directoryEntries[i].isDirectory == 1){ // IT IS A DIRECTORY
                directoryNumber++;
            }else{ // IT IS A FILE
                fileNumber++;
            }
        }
    }

    printf("NUMBER OF FILES: %d\n", fileNumber);
    printf("NUMBER OF DIRECTORIES: %d\n", directoryNumber);

    for(int i = 0; i < MAXFILES; i++){ // PRINT OCCUPIED BLOCKS FOR FILES
        if(directoryEntries[i].exist == 1 && directoryEntries[i].isDirectory == 0){
            printf("OCCUPIED BLOCKS FOR %s: ", directoryEntries[i].name);
            int start = directoryEntries[i].startBlock;
            while(start != -1){
                printf("%d ", start);
                start = fat_table[start];
            }
            printf("\n");
        }
    }
}
```

- write

The write function is used to write a file to file system.

The function first checks if the file already exists in the directory. If it does, it checks the write permissions of the file. If the file does not have write permissions ('W'), the function prints a message and returns without making any changes.

The function also checks if the file is password protected. If a password is required and either no password is provided or the provided password does not match the file's password, the function prints a message and returns without making any changes.

If the file exists, has write permissions, and either is not password protected or the correct password is provided, the function deletes the existing file before proceeding with the write operation.

After writing the file, the function updates the directory entry for the file, including setting its permissions. The permissions are retrieved using the stat function, which stores the file's information in a struct stat variable. The read and write permissions are then set based on the st_mode member of this variable.

```
void write(char * parent, char * child, char * filename, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], int free_table[], char* password) {
    int fileType = entryType(parent, child, directoryEntries);
    int dirIndex = -1;

    // Check if the child is a directory
    if (fileType == 1) {
        printf("\"%s\": IS A DIRECTORY!\n", child);
        return;
    }

    for (int i = 0; i < MAXFILES; i++) {
        if ((directoryEntries[i].exist == 1) && strcmp(directoryEntries[i].name, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0) {
            dirIndex = i;
            break;
        }
    }

    // Check for write permissions and password protection if the file exists
    if (dirIndex != -1) {
        char writePermission = directoryEntries[dirIndex].permission[1];
        if (writePermission != 'W') {
            printf("No write permission for \"%s\"\n", child);
            return;
        }
        if (directoryEntries[dirIndex].password[0] != '\0') {
            if (password == NULL) {
                printf("Password required for \"%s\"\n", child);
                return;
            }
            if (strcmp(password, directoryEntries[dirIndex].password) != 0) {
                printf("Incorrect password for \"%s\"\n", child);
                return;
            }
        }
        del(parent, child, superBlock, directoryEntries, fat_table, free_table); // Delete the file
    }
```

```
// Open the file to read
FILE * file = fopen(filename, "rb");
if (file == NULL) {
    perror("fopen");
    exit(-1);
}

// Get the size of the file
fseek(file, 0, SEEK_END);
int size = ftell(file);
rewind(file);

// Calculate the number of blocks needed for the file
int file_blocks = (size + superBlock.blockSize - 1) / superBlock.blockSize;

// Allocate memory to read the file into blocks
char ** fileArray = (char**) malloc(file_blocks * sizeof(char*));
for (int i = 0; i < file_blocks; i++) {
    fileArray[i] = (char*) malloc(superBlock.blockSize * sizeof(char));
    int bytesRead = fread(fileArray[i], sizeof(char), superBlock.blockSize, file);
    if (bytesRead < superBlock.blockSize) {
        memset(fileArray[i] + bytesRead, 0, superBlock.blockSize - bytesRead);
    }
}
```

```
// Prepare to write: find free blocks and update FAT
int remainingSize = size;
int firstBlock = -1, lastBlock = -1;

for (int i = 0; i < superBlock.totalBlocks && remainingSize > 0; i++) {
    if ((free_table[i / 8] & (1 << (i % 8))) == 0) { // FREE BLOCK IS FOUND
        free_table[i / 8] |= (1 << (i % 8)); // Mark block as used
        if (firstBlock == -1) {
            firstBlock = i;
        }
        if (lastBlock != -1) {
            fat_table[lastBlock] = i;
        }
        lastBlock = i;
        remainingSize -= superBlock.blockSize;
    }
}

// If not enough space
if (remainingSize > 0) {
    printf("SOME PARTS OF THE FILE IS NOT WRITTEN TO \"%s\": MEMORY IS FULL!\n", child);
    fclose(file);
    return;
}

// Write the file blocks to the filesystem
FILE * fs = fopen(fileSystemName, "rb+");
if (fs == NULL) {
    perror("fopen");
    exit(-1);
}

int currentBlock = firstBlock;
for (int i = 0; i < file_blocks; i++) {
    int position = sizeof(SuperBlock) + currentBlock * superBlock.blockSize;
    fseek(fs, position, SEEK_SET);
    fwrite(fileArray[i], sizeof(char), superBlock.blockSize, fs);
    currentBlock = fat_table[currentBlock];
}
```

```
// Set -1 to the last block
if (lastBlock != -1) {
    fat_table[lastBlock] = -1;
}

// Get the file's permissions
struct stat st;
stat(filename, &st);

// Update the directory entry for the new file
if (dirIndex == -1) {
    for (int i = 0; i < MAXFILES; i++) {
        if (directoryEntries[i].exist != 1) {
            directoryEntries[i].name = new char[strlen(child) + 1];
            directoryEntries[i].parent = new char[strlen(parent) + 1];
            strcpy(directoryEntries[i].name, child);
            strcpy(directoryEntries[i].parent, parent);
            time(&directoryEntries[i].lastModified);
            time(&directoryEntries[i].created);
            directoryEntries[i].isDirectory = 0;
            directoryEntries[i].size = size;
            directoryEntries[i].startBlock = firstBlock;
            directoryEntries[i].exist = 1;
            directoryEntries[i].permission[0] = (st.st_mode & S_IRUSR) ? 'R' : '-'; // Read permission
            directoryEntries[i].permission[1] = (st.st_mode & S_IWUSR) ? 'W' : '-'; // Write permission
            break;
        }
    }
}
// Clean up and close files
fclose(file);
fclose(fs);
for (int i = 0; i < file_blocks; i++) {
    free(fileArray[i]);
}
free(fileArray);
freeTableChanged = 1;
fatTableChanged = 1;
directoryEntriesChanged = 1;
```

- read

The read function is used to read a file from a custom file system.

The function first determines the type of the entry (file or directory. If the entry is a directory or does not exist, it prints a message and returns.

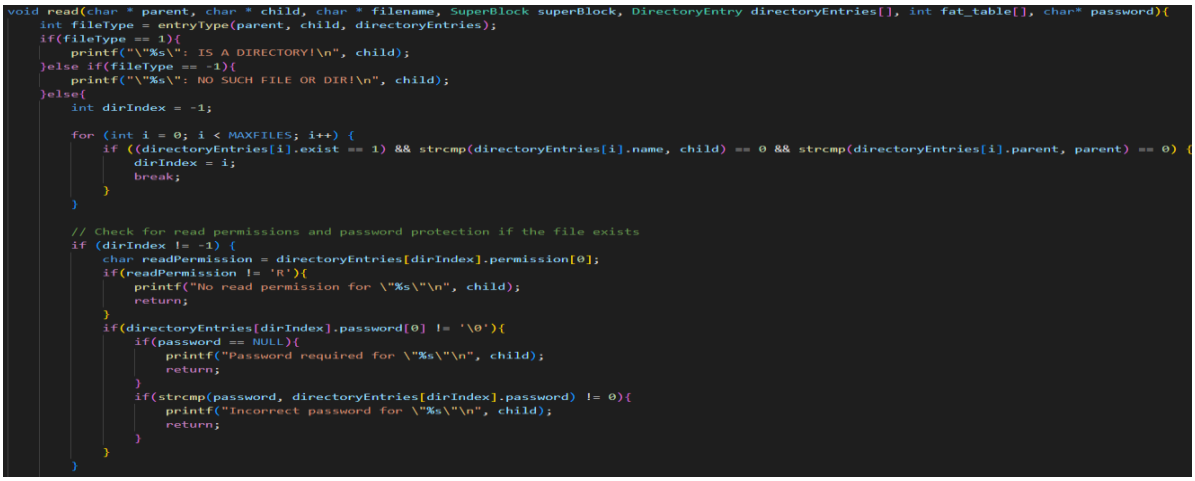
If the entry is a file, it finds the directory entry for the file. If the file exists, it checks the read permissions of the file. If the file does not have read permissions ('R'), the function prints a message and returns.

The function also checks if the file is password protected. If a password is required and either no password is provided or the provided password does not match the file's password, the function prints a message and returns.

If the file has read permissions and either is not password protected or the correct password is provided, the function reads the file from the file system and writes it to a local file. The local file is opened in binary write mode, and the file system is opened in binary read/write mode.

The function reads the file from the file system block by block, using the File Allocation Table (FAT) to find the next block of the file. It writes each block to the local file until it has read the entire file.

After reading the file, the function sets the permissions of the local file to match the permissions of the file in the file system. The permissions are set using the chmod function, which changes the permissions of a file. The read and write permissions are set based on the permission member of the directory entry for the file.



```
void read(char * parent, char * child, char * filename, SuperBlock superBlock, DirectoryEntry directoryEntries[], int fat_table[], char* password){
    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{
        int dirIndex = -1;

        for (int i = 0; i < MAXFILES; i++) {
            if ((directoryEntries[i].exist == 1) && strcmp(directoryEntries[i].name, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0) {
                dirIndex = i;
                break;
            }
        }

        // Check for read permissions and password protection if the file exists
        if (dirIndex != -1) {
            char readPermission = directoryEntries[dirIndex].permission[0];
            if(readPermission != 'R'){
                printf("No read permission for \"%s\"\n", child);
                return;
            }
            if(directoryEntries[dirIndex].password[0] != '\0'){
                if(password == NULL){
                    printf("Password required for \"%s\"\n", child);
                    return;
                }
                if(strcmp(password, directoryEntries[dirIndex].password) != 0){
                    printf("Incorrect password for \"%s\"\n", child);
                    return;
                }
            }
        }
```

```
FILE * file = fopen(filename, "wb");  // Open the file in binary write mode
if(file == NULL){
    perror("fopen");
    return;
}
FILE * fs = fopen(fileSystemName, "rb+"); // Open the file system in read binary mode
if(fs == NULL){
    perror("fopen");
    return;
}
for(int i = 0; i < MAXFILES; i++){
    if(strcmp(directoryEntries[i].name, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0){
        int start = directoryEntries[i].startBlock;

        mode_t mode = 0;
        if(directoryEntries[i].permission[0] == 'R'){
            mode |= S_IRUSR;
        }
        if(directoryEntries[i].permission[1] == 'W'){
            mode |= S_IWUSR;
        }

        if(chmod(filename, mode) == -1){
            perror("chmod");
            return;
        }
            int fileSize = directoryEntries[i].size;
            int remainingSize = fileSize;

            while(start != -1 && remainingSize > 0){
                char buffer[superBlock.blockSize];
                fseek(fs, sizeof(SuperBlock) + start * superBlock.blockSize, SEEK_SET); // Seek to the start block
                size_t bytesToRead = (remainingSize > superBlock.blockSize) ? superBlock.blockSize : remainingSize; // Read the block
                int bytesRead = fread(buffer, sizeof(char), bytesToRead, fs); // Write the block to the file
                fwrite(buffer, sizeof(char), bytesRead, file); // Update the remaining size
                remainingSize -= bytesRead;
                start = fat_table[start]; // Move to the next block
            }
            break;
        }
    }
    fclose(file);
    fclose(fs);
}
```

- del

The del function is used to delete a file from the file system. The function first determines the type of the entry (file or directory. If the entry is a directory, it prints a message and returns. If the entry does not exist, it also prints a message and returns. If the entry is a file, it finds the directory entry for the file. It then enters a loop where it follows the chain of blocks in the File Allocation Table (FAT) that make up the file. For each block in the chain, it frees the block in the free table and sets the corresponding entry in the FAT to -1, indicating that the block is no longer part of a file.

```
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 < MAXFILES; i++){
            if(strcmp(directoryEntries[i].name, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0){
                int current = directoryEntries[i].startBlock;
                int next;
                do{
                    next = fat_table[current];
                    free_table[current / 8] &= ~(1 << (current % 8));
                    fat_table[current] = -1;
                    current = next;

                }while(current != -1);
                    strcpy(directoryEntries[i].name, "");
                    strcpy(directoryEntries[i].parent, "");
                    directoryEntries[i].lastModified = 0;
                    directoryEntries[i].created = 0;
                    directoryEntries[i].isDirectory = 0;
                    directoryEntries[i].size = 0;
                    directoryEntries[i].exist = 0;
                break;
            }
        }
        freeTableChanged = 1;
        fatTableChanged = 1;
        directoryEntriesChanged = 1;
    }
}
```

- chmod

The chmod function is used to change the permissions of a file. The permission argument specifies the changes to the permissions. It should be a string starting with '+' (to add permissions) or '-' (to remove permissions), followed by 'r' (for read permissions) and/or 'w' (for write permissions). Here are the possible combinations:

-r: This will remove the read permission from the file.

-w: This will remove the write permission from the file.

+r: This will add the read permission to the file.

+w: This will add the write permission to the file.

+rw: This will add both the read and write permissions to the file.

-rw: This will remove both the read and write permissions from the file.

The function changes the permissions by modifying the permission field of the directory entry for the file. The permission field is a two-character array, where the first character represents the read permission and the second character represents the write permission. 'R' means the file has read permission, 'W' means the file has write permission, and '-' means the file does not have the corresponding permission.

```
void chmod(char * parent, char * child, char * permission, DirectoryEntry directoryEntries[]){
    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{ // IT IS A FILE
        for(int i = 0; i < MAXFILES; i++){
            if(strcmp(directoryEntries[i].name, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0){
                if(permission[0] == '+' || permission[0] == '-'){
                    if(permission[1] == 'r' || permission[2] == 'r'){
                        if(permission[0] == '+'){
                            directoryEntries[i].permission[0] = 'R';
                        }
                        else if(permission[0] == '-'){
                            directoryEntries[i].permission[0] = '-';
                        }
                    }
                    if(permission[1] == 'w' || permission[2] == 'w'){
                        if(permission[0] == '+'){
                            directoryEntries[i].permission[1] = 'W';
                        }
                        else if(permission[0] == '-'){
                            directoryEntries[i].permission[1] = '-';
                        }
                    }
                }
                directoryEntries[i].lastModified = time(0);
                directoryEntriesChanged = 1;
                break;
            }
        }
    }
}
```

- addpw

The addpw function is used to add a password to a file in a directory.

The function first determines the type of the entry. If the entry is a directory, it prints a message stating that the entry is a directory and returns. If the child does not exist, it prints a message stating that the child does not exist and returns. If the entry is a file, it iterates over the directoryEntries array. If it finds a match, it copies the password into the password field of the directory entry, updates the lastModified field of the directory entry to the current time.

```
void addpw(char * parent, char * child, char * password, DirectoryEntry directoryEntries[]){
    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{ // IT IS A FILE
        for(int i = 0; i < MAXFILES; i++){
            if(strcmp(directoryEntries[i].name, child) == 0 && strcmp(directoryEntries[i].parent, parent) == 0){
                strcpy(directoryEntries[i].password, password);
                directoryEntries[i].lastModified = time(0);
                directoryEntriesChanged = 1;
                break;
            }
        }
    }
}
```

After running each command depending on the data changed I write them back to the filesystem.

```
if(entryChanged == 1){
    rewind(fileSystemWrite);
    long dirEntryPos = sizeof(SuperBlock) + sizeof(freeTable) + (superblock.totalBlocks * sizeof(int));
    fseek(fileSystemWrite, dirEntryPos, SEEK_SET);
    for(int i=0; i<MAXFILES; ++i){
        int nameLength = strlen(entries[i].name) + 1;
        int parentLength = strlen(entries[i].parent) + 1;
        fwrite(&nameLength, sizeof(int), 1, fileSystemWrite);
        fwrite(entries[i].name, nameLength, 1, fileSystemWrite);
        fwrite(&parentLength, sizeof(int), 1, fileSystemWrite);
        fwrite(entries[i].parent, parentLength, 1, fileSystemWrite);
        fwrite(&entries[i].size, sizeof(int), 1, fileSystemWrite);
        fwrite(entries[i].permission, sizeof(char), 2, fileSystemWrite);
        fwrite(&entries[i].isDirectory, sizeof(int), 1, fileSystemWrite);
        fwrite(&entries[i].created, sizeof(time_t), 1, fileSystemWrite);
        fwrite(&entries[i].lastModified, sizeof(time_t), 1, fileSystemWrite);
        fwrite(entries[i].password, sizeof(char), 32, fileSystemWrite);
        fwrite(&entries[i].exist, sizeof(int), 1, fileSystemWrite);
        fwrite(&entries[i].startBlock, sizeof(int), 1, fileSystemWrite);
    }
}
if(fatTableChanged == 1){
    rewind(fileSystemWrite);

    fseek(fileSystemWrite, sizeof(SuperBlock) + (superblock.blockSize * superblock.freeBlocks), SEEK_SET);

    fwrite(fatTable, sizeof(int), superblock.totalBlocks, fileSystemWrite);
}
if(freeTableChanged == 1){
    rewind(fileSystemWrite);

    fseek(fileSystemWrite, sizeof(SuperBlock), SEEK_SET);

    fwrite(freeTable, sizeof(freeTable), 1, fileSystemWrite);
}
```

4. Test Result

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./makeFileSystem 1 mySystem.data
File system created successfully
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data mkdir "/usr"
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/"
Directory: /
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
d          R     W     0          Fri Jun  7 18:18:42 2024  Fri Jun  7 18:18:42 2024  No             usr
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data mkdir "/usr/ysa"
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/usr"
Directory: usr
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
d          R     W     0          Fri Jun  7 18:19:08 2024  Fri Jun  7 18:19:08 2024  No             ysa
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data mkdir "/bin/ysa"
NO SUCH FILE OR DIR!
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data write "/usr/ysa/file1" linuxFile.data
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/usr/ysa"
Directory: ysa
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
-          R     W     114        Fri Jun  7 18:19:48 2024  Fri Jun  7 18:19:48 2024  No             file1
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data write "/usr/file2" linuxFile.data
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/usr"
Directory: usr
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
d          R     W     0          Fri Jun  7 18:19:08 2024  Fri Jun  7 18:19:08 2024  No             ysa
-          R     W     114        Fri Jun  7 18:20:24 2024  Fri Jun  7 18:20:24 2024  No             file2
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data write "/file3" linuxFile.data
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/"
Directory: /
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
d          R     W     0          Fri Jun  7 18:18:42 2024  Fri Jun  7 18:18:42 2024  No             usr
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data del "/usr/ysa/file1"
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/usr/ysa"
Directory: ysa
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dumpe2fs
BLOCK SIZE: 1024
NUMBER OF BLOCKS: 4096
FREE BLOCKS NUMBER: 40
NUMBER OF FILES: 2
NUMBER OF DIRECTORIES: 3
OCCUPIED BLOCKS FOR file2: 39
OCCUPIED BLOCKS FOR file3: 40
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data read "/usr/file2" linuxFile2.data
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/usr"
Directory: usr
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
d          R     W     0          Fri Jun  7 18:19:08 2024  Fri Jun  7 18:19:08 2024  No             ysa
-          R     W     114        Fri Jun  7 18:20:24 2024  Fri Jun  7 18:20:24 2024  No             file2
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ cmp linuxFile.data linuxFile2.data
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data chmod "/usr/file2" -rw
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/usr"
Directory: usr
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
d          R     W     0          Fri Jun  7 18:19:08 2024  Fri Jun  7 18:19:08 2024  No             ysa
-          -     -     114        Fri Jun  7 18:22:36 2024  Fri Jun  7 18:20:24 2024  No             file2
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data read "/usr/file2" linuxFile2.data
No read permission for "file2"
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data chmod "/usr/file2" +rw
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/usr"
Directory: usr
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
d          R     W     0          Fri Jun  7 18:19:08 2024  Fri Jun  7 18:19:08 2024  No             ysa
-          R     W     114        Fri Jun  7 18:23:13 2024  Fri Jun  7 18:20:24 2024  No             file2
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data addpw "/usr/file2" test1234
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data dir "/usr"
Directory: usr
--------------------------------------------------
Type       Read  Write Size       Last Modified            Created                  Password       Name
d          R     W     0          Fri Jun  7 18:19:08 2024  Fri Jun  7 18:19:08 2024  No             ysa
-          R     W     114        Fri Jun  7 18:23:42 2024  Fri Jun  7 18:20:24 2024  Yes            file2
--------------------------------------------------
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data read "/usr/file2" linuxFile2.data
Password required for "file2"
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$ ./fileSystemOper mySystem.data read "/usr/file2" linuxFile2.data test1234
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/os/HW2$
```

After each step I entered dir function to view the corresponding directories entries. Also I used “/” for declaring paths.