CSE 312 Spring 2023

Homework 3

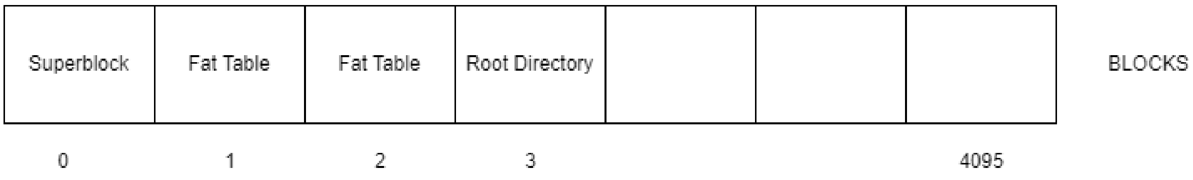
Report

1801042614

Samet Nalbant

Simplified FAT like System Design

In the homework design, since we use FAT12 like system, file system contains 2^12 blocks. According to given block size, positions of Superblock, Fat Table and root directory positions are calculated.



Superblock contains informations about system. These are the attributes that superblock has:

- Number of blocks.
- Number of free blocks.
- Block size.
- Root directory block index.

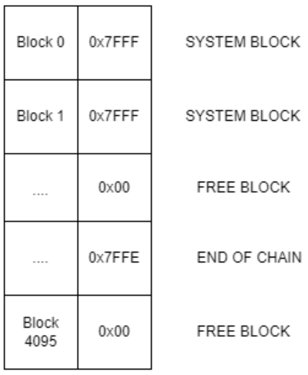
Fat table is a simple array that contains the next block number of each block number. These are the values that can fat table entry can has:

- EMPTY_BLOCK 0x00
- END_OF_CHAIN 0x7FFE
- SYSTEM_BLOCK 0x7FFF

EMPTY_BLOCK represents the free blocks to use.

END_OF_CHAIN represents the end of chain to keep end of files that has multiple blocks.

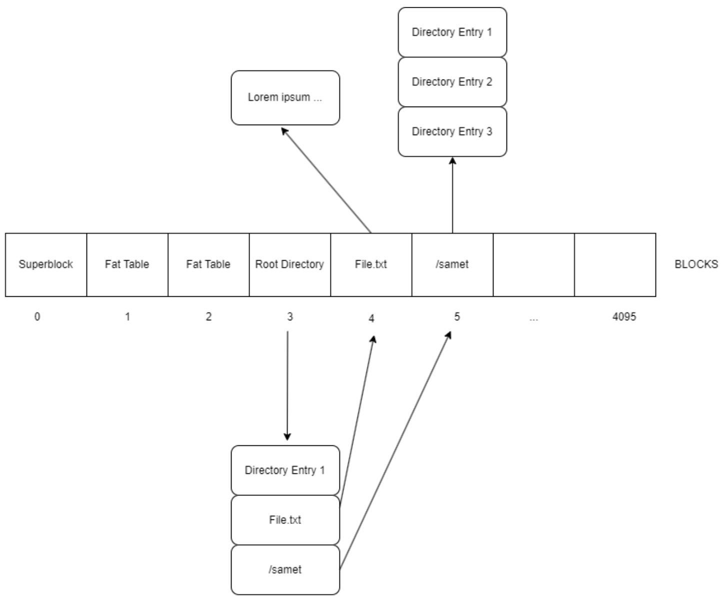
SYSTEM_BLOCK represents the required blocks that the system use, such as superblock, fat table blocks and root directory block.



Block 0	0x7FFF	SYSTEM BLOCK
Block 1	0x7FFF	SYSTEM BLOCK
....	0x00	FREE BLOCK
....	0x7FFE	END OF CHAIN
Block 4095	0x00	FREE BLOCK

In system, for each directory and file there is assigned at least one block that holds contents of the files and directories. Files contains file content and directory block contains list of directory entries that belong to the folder. Directory entries contains these attributes:

- File name
- File extension
- First block number
- File size
- Last modification time
- Directory or file information



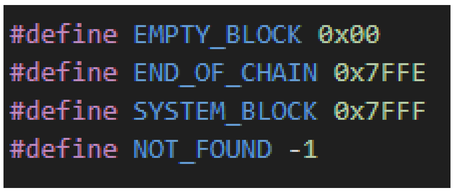
Project File Structure

These are the files that I used to build project.

- Constants.h
- DateTime.h
- DirectoryEntry.h
- DirectoryTable.h
- FileAllocationTable.h
- FileAllocationTableEntry.h
- FileManager.h
- FileSystem.h

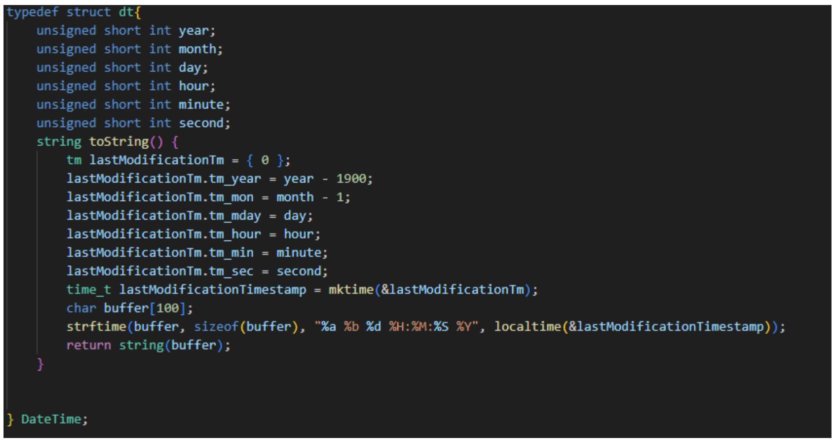
- SuperBlock.h

Constants.h: These are constants that used many places in the program. Contains macros.



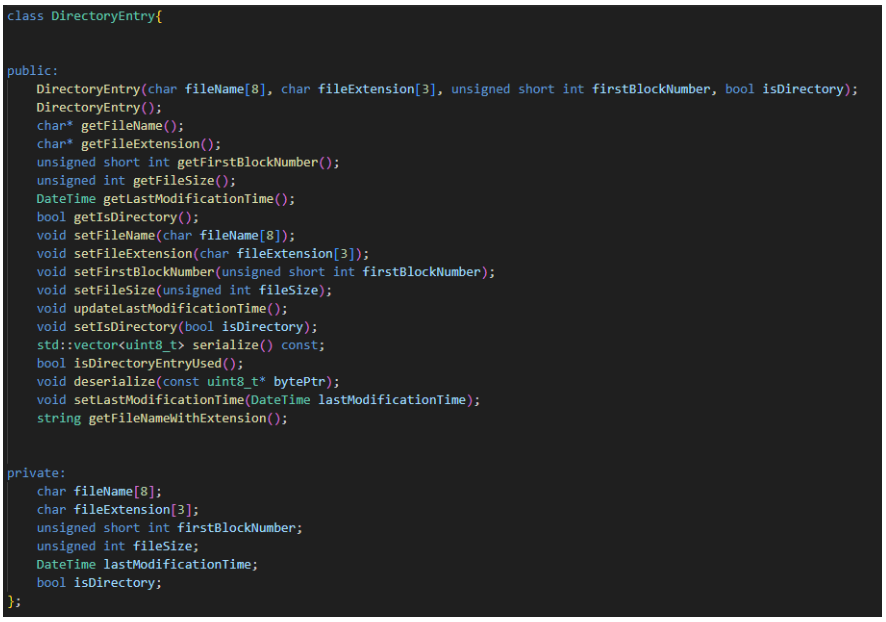
```
#define EMPTY_BLOCK 0x00
#define END_OF_CHAIN 0x7FFE
#define SYSTEM_BLOCK 0x7FFF
#define NOT_FOUND -1
```

DateTime.h: It contains one DateTime struct and function to get current date time. It is used instead of tm structure of C++ because, the original tm structure requires lots of byte and to reduce this amount I create a simple date time structure to hold last modification time for directory entries.



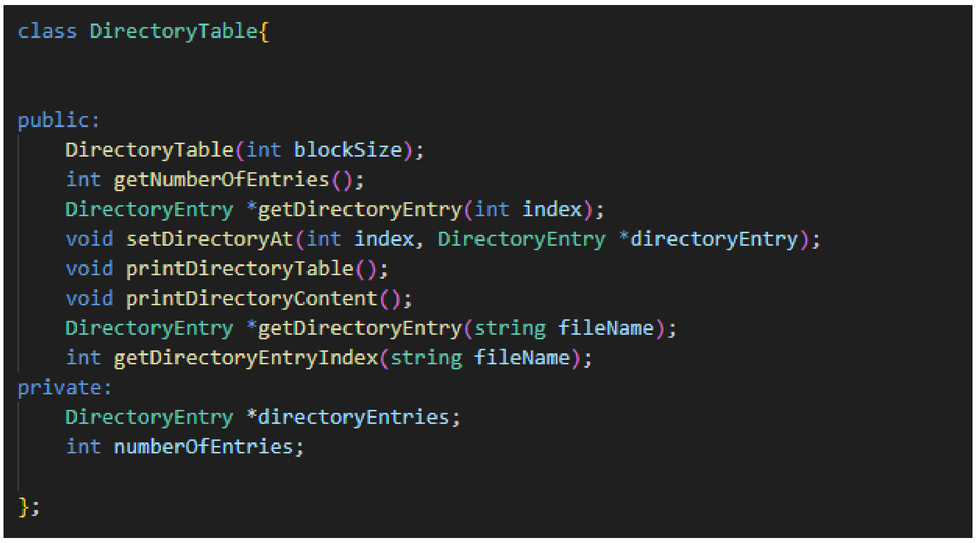
```
typedef struct dt{
    unsigned short int year;
    unsigned short int month;
    unsigned short int day;
    unsigned short int hour;
    unsigned short int minute;
    unsigned short int second;
    string toString() {
        tm lastModificationTm = { 0 };
        lastModificationTm.tm_year = year - 1900;
        lastModificationTm.tm_mon = month - 1;
        lastModificationTm.tm_mday = day;
        lastModificationTm.tm_hour = hour;
        lastModificationTm.tm_min = minute;
        lastModificationTm.tm_sec = second;
        time_t lastModificationTimestamp = mktime(&lastModificationTm);
        char buffer[100];
        strftime(buffer, sizeof(buffer), "%a %b %d %H:%M:%S %Y", localtime(&lastModificationTimestamp));
        return string(buffer);
    }


} DateTime;
```

DirectoryEntry.h: It contains attributes of directory entry attributes and number of functions to build file system. To reduce the size of Directory Entry, unsigned short int and short int is prefered according to their required ranges.



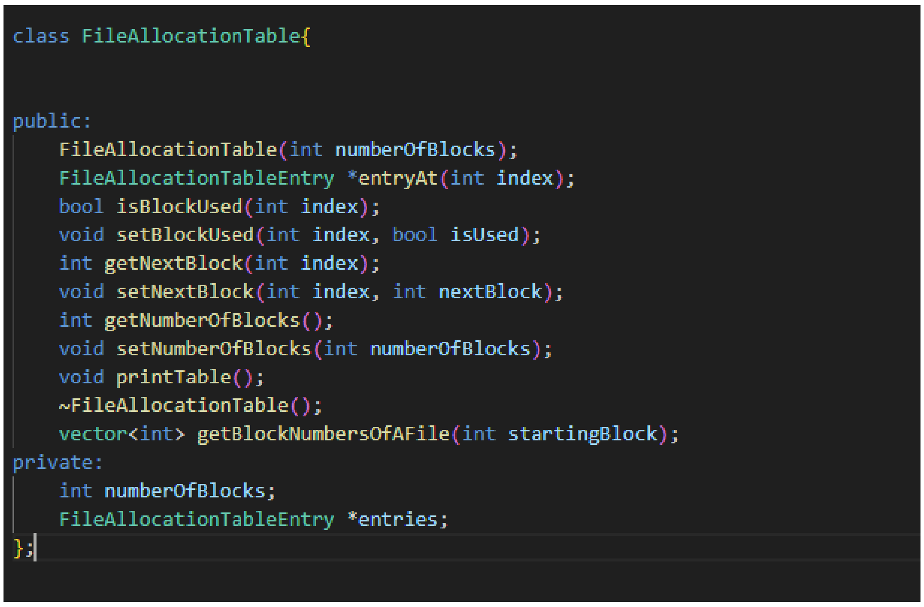
```
class DirectoryEntry{


public:
    DirectoryEntry(char fileName[8], char fileExtension[3], unsigned short int firstBlockNumber, bool isDirectory);
    DirectoryEntry();
    char* getFileName();
    char* getFileExtension();
    unsigned short int getFirstBlockNumber();
    unsigned int getFileSize();
    DateTime getLastModificationTime();
    bool getIsDirectory();
    void setFileName(char fileName[8]);
    void setFileExtension(char fileExtension[3]);
    void setFirstBlockNumber(unsigned short int firstBlockNumber);
    void setFileSize(unsigned int fileSize);
    void updateLastModificationTime();
    void setIsDirectory(bool isDirectory);
    std::vector<uint8_t> serialize() const;
    bool isDirectoryEntryUsed();
    void deserialize(const uint8_t* bytePtr);
    void setLastModificationTime(DateTime lastModificationTime);
    string getFileNameWithExtension();


private:
    char fileName[8];
    char fileExtension[3];
    unsigned short int firstBlockNumber;
    unsigned int fileSize;
    DateTime lastModificationTime;
    bool isDirectory;
};
```

DirectoryTable.h: It contains list of directory table entries to represent directory table structure and functions that used to build file system.



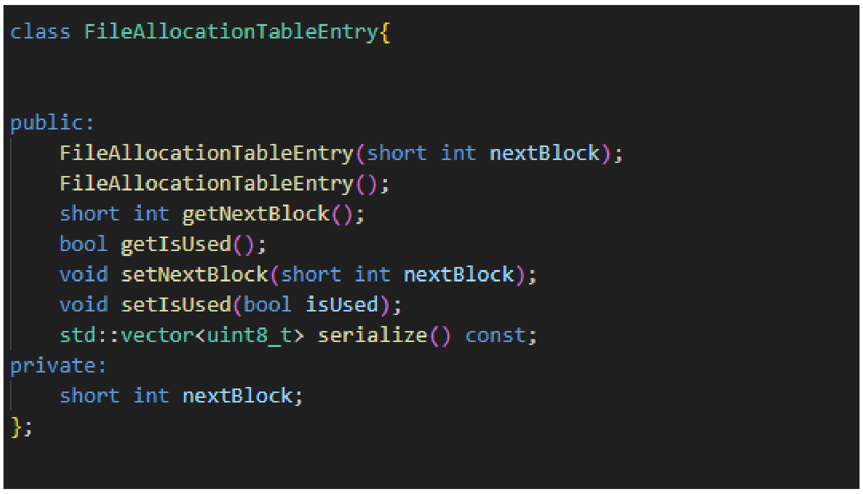
```
class DirectoryTable{


public:
    DirectoryTable(int blockSize);
    int getNumberOfEntries();
    DirectoryEntry *getDirectoryEntry(int index);
    void setDirectoryAt(int index, DirectoryEntry *directoryEntry);
    void printDirectoryTable();
    void printDirectoryContent();
    DirectoryEntry *getDirectoryEntry(string fileName);
    int getDirectoryEntryIndex(string fileName);
private:
    DirectoryEntry *directoryEntries;
    int numberOfEntries;

};
```

FileAllocationTable.h: Similar to directory table, it contains file allocation table entries and required functions to build file system.



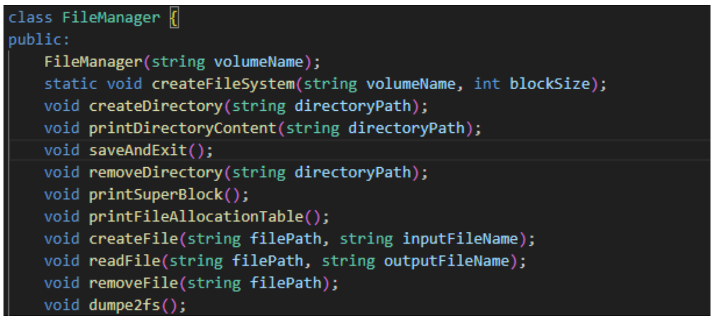
```
class FileAllocationTable{


public:
    FileAllocationTable(int numberOfBlocks);
    FileAllocationTableEntry *entryAt(int index);
    bool isBlockUsed(int index);
    void setBlockUsed(int index, bool isUsed);
    int getNextBlock(int index);
    void setNextBlock(int index, int nextBlock);
    int getNumberOfBlocks();
    void setNumberOfBlocks(int numberOfBlocks);
    void printTable();
    ~FileAllocationTable();
    vector<int> getBlockNumbersOfAFile(int startingBlock);
private:
    int numberOfBlocks;
    FileAllocationTableEntry *entries;
};
```

FileAllocationTableEntry.h: It contains the attributes that required to represent file allocation table structure. It only contains next block number. To increase amount of file allocation table entry for per block, I prefered to use short int by caring the range that I need for the system.

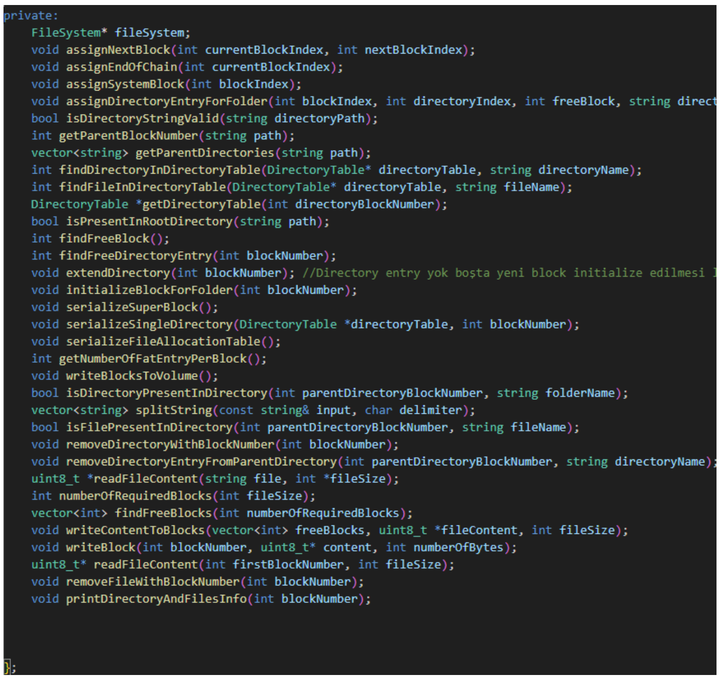


```cpp
class FileAllocationTableEntry{


public:
    FileAllocationTableEntry(short int nextBlock);
    FileAllocationTableEntry();
    short int getNextBlock();
    bool getIsUsed();
    void setNextBlock(short int nextBlock);
    void setIsUsed(bool isUsed);
    std::vector<uint8_t> serialize() const;
private:
    short int nextBlock;
};
```

FileManager.h: File manager is the most important part of the design. It is responsible for managing file system and meeting user reqests. It is designed as friend class of FileSystem class. It has methods for creating file system, reading file system, creating file, creating directory, removing directory, removing file etc. Also it has many helper functions to meet requirements.

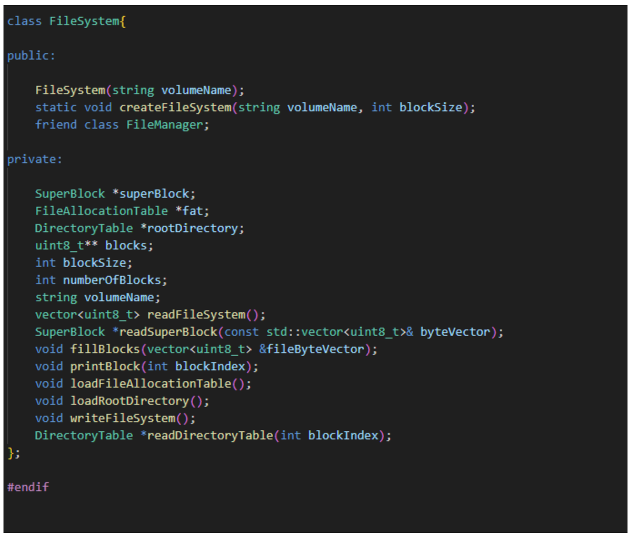


```cpp
class FileManager {
public:
    FileManager(string volumeName);
    static void createFileSystem(string volumeName, int blockSize);
    void createDirectory(string directoryPath);
    void printDirectoryContent(string directoryPath);
    void saveAndExit();
    void removeDirectory(string directoryPath);
    void printSuperBlock();
    void printFileAllocationTable();
    void createFile(string filePath, string inputFileName);
    void readFile(string filePath, string outputFileName);
    void removeFile(string filePath);
    void dumpe2fs();
```



```cpp
private:
    FileSystem* fileSystem;
    void assignNextBlock(int currentBlockIndex, int nextBlockIndex);
    void assignEndOfChain(int currentBlockIndex);
    void assignSystemBlock(int blockIndex);
    void assignDirectoryEntryForFolder(int blockIndex, int directoryIndex, int freeBlock, string direct
    bool isDirectoryStringValid(string directoryPath);
    int getParentBlockNumber(string path);
    vector<string> getParentDirectories(string path);
    int findDirectoryInDirectoryTable(DirectoryTable* directoryTable, string directoryName);
    int findFileInDirectoryTable(DirectoryTable* directoryTable, string fileName);
    DirectoryTable *getDirectoryTable(int directoryBlockNumber);
    bool isPresentInRootDirectory(string path);
    int findFreeBlock();
    int findFreeDirectoryEntry(int blockNumber);
    void extendDirectory(int blockNumber); //Directory entry yok boşta yeni block initialize edilmesi l
    void initializeBlockForFolder(int blockNumber);
    void serializeSuperBlock();
    void serializeSingleDirectory(DirectoryTable *directoryTable, int blockNumber);
    void serializeFileAllocationTable();
    int getNumberOfFatEntryPerBlock();
    void writeBlocksToVolume();
    bool isDirectoryPresentInDirectory(int parentDirectoryBlockNumber, string folderName);
    vector<string> splitString(const string& input, char delimiter);
    bool isFilePresentInDirectory(int parentDirectoryBlockNumber, string fileName);
    void removeDirectoryWithBlockNumber(int blockNumber);
    void removeDirectoryEntryFromParentDirectory(int parentDirectoryBlockNumber, string directoryName);
    uint8_t *readFileContent(string file, int *fileSize);
    int numberOfRequiredBlocks(int fileSize);
    vector<int> findFreeBlocks(int numberOfRequiredBlocks);
    void writeContentToBlocks(vector<int> freeBlocks, uint8_t *fileContent, int fileSize);
    void writeBlock(int blockNumber, uint8_t* content, int numberOfBytes);
    uint8_t* readFileContent(int firstBlockNumber, int fileSize);
    void removeFileWithBlockNumber(int blockNumber);
    void printDirectoryAndFilesInfo(int blockNumber);



};
```

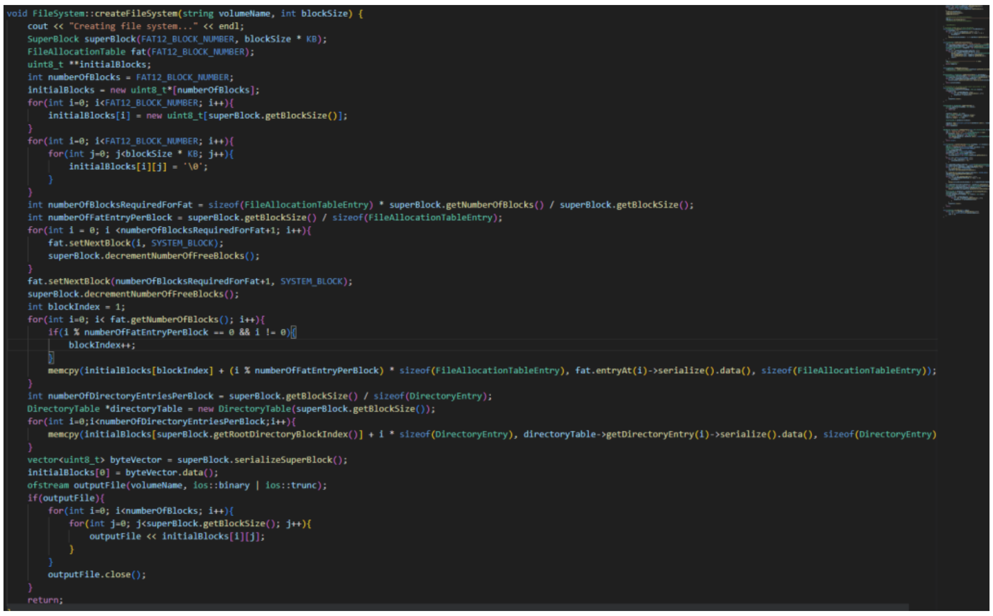
FileSystem.h: File system contains the all blocks, file allocation table, superblocki root directory in itself when the file system is created or readed. It is like a container class. Also it has some helper functions to realize some operations.



```
class FileSystem{

public:

    FileSystem(string volumeName);
    static void createFileSystem(string volumeName, int blockSize);
    friend class FileManager;

private:

    SuperBlock *superBlock;
    FileAllocationTable *fat;
    DirectoryTable *rootDirectory;
    uint8_t** blocks;
    int blockSize;
    int numberOfBlocks;
    string volumeName;
    vector<uint8_t> readFileSystem();
    SuperBlock *readSuperBlock(const std::vector<uint8_t>& byteVector);
    void fillBlocks(vector<uint8_t> &fileByteVector);
    void printBlock(int blockIndex);
    void loadFileAllocationTable();
    void loadRootDirectory();
    void writeFileSystem();
    DirectoryTable *readDirectoryTable(int blockIndex);
};

#endif
```

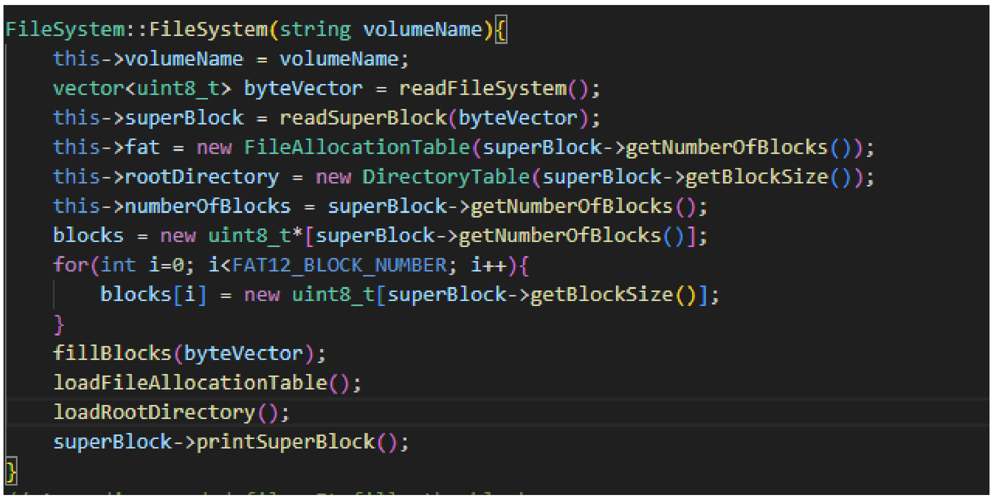
How operations are implemented ?

Make File System: To create file system, file name and block size is taken from user. According to block size, block are created in memory. After than file allocation table and superblocks are created with the calculated values. Required block count is reserved for file allocation table and their next block numbers are assigned. Then, file allocation tables content copied to necessary blocks byte by byte. After than, superblock is copied to first block with its assigned values. Than the block for root directory is assigned and block filled with empty directory entries. After file system is created in memory, it is written to the input file empty places of blocks are filled with '\0' charachter.



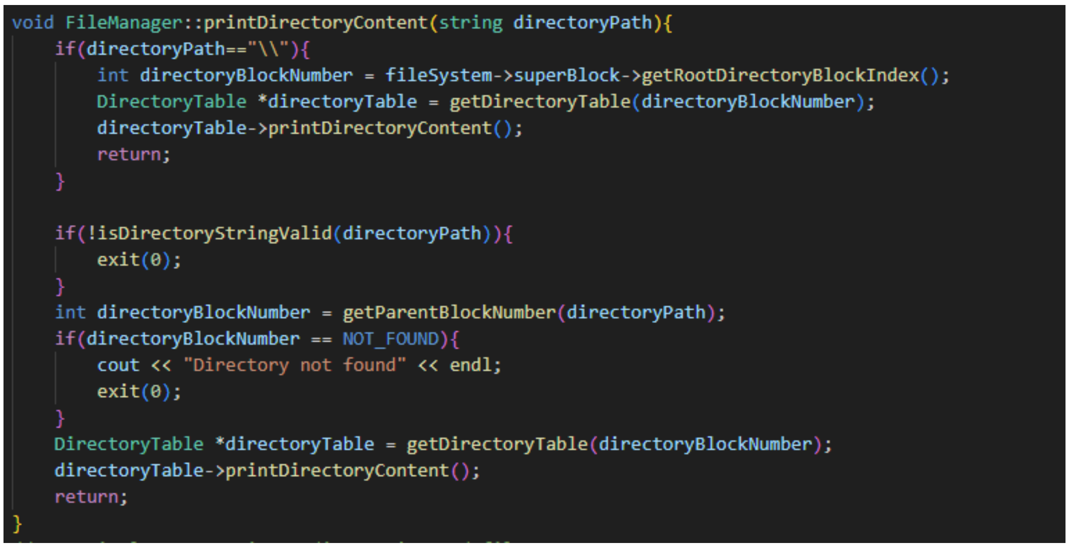
```
void FileSystem::createFileSystem(string volumeName, int blockSize) {
    cout << "Creating file system..." << endl;
    SuperBlock superBlock(FAT12_BLOCK_NUMBER, blockSize * KB);
    FileAllocationTable fat(FAT12_BLOCK_NUMBER);
    uint8_t **initialBlocks;
    int numberOfBlocks = FAT12_BLOCK_NUMBER;
    initialBlocks = new uint8_t*[numberOfBlocks];
    for(int i=0; i<FAT12_BLOCK_NUMBER; i++){
        initialBlocks[i] = new uint8_t[superBlock.getBlockSize()];
    }
    for(int i=0; i<FAT12_BLOCK_NUMBER; i++){
        for(int j=0; j<blockSize * KB; j++){
            initialBlocks[i][j] = '\0';
        }
    }
    int numberOfBlocksRequiredForFat = sizeof(FileAllocationTableEntry) * superBlock.getNumberOfBlocks() / superBlock.getBlockSize();
    int numberOfFatEntryPerBlock = superBlock.getBlockSize() / sizeof(FileAllocationTableEntry);
    for(int i = 0; i <numberOfBlocksRequiredForFat+1; i++){
        fat.setNextBlock(i, SYSTEM_BLOCK);
        superBlock.decrementNumberOfFreeBlocks();
    }
    fat.setNextBlock(numberOfBlocksRequiredForFat+1, SYSTEM_BLOCK);
    superBlock.decrementNumberOfFreeBlocks();
    int blockIndex = 1;
    for(int i=0; i< fat.getNumberOfBlocks(); i++){
        if(i % numberOfFatEntryPerBlock == 0 && i != 0){
            blockIndex++;
        }
        memcpy(initialBlocks[blockIndex] + (i % numberOfFatEntryPerBlock) * sizeof(FileAllocationTableEntry), fat.entryAt(i)->serialize().data(), sizeof(FileAllocationTableEntry));
    }
    int numberOfDirectoryEntriesPerBlock = superBlock.getBlockSize() / sizeof(DirectoryEntry);
    DirectoryTable *directoryTable = new DirectoryTable(superBlock.getBlockSize());
    for(int i=0;i<numberOfDirectoryEntriesPerBlock;i++){
        memcpy(initialBlocks[superBlock.getRootDirectoryBlockIndex()] + i * sizeof(DirectoryEntry), directoryTable->getDirectoryEntry(i)->serialize().data(), sizeof(DirectoryEntry)
    }
    vector<uint8_t> byteVector = superBlock.serializeSuperBlock();
    initialBlocks[0] = byteVector.data();
    ofstream outputFile(volumeName, ios::binary | ios::trunc);
    if(outputFile){
        for(int i=0; i<numberOfBlocks; i++){
            for(int j=0; j<superBlock.getBlockSize(); j++){
                outputFile << initialBlocks[i][j];
            }
        }
        outputFile.close();
    }
    return;
```

Read File System: To read file system, constructor of file system and helper functions are used. File is readed into byte array and than it is divided into block by block. Superblock is readed from first block. According to superblock informations, whole system is readed.



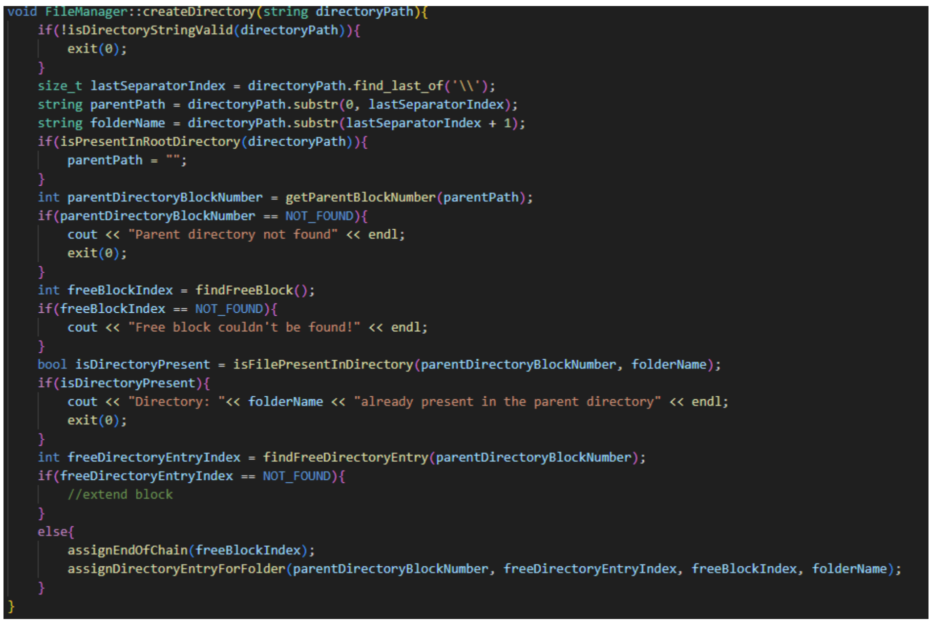
```
FileSystem::FileSystem(string volumeName){
    this->volumeName = volumeName;
    vector<uint8_t> byteVector = readFileSystem();
    this->superBlock = readSuperBlock(byteVector);
    this->fat = new FileAllocationTable(superBlock->getNumberOfBlocks());
    this->rootDirectory = new DirectoryTable(superBlock->getBlockSize());
    this->numberOfBlocks = superBlock->getNumberOfBlocks();
    blocks = new uint8_t*[superBlock->getNumberOfBlocks()];
    for(int i=0; i<FAT12_BLOCK_NUMBER; i++){
        blocks[i] = new uint8_t[superBlock->getBlockSize()];
    }
    fillBlocks(byteVector);
    loadFileAllocationTable();
    loadRootDirectory();
    superBlock->printSuperBlock();
}
```

Dir: For dir operation, printDirectoryContent function is used. First it checks whether the directory path is root directory or not. If it is, it gets and prints the directory table. If it is not, it checks whether the directory string valid or not. After that, It gets the block number of directory then it gets the directory table. Finally it prints the contents.



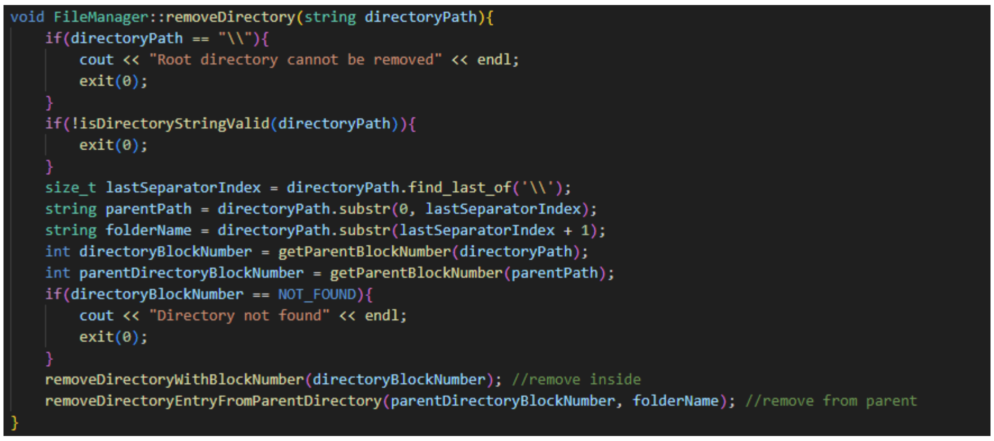
```
void FileManager::printDirectoryContent(string directoryPath){
    if(directoryPath=="\\"){
        int directoryBlockNumber = fileSystem->superBlock->getRootDirectoryBlockIndex();
        DirectoryTable *directoryTable = getDirectoryTable(directoryBlockNumber);
        directoryTable->printDirectoryContent();
        return;
    }

    if(!isDirectoryStringValid(directoryPath)){
        exit(0);
    }
    int directoryBlockNumber = getParentBlockNumber(directoryPath);
    if(directoryBlockNumber == NOT_FOUND){
        cout << "Directory not found" << endl;
        exit(0);
    }
    DirectoryTable *directoryTable = getDirectoryTable(directoryBlockNumber);
    directoryTable->printDirectoryContent();
    return;
}
```

Mkdir: For mkdir operation, it controls the directory string. After that, it gets the parent directory and folder name by splitting the given path. It checks whether parent directory is exist or not. If it is exist, it gets directory table and check whether there is a file with the same name or not. If it is not, it finds empty block to assign for new directory. It creates directory entry for new folder in parent directory. Updates file allocation table and superblock.



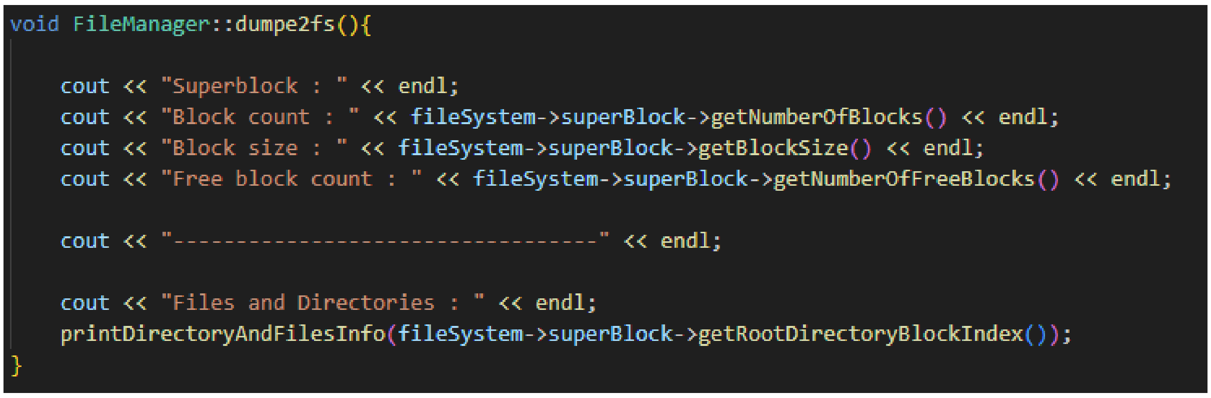
```cpp
void FileManager::createDirectory(string directoryPath){
    if(!isDirectoryStringValid(directoryPath)){
        exit(0);
    }
    size_t lastSeparatorIndex = directoryPath.find_last_of('\\');
    string parentPath = directoryPath.substr(0, lastSeparatorIndex);
    string folderName = directoryPath.substr(lastSeparatorIndex + 1);
    if(isPresentInRootDirectory(directoryPath)){
        parentPath = "";
    }
    int parentDirectoryBlockNumber = getParentBlockNumber(parentPath);
    if(parentDirectoryBlockNumber == NOT_FOUND){
        cout << "Parent directory not found" << endl;
        exit(0);
    }
    int freeBlockIndex = findFreeBlock();
    if(freeBlockIndex == NOT_FOUND){
        cout << "Free block couldn't be found!" << endl;
    }
    bool isDirectoryPresent = isFilePresentInDirectory(parentDirectoryBlockNumber, folderName);
    if(isDirectoryPresent){
        cout << "Directory: "<< folderName << "already present in the parent directory" << endl;
        exit(0);
    }
    int freeDirectoryEntryIndex = findFreeDirectoryEntry(parentDirectoryBlockNumber);
    if(freeDirectoryEntryIndex == NOT_FOUND){
        //extend block
    }
    else{
        assignEndOfChain(freeBlockIndex);
        assignDirectoryEntryForFolder(parentDirectoryBlockNumber, freeDirectoryEntryIndex, freeBlockIndex, folderName);
    }
}
```

Rmdir: For rmdir operation, directory path is checked whether it is valid or not. After that parent path is tried to find. If it is found, directory entry from directory table and files that belongs to this directories are removed from the system recursively.



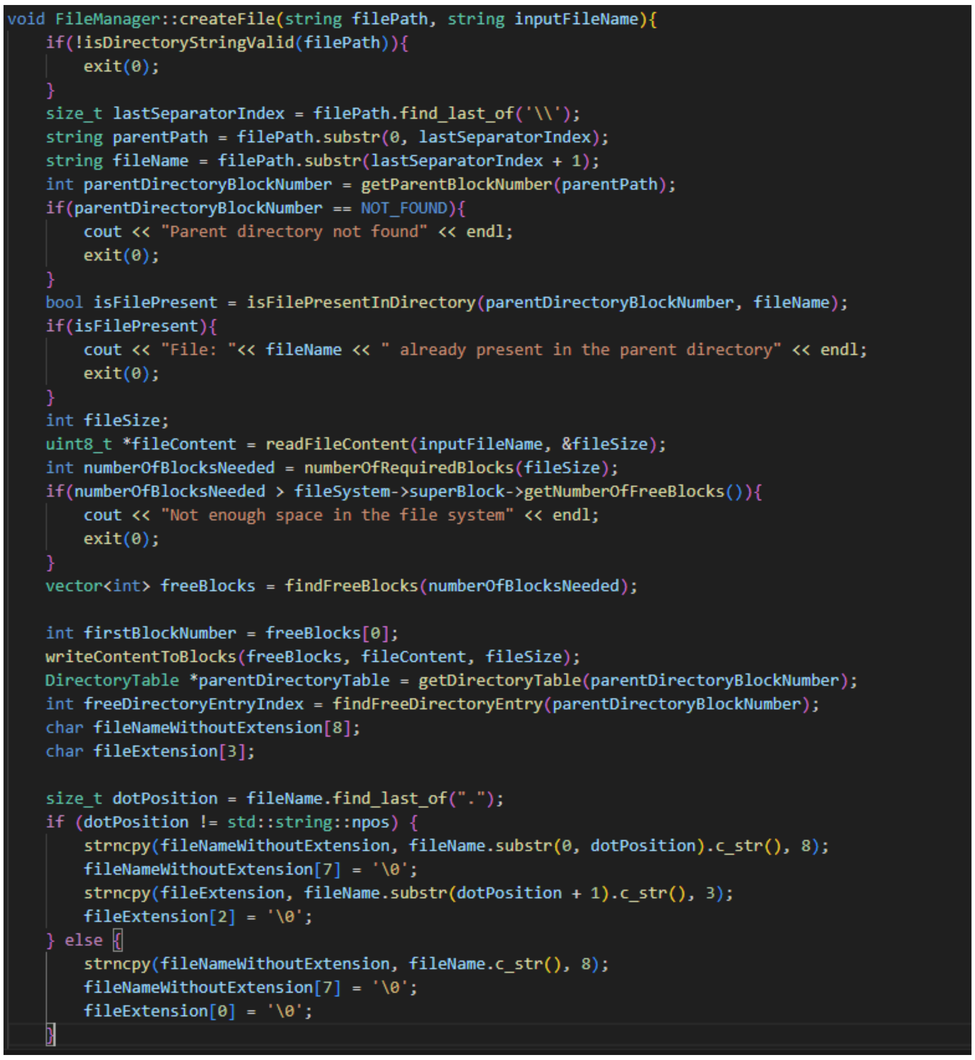
```cpp
void FileManager::removeDirectory(string directoryPath){
    if(directoryPath == "\\"){
        cout << "Root directory cannot be removed" << endl;
        exit(0);
    }
    if(!isDirectoryStringValid(directoryPath)){
        exit(0);
    }
    size_t lastSeparatorIndex = directoryPath.find_last_of('\\');
    string parentPath = directoryPath.substr(0, lastSeparatorIndex);
    string folderName = directoryPath.substr(lastSeparatorIndex + 1);
    int directoryBlockNumber = getParentBlockNumber(directoryPath);
    int parentDirectoryBlockNumber = getParentBlockNumber(parentPath);
    if(directoryBlockNumber == NOT_FOUND){
        cout << "Directory not found" << endl;
        exit(0);
    }
    removeDirectoryWithBlockNumber(directoryBlockNumber); //remove inside
    removeDirectoryEntryFromParentDirectory(parentDirectoryBlockNumber, folderName); //remove from parent
}
```

dumpe2fs: This function prints informations about the system. Also prints the files with occupied block numbers.

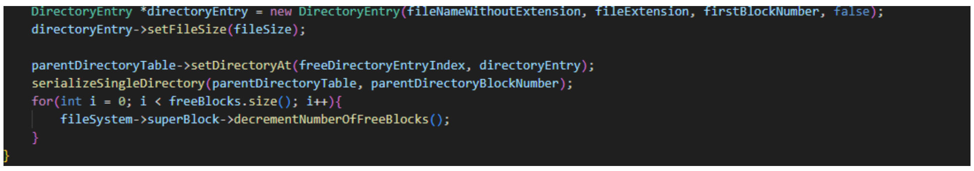


```cpp
void FileManager::dumpe2fs(){

    cout << "Superblock : " << endl;
    cout << "Block count : " << fileSystem->superBlock->getNumberOfBlocks() << endl;
    cout << "Block size : " << fileSystem->superBlock->getBlockSize() << endl;
    cout << "Free block count : " << fileSystem->superBlock->getNumberOfFreeBlocks() << endl;

    cout << "-----------------------------------" << endl;

    cout << "Files and Directories : " << endl;
    printDirectoryAndFilesInfo(fileSystem->superBlock->getRootDirectoryBlockIndex());
}
```

Write: This function first check whether file path for the new file is valid or not. Than it tries to find its parent directory. After that, it checks whether the file present in directory or not. Than, it reads the input file from the linux system. Allocates free blocks to keep file in file system. Than, it writes file to these blocks. Puts directory entry to parents directory table and updates file allocation table and superblocks informations.

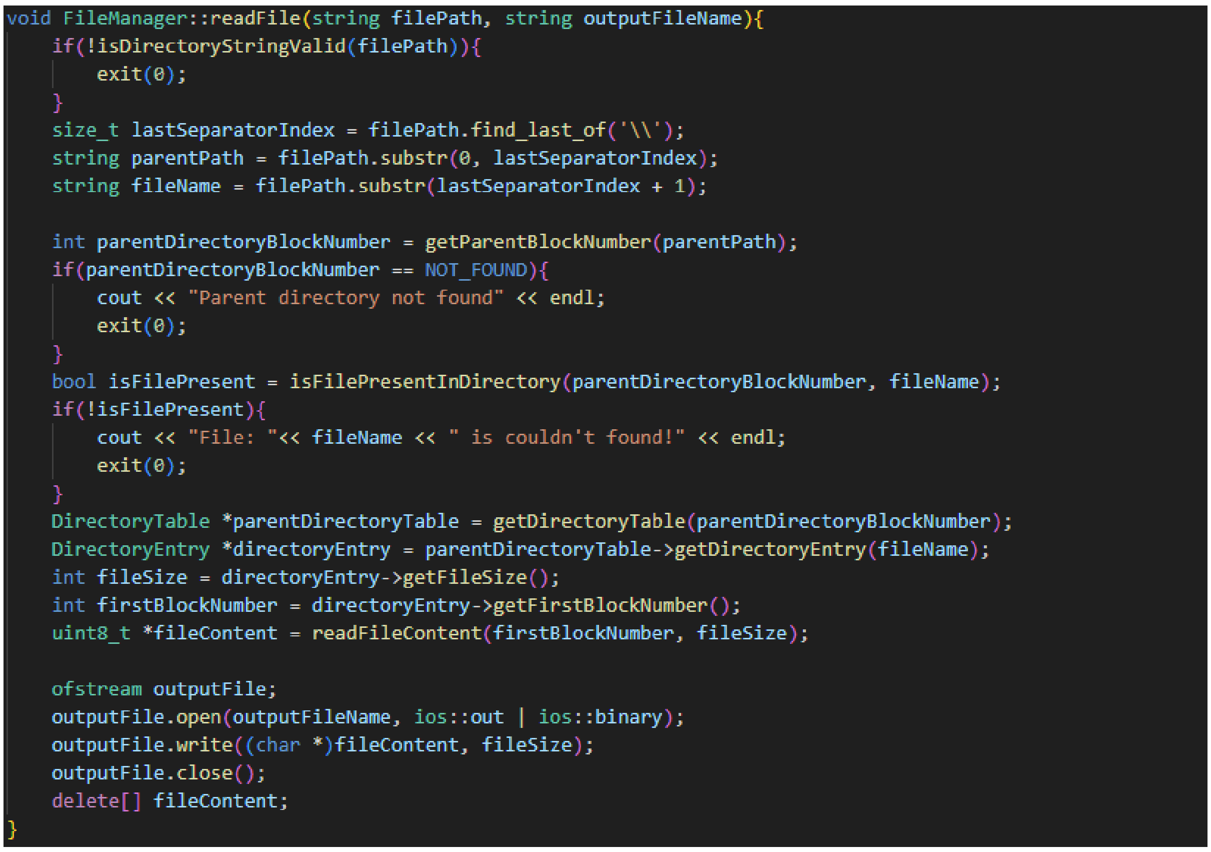


```cpp
void FileManager::createFile(string filePath, string inputFileName){
    if(!isDirectoryStringValid(filePath)){
        exit(0);
    }
    size_t lastSeparatorIndex = filePath.find_last_of('\\');
    string parentPath = filePath.substr(0, lastSeparatorIndex);
    string fileName = filePath.substr(lastSeparatorIndex + 1);
    int parentDirectoryBlockNumber = getParentBlockNumber(parentPath);
    if(parentDirectoryBlockNumber == NOT_FOUND){
        cout << "Parent directory not found" << endl;
        exit(0);
    }
    bool isFilePresent = isFilePresentInDirectory(parentDirectoryBlockNumber, fileName);
    if(isFilePresent){
        cout << "File: "<< fileName << " already present in the parent directory" << endl;
        exit(0);
    }
    int fileSize;
    uint8_t *fileContent = readFileContent(inputFileName, &fileSize);
    int numberOfBlocksNeeded = numberOfRequiredBlocks(fileSize);
    if(numberOfBlocksNeeded > fileSystem->superBlock->getNumberOfFreeBlocks()){
        cout << "Not enough space in the file system" << endl;
        exit(0);
    }
    vector<int> freeBlocks = findFreeBlocks(numberOfBlocksNeeded);

    int firstBlockNumber = freeBlocks[0];
    writeContentToBlocks(freeBlocks, fileContent, fileSize);
    DirectoryTable *parentDirectoryTable = getDirectoryTable(parentDirectoryBlockNumber);
    int freeDirectoryEntryIndex = findFreeDirectoryEntry(parentDirectoryBlockNumber);
    char fileNameWithoutExtension[8];
    char fileExtension[3];

    size_t dotPosition = fileName.find_last_of(".");
    if (dotPosition != std::string::npos) {
        strncpy(fileNameWithoutExtension, fileName.substr(0, dotPosition).c_str(), 8);
        fileNameWithoutExtension[7] = '\0';
        strncpy(fileExtension, fileName.substr(dotPosition + 1).c_str(), 3);
        fileExtension[2] = '\0';
    } else {
        strncpy(fileNameWithoutExtension, fileName.c_str(), 8);
        fileNameWithoutExtension[7] = '\0';
        fileExtension[0] = '\0';
    }
```



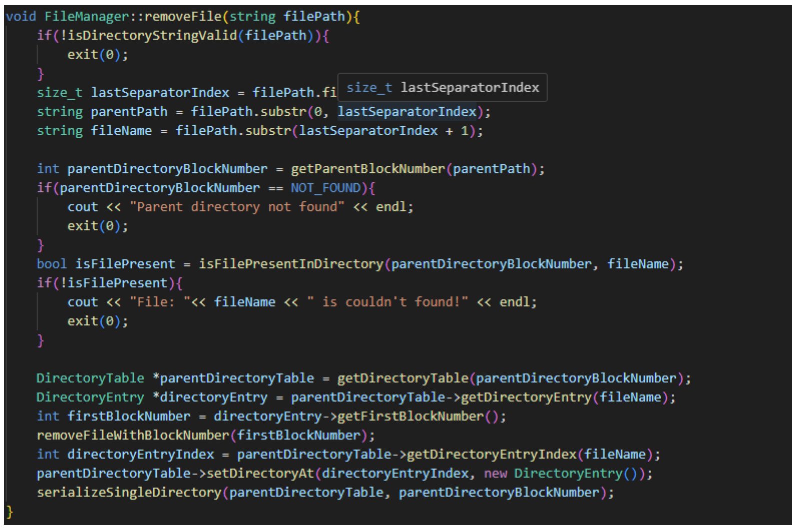
```cpp
    DirectoryEntry *directoryEntry = new DirectoryEntry(fileNameWithoutExtension, fileExtension, firstBlockNumber, false);
    directoryEntry->setFileSize(fileSize);

    parentDirectoryTable->setDirectoryAt(freeDirectoryEntryIndex, directoryEntry);
    serializeSingleDirectory(parentDirectoryTable, parentDirectoryBlockNumber);
    for(int i = 0; i < freeBlocks.size(); i++){
        fileSystem->superBlock->decrementNumberOfFreeBlocks();
    }
}
```

Read: Function first check is directory string valid or not. Than tries to find parent directory. If it is found it checks whether file is present in directory or not. If it is present, it gets first block number and reads the chain. Than it writes content to output file.



```cpp
void FileManager::readFile(string filePath, string outputFileName){
    if(!isDirectoryStringValid(filePath)){
        exit(0);
    }
    size_t lastSeparatorIndex = filePath.find_last_of('\\');
    string parentPath = filePath.substr(0, lastSeparatorIndex);
    string fileName = filePath.substr(lastSeparatorIndex + 1);

    int parentDirectoryBlockNumber = getParentBlockNumber(parentPath);
    if(parentDirectoryBlockNumber == NOT_FOUND){
        cout << "Parent directory not found" << endl;
        exit(0);
    }
    bool isFilePresent = isFilePresentInDirectory(parentDirectoryBlockNumber, fileName);
    if(!isFilePresent){
        cout << "File: "<< fileName << " is couldn't found!" << endl;
        exit(0);
    }
    DirectoryTable *parentDirectoryTable = getDirectoryTable(parentDirectoryBlockNumber);
    DirectoryEntry *directoryEntry = parentDirectoryTable->getDirectoryEntry(fileName);
    int fileSize = directoryEntry->getFileSize();
    int firstBlockNumber = directoryEntry->getFirstBlockNumber();
    uint8_t *fileContent = readFileContent(firstBlockNumber, fileSize);

    ofstream outputFile;
    outputFile.open(outputFileName, ios::out | ios::binary);
    outputFile.write((char *)fileContent, fileSize);
    outputFile.close();
    delete[] fileContent;
}
```

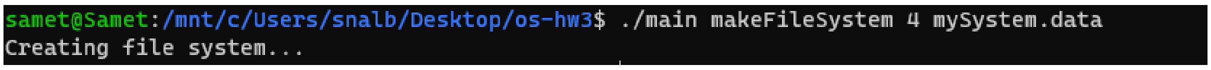
Del: It checks whether file path is valid or not. Than it tries to find parent directory. If it is found, It checks whether the file is present or not. If it is present, from starting block of chain to end of the chain, it frees the blocks, update file allocation table and superblock. After that it removes the directory entry from parent directory.



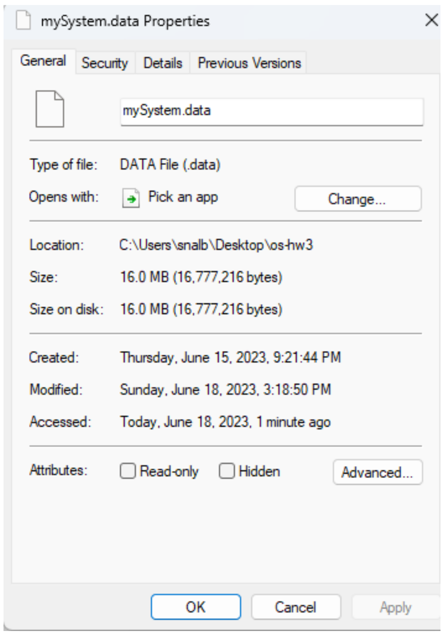
```cpp
void FileManager::removeFile(string filePath){
    if(!isDirectoryStringValid(filePath)){
        exit(0);
    }
    size_t lastSeparatorIndex = filePath.fi size_t lastSeparatorIndex
    string parentPath = filePath.substr(0, lastSeparatorIndex);
    string fileName = filePath.substr(lastSeparatorIndex + 1);

    int parentDirectoryBlockNumber = getParentBlockNumber(parentPath);
    if(parentDirectoryBlockNumber == NOT_FOUND){
        cout << "Parent directory not found" << endl;
        exit(0);
    }
    bool isFilePresent = isFilePresentInDirectory(parentDirectoryBlockNumber, fileName);
    if(!isFilePresent){
        cout << "File: "<< fileName << " is couldn't found!" << endl;
        exit(0);
    }

    DirectoryTable *parentDirectoryTable = getDirectoryTable(parentDirectoryBlockNumber);
    DirectoryEntry *directoryEntry = parentDirectoryTable->getDirectoryEntry(fileName);
    int firstBlockNumber = directoryEntry->getFirstBlockNumber();
    removeFileWithBlockNumber(firstBlockNumber);
    int directoryEntryIndex = parentDirectoryTable->getDirectoryEntryIndex(fileName);
    parentDirectoryTable->setDirectoryAt(directoryEntryIndex, new DirectoryEntry());
    serializeSingleDirectory(parentDirectoryTable, parentDirectoryBlockNumber);
}
```

TESTS:

MakeFileSystem



```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main makeFileSystem 4 mySystem.data
Creating file system...
```



mySystem.data Properties

General | Security | Details | Previous Versions

mySystem.data

Type of file: DATA File (.data)

Opens with: Pick an app

Location: C:\Users\snalb\Desktop\os-hw3

Size: 16.0 MB (16,777,216 bytes)

Size on disk: 16.0 MB (16,777,216 bytes)

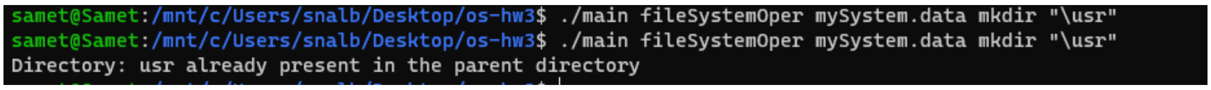
Created: Thursday, June 15, 2023, 9:21:44 PM

Modified: Sunday, June 18, 2023, 3:18:50 PM

Accessed: Today, June 18, 2023, 1 minute ago

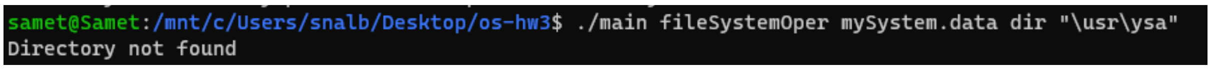
Attributes: ☐ Read-only ☐ Hidden

Creating directory

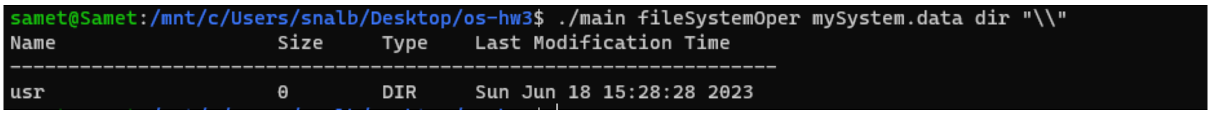


```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data mkdir "\usr"
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data mkdir "\usr"
Directory: usr already present in the parent directory
```

Dir

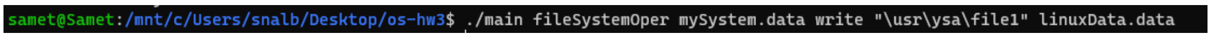


```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data dir "\usr\ysa"
Directory not found
```

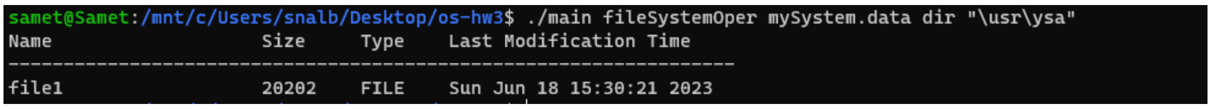


```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data dir "\\"
Name                    Size     Type    Last Modification Time
------------------------------------------------------------------
usr                     0        DIR     Sun Jun 18 15:28:28 2023
```

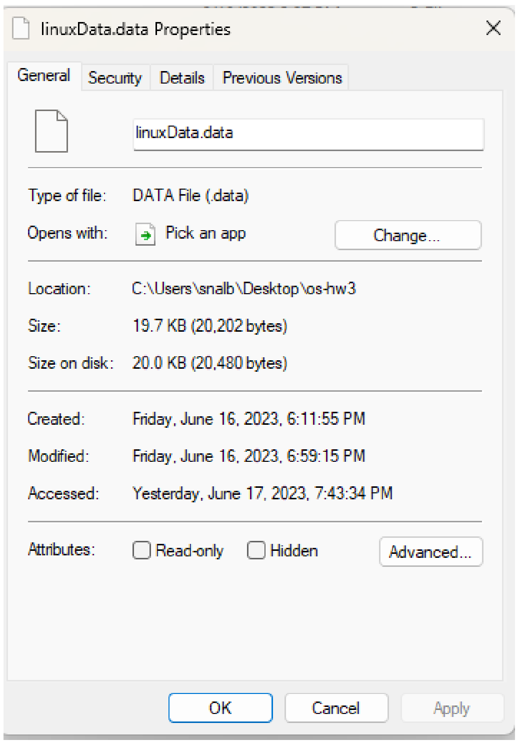
Write



```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data write "\usr\ysa\file1" linuxData.data
```



```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data dir "\usr\ysa"
Name                    Size     Type    Last Modification Time
------------------------------------------------------------------
file1                   20202    FILE    Sun Jun 18 15:30:21 2023
```



linuxData.data Properties

General | Security | Details | Previous Versions

linuxData.data

Type of file: DATA File (.data)

Opens with: Pick an app

Location: C:\Users\snalb\Desktop\os-hw3

Size: 19.7 KB (20,202 bytes)

Size on disk: 20.0 KB (20,480 bytes)

Created: Friday, June 16, 2023, 6:11:55 PM

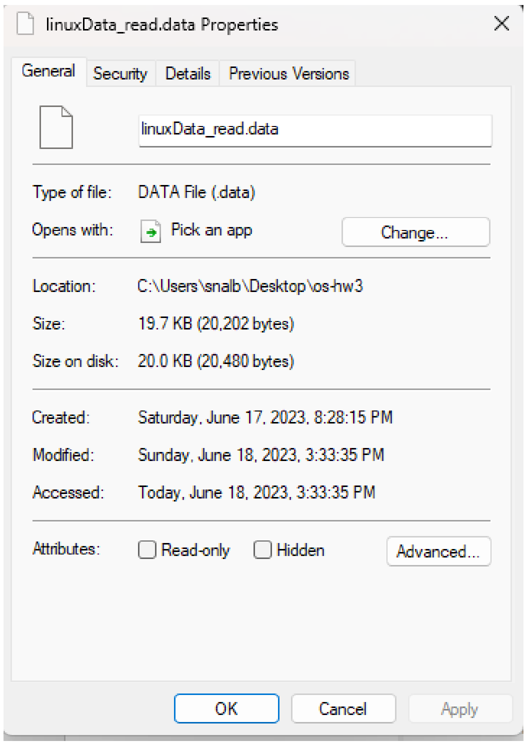
Modified: Friday, June 16, 2023, 6:59:15 PM

Accessed: Yesterday, June 17, 2023, 7:43:34 PM

Attributes: Read-only, Hidden

Read

```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data read "\usr\ysa\file1" linuxData_read.data
```



linuxData_read.data Properties

General | Security | Details | Previous Versions

linuxData_read.data

Type of file: DATA File (.data)

Opens with: Pick an app

Location: C:\Users\snalb\Desktop\os-hw3

Size: 19.7 KB (20,202 bytes)

Size on disk: 20.0 KB (20,480 bytes)

Created: Saturday, June 17, 2023, 8:28:15 PM

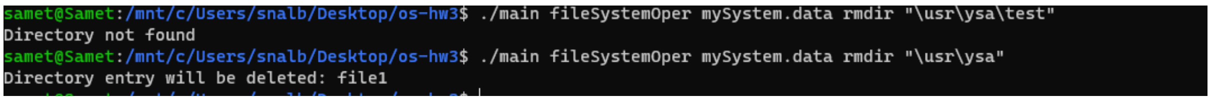
Modified: Sunday, June 18, 2023, 3:33:35 PM

Accessed: Today, June 18, 2023, 3:33:35 PM

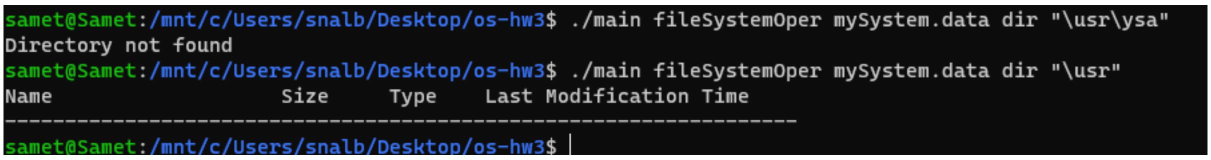
Attributes: Read-only, Hidden

```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ cmp linuxData.data linuxData_Read.data
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ 
```

Rmdir

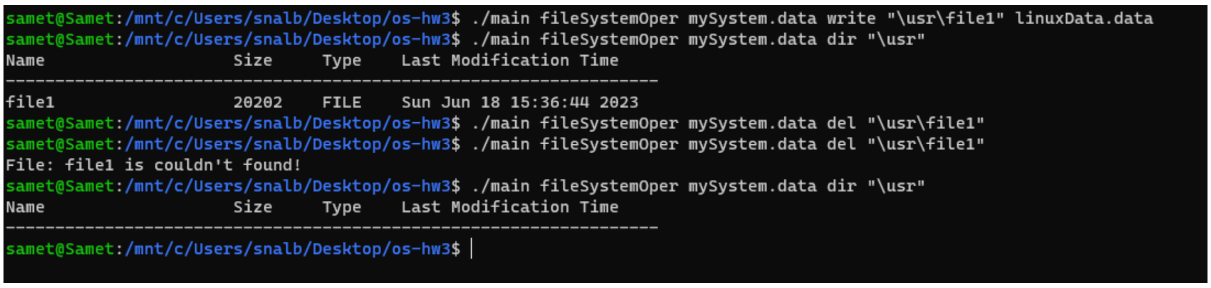


```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data rmdir "\usr\ysa\test"
Directory not found
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data rmdir "\usr\ysa"
Directory entry will be deleted: file1
```



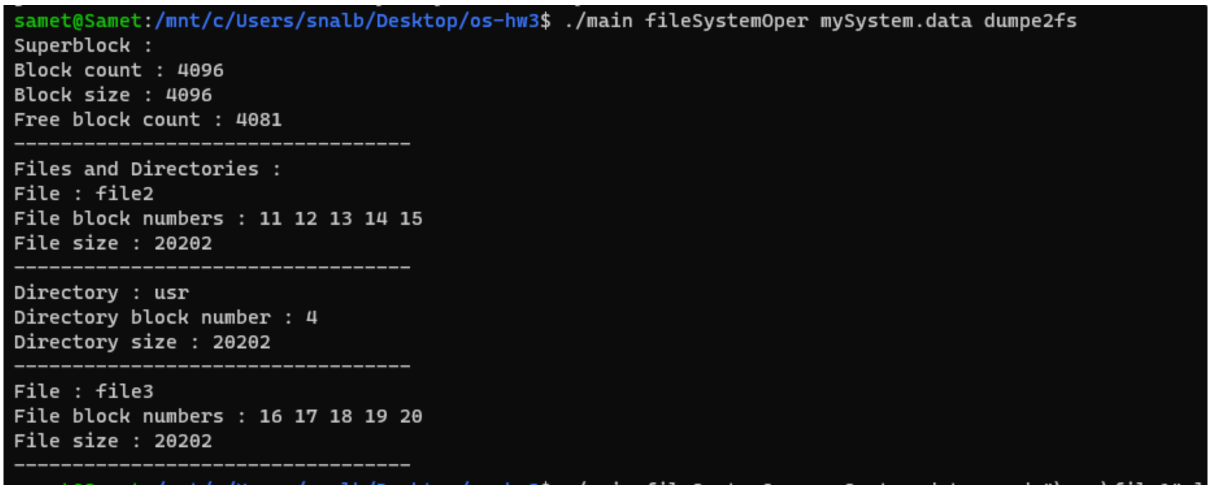
```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data dir "\usr\ysa"
Directory not found
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data dir "\usr"
Name                   Size     Type     Last Modification Time
------------------------------------------------------------------
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$
```

Del



```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data write "\usr\file1" linuxData.data
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data dir "\usr"
Name                   Size     Type     Last Modification Time
------------------------------------------------------------------
file1                  20202    FILE     Sun Jun 18 15:36:44 2023
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data del "\usr\file1"
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data del "\usr\file1"
File: file1 is couldn't found!
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data dir "\usr"
Name                   Size     Type     Last Modification Time
------------------------------------------------------------------
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$
```

dumpe2fs



```
samet@Samet:/mnt/c/Users/snalb/Desktop/os-hw3$ ./main fileSystemOper mySystem.data dumpe2fs
Superblock :
Block count : 4096
Block size : 4096
Free block count : 4081
----------------------------------
Files and Directories :
File : file2
File block numbers : 11 12 13 14 15
File size : 20202
----------------------------------
Directory : usr
Directory block number : 4
Directory size : 20202
----------------------------------
File : file3
File block numbers : 16 17 18 19 20
File size : 20202
----------------------------------
```