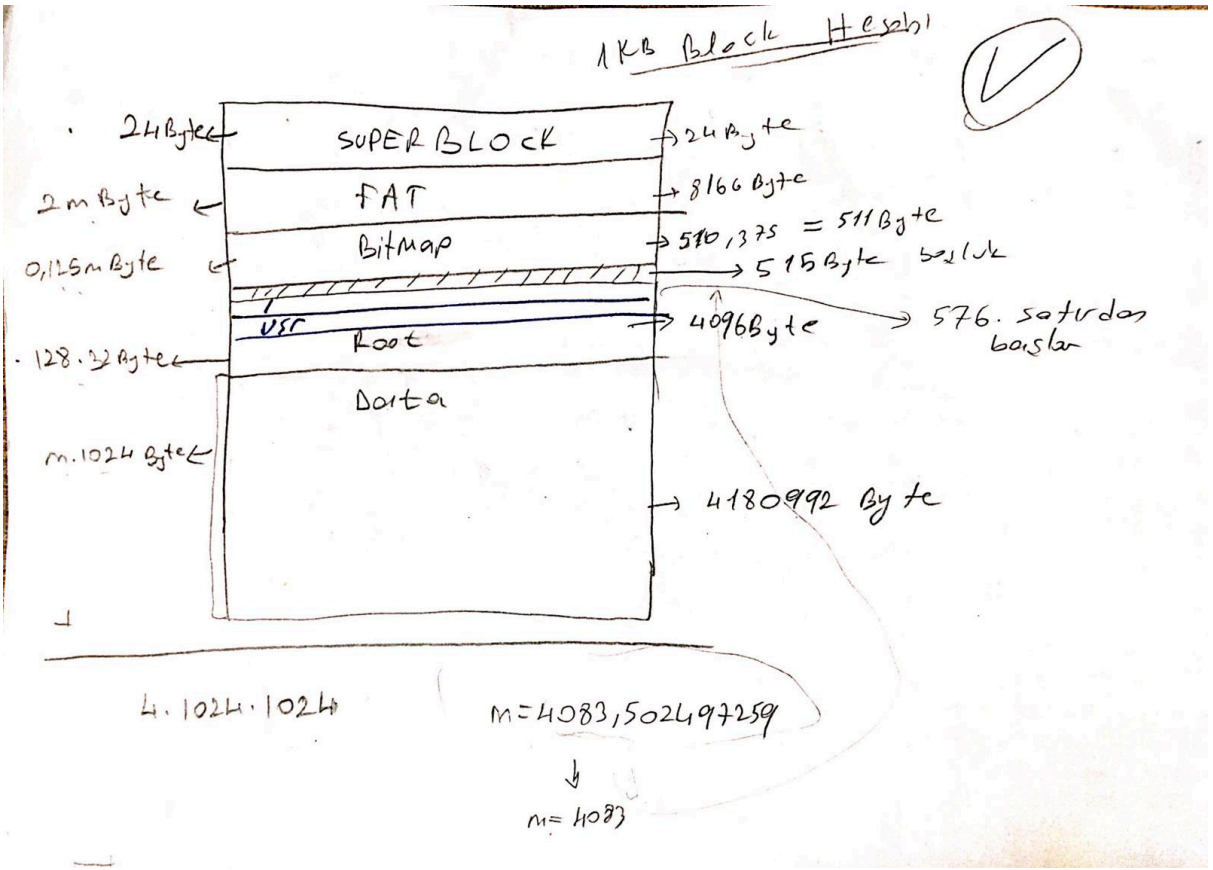
CSE312 HW-2

Author: Hacı Hasan Savan
Num: 1901042704

Part1-2:

To implement the filesystem, some calculations have to be made:

The calculation is simple actually. there is a tricky part to be talked about which is empty space between bitmap and root. this space is intentionally leaved. The aim is to achieve calculations easily in the operations.



Note: The same calculations were made for **512 kb** block sizes. you can see that in the code.

1. Directory Table and Directory Entries

Directory Table:

The directory table in our FAT12-like file system is defined within the root table structure. The root directory contains an array of FileEntry structures, each representing a file or directory within the file system.

```
// toplam 32 byte
struct FileEntry
{
    uint16_t permissions;      // 2byte first byte R read permission, second byte W write permission
    char fileName[11];         // 11byte 8+3 extension yok
    char password[9];          // 9byte
    uint8_t size;              // 1byte bu dosyanın size'ı
    uint8_t attribute;         // 1byte 0 means directory 1 means file
    uint16_t lastModifiedDate; // 2byte
    uint16_t lastModifiedTime; // 2byte
    uint16_t creationTime;     // 2byte
    uint16_t creationDate;     // 2byte
};
```

Directory Entry Structure:

The FileEntry structure is defined as follows:

- uint16_t permissions; - This field is used to store the permissions for the file or directory. The first byte indicates read permissions, and the second byte indicates write permissions.
- char fileName[11]; - This field stores the name of the file or directory, limited to 11 characters (8 characters for the name and 3 characters for the extension).
- char password[9]; - This field stores an optional password for the file or directory, limited to 9 characters.
- uint8_t size; - This field stores the size of the file in bytes.
- uint8_t attribute; - This field indicates whether the entry is a file or a directory (0 means directory, 1 means file).
- uint16_t lastModifiedDate; - This field stores the last modified date of the file or directory in FAT date format.
- uint16_t lastModifiedTime; - This field stores the last modified time of the file or directory in FAT time format.
- uint16_t creationTime; - This field stores the creation time of the file or directory in FAT time format.
- uint16_t creationDate; - This field stores the creation date of the file or directory in FAT date format.

2. Free Blocks Management

FAT (File Allocation Table):

The FAT is used to keep track of free and used blocks within the file system. Each entry in the FAT represents a block, with a value of 0x0000 indicating that the block is free and a non-zero value indicating that the other block of the file.

```
struct FATEntry
{
    uint16_t next;
};
```

Bitmap:

A bitmap is also utilized to track the status of blocks. Each bit in the bitmap represents a block, with 0 indicating an unused block and 1 indicating a used block. The bitmap helps in quickly identifying free blocks during file creation and allocation.

```
struct BitMap
{
    uint8_t status;
};
```

3. Handling Arbitrary Length of File Names

File Name Length:

The file system currently supports file names of up to 11 characters (8 characters for the name and 3 characters for the extension).

4. Handling Permissions

Permissions Management:

Permissions are handled using the permissions field in the FileEntry structure. This field consists of two bytes:

- The first byte represents read permission
- The second byte represents write permission

Checking Permissions:

Before performing any file operation (read/write), the system checks the relevant bits in the permissions field to determine if the operation is allowed.

5. Handling Password Protection

Password Storage:

Password protection is implemented using the password field in the FileEntry structure.

Password Verification:

When a user attempts to access a password-protected file or directory, the system prompts for the password. The entered password is hashed and compared with the stored hashed password. If the passwords match, access is granted; otherwise, access is denied.

This is the main class to control the system:

```
class FileSystem
{
private:
    Superblock superblock;
    std::vector<FATEntry> fat;
    std::vector<BitMap> bitMap;
    std::vector<FileEntry> root;
    std::vector<char> emptyBlock;
    std::vector<char> dataBlocks;
    bool block_1kb = true; // true 1kb false 512byte
```

Super Block:

```
struct Superblock
{
    char name[20];      // 20 bytes
    uint16_t blockSize; // 2 bytes
    uint16_t maxBlocks; // 2 bytes
};
```

Functions those are in part1-2 implementation:

uint16_t getCurrentDate()
Returns the current date in FAT date format (YYYYYYYMMMMDDDDD).

uint16_t getCurrentTime()
Returns the current time in FAT time format (HHHHHMMMMMMSSSSS, 2-second increments).

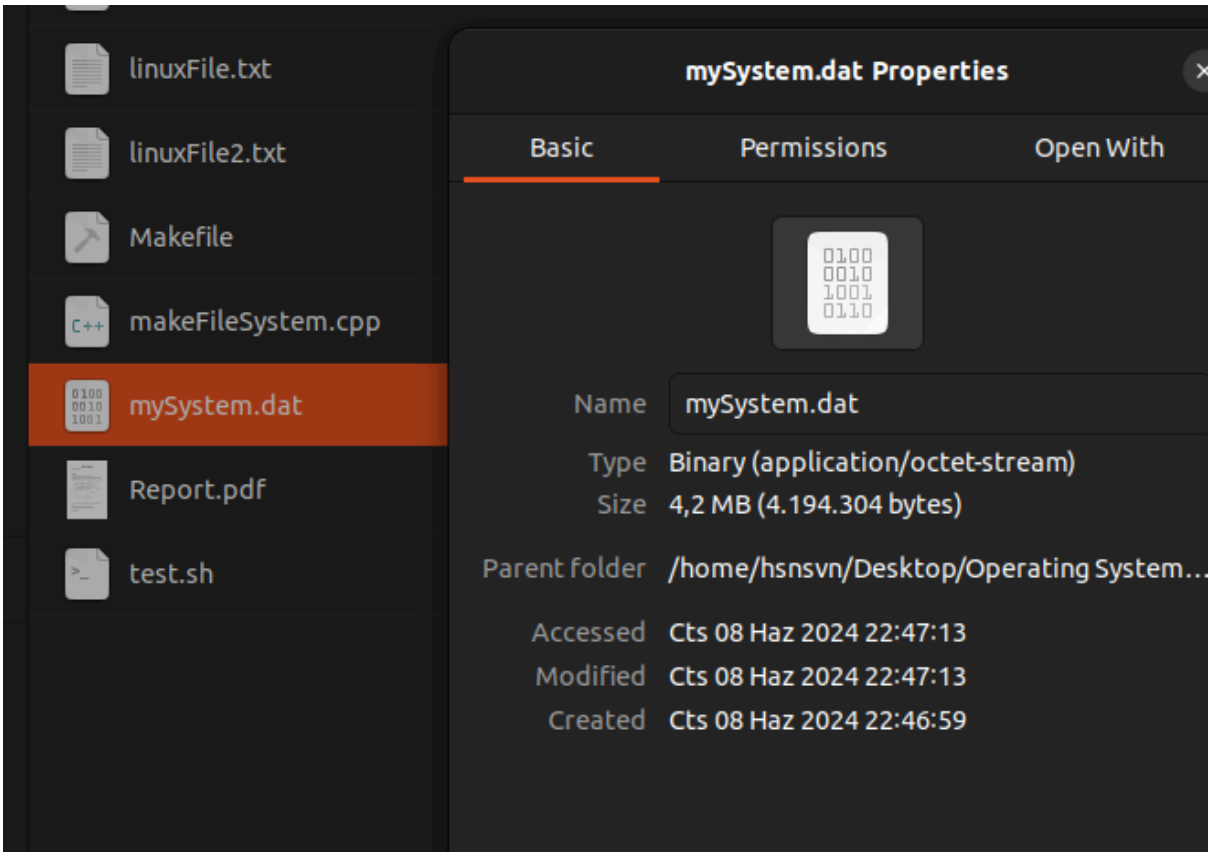
void displayDate(uint16_t date)
Displays the given date in human-readable format (YYYY-MM-DD).

void displayTime(uint16_t time)
Displays the given time in human-readable format (HH:MM).

FileSystem(const char name, uint16_t blockSize, uint16_t maxBlocks)
Constructor for the FileSystem class; initializes the file system with a name, block size, and maximum number of blocks.

void createFileSystem(const char filePath)
Creates the file system on disk by writing the superblock, FAT, bitmap, empty blocks, root directory, and data blocks to a file at the specified file path.

The files system is exactly 4mb but in ubuntu it is shown 4.2mb. we can understand that by calculating 4*1024*1024 which is exactly 4.194.304 bytes



PART3:

File System Management

1. **bool loadFileSystem(const string& fileName, FileSystem& fs);**
    - Loads the file system data from a binary file into the FileSystem structure.
2. **void saveFileSystem(const string& fileName, const FileSystem& fs);**

    - Saves the current state of the FileSystem structure to a binary file.
3. **void initializeFileSystem(FileSystem &fs);**
    - Initializes the file system with a root directory and prepares it for use.

Directory Operations

4. **void makeDirectory(FileSystem &fs, const std::string &directoryPath);**
    - Creates a new directory at the specified path within the file system.
5. **bool removeDirectory(FileSystem &fs, const std::string &directoryPath);**
    - Removes the specified directory from the file system if it is empty.
6. **void listDirectory(const FileSystem &fs, const std::string &directoryPath);**
    - Lists the contents of the specified directory, similar to the ls command in Linux.

File Operations

7. **void addFile(FileSystem &fs, const std::string &fileName, const std::vector<char> &fileData);**
    - Adds a new file with the given data to the file system.
8. **void readFile(FileSystem &fs, const std::string &filePath, const std::string &outputFileName, const std::string& password="");**
    - Reads a file from the file system, checks permissions, and writes the content to an output file.
9. **bool deleteFile(FileSystem &fs, const std::string &filePath);**
    - Deletes a file from the file system and frees the associated data blocks.

File Entry and Data Block Management

10. **FileEntry* findFileEntry(FileSystem &fs, const std::string &filePath, const std::string &password = "");**
    - Finds and returns a pointer to a file entry within the file system, optionally checking for a password.
11. **std::string readDataFromBlock(FileSystem &fs, int blockIndex, size_t fileSize);**
    - Reads data from a specified block and returns it as a string.
12. **void addToRootDirectory(FileSystem& fs, const FileEntry& entry);**
    - Adds a new file entry to the root directory of the file system.

Utility Functions

13. **uint16_t getCurrentDate();**
    - Returns the current date in FAT date format.
14. **uint16_t getCurrentTime();**
    - Returns the current time in FAT time format.
15. **std::string displayDate(uint16_t date);**
    - Converts a FAT date format to a human-readable string.
16. **std::string displayTime(uint16_t time);**
    - Converts a FAT time format to a human-readable string.

Security and Permissions

17. **void addPassword(FileSystem &fs, const std::string &filePath, const std::string &password, const std::string &passwordOld);**
    - Adds or updates the password for a specified file, with optional old password verification.
18. **void chmodFile(FileSystem &fs, const std::string &filePath, uint16_t permissions, const std::string &password="");**
    - Changes the permissions of a specified file, optionally requiring a password.
19. **std::string getChmod(FileSystem &fs, const std::string &filePath);**
    - Retrieves the permissions of a specified file as a string.
20. **std::string getPassword(FileSystem &fs, const std::string &filePath);**
    - Retrieves the password of a specified file as a string.

Debugging and Information

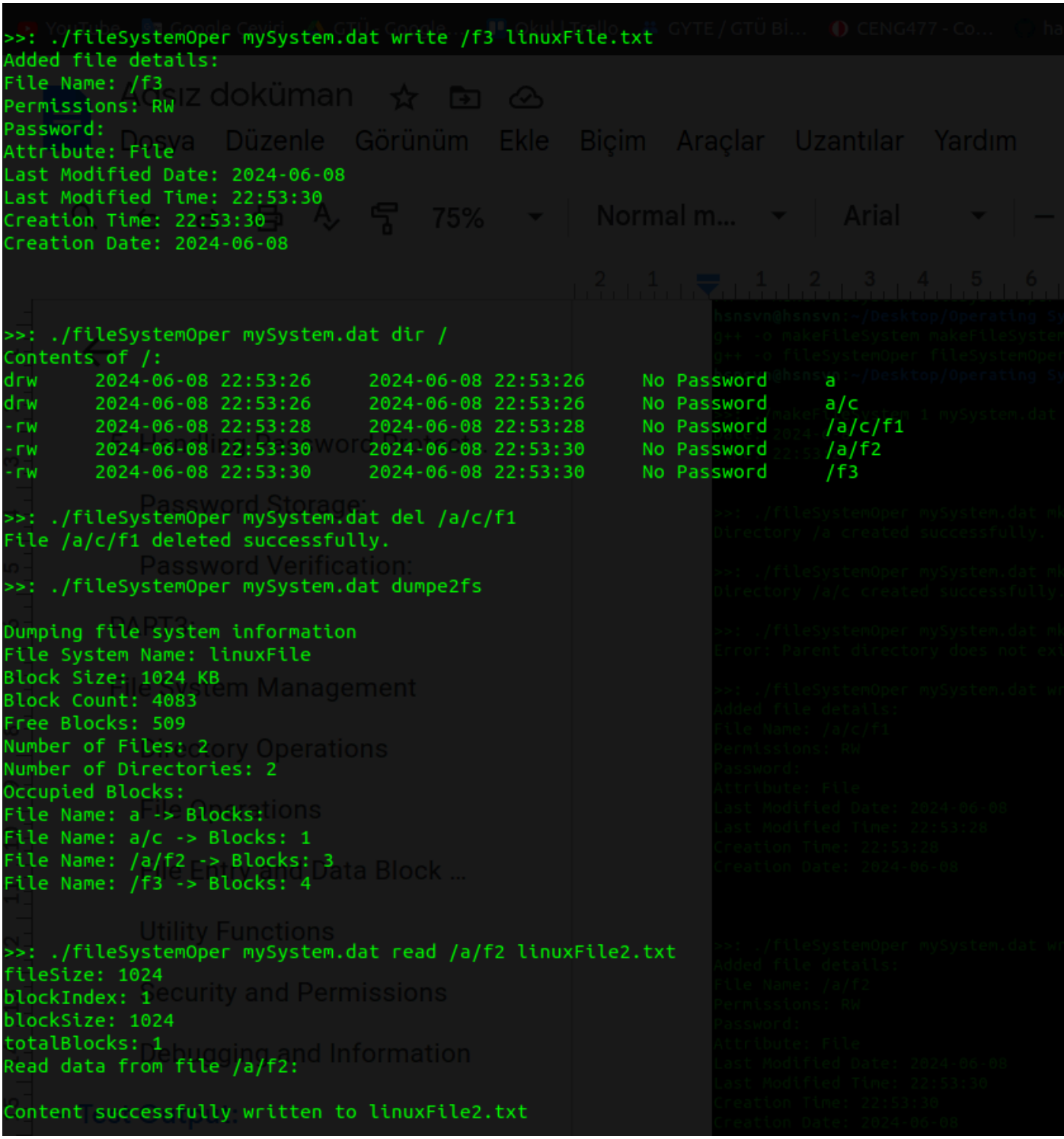
21. **void dumpe2fs(const FileSystem& fs);**
    - Dumps the file system information, including free blocks, file and directory counts, and occupied blocks.
22. **void printOccupiedRootEntries(const FileSystem& fs);**
    - Prints details of all occupied entries in the root directory.

Test Output:

To run:
1- make
2- chmod +x test.sh
3- ./test.sh

The functions works in **1kb** and **0.5** kb as expected.

```
hsnsvn@hsnsvn:~/Desktop/Operating System/hw2/1901042704$ make
g++ -o makeFileSystem makeFileSystem.cpp
g++ -o fileSystemOper fileSystemOper.cpp
hsnsvn@hsnsvn:~/Desktop/Operating System/hw2/1901042704$ ./test.sh

>>: ./makeFileSystem 1 mySystem.dat
Date: 2024-6-8
Time: 22:53:24
32

>>: ./fileSystemOper mySystem.dat mkdir /a
Directory /a created successfully.

>>: ./fileSystemOper mySystem.dat mkdir /a/c
Directory /a/c created successfully.

>>: ./fileSystemOper mySystem.dat mkdir /b/c
Error: Parent directory does not exist.

>>: ./fileSystemOper mySystem.dat write /a/c/f1 linuxFile.txt
Added file details:
File Name: /a/c/f1
Permissions: RW
Password:
Attribute: File
Last Modified Date: 2024-06-08
Last Modified Time: 22:53:28
Creation Time: 22:53:28
Creation Date: 2024-06-08



>>: ./fileSystemOper mySystem.dat write /a/f2 linuxFile.txt
Added file details:
File Name: /a/f2
Permissions: RW
Password:
Attribute: File
Last Modified Date: 2024-06-08
Last Modified Time: 22:53:30
Creation Time: 22:53:30
Creation Date: 2024-06-08
```


>>: ./fileSystemOper mySystem.dat write /f3 linuxFile.txt
Added file details:
File Name: /f3
Permissions: RW
Password:
Attribute: File
Last Modified Date: 2024-06-08
Last Modified Time: 22:53:30
Creation Time: 22:53:30
Creation Date: 2024-06-08
>>: ./fileSystemOper mySystem.dat dir /
Contents of /:
drw 2024-06-08 22:53:26 2024-06-08 22:53:26 No Password a
drw 2024-06-08 22:53:26 2024-06-08 22:53:26 No Password a/c
-rw 2024-06-08 22:53:28 2024-06-08 22:53:28 No Password /a/c/f1
-rw 2024-06-08 22:53:30 2024-06-08 22:53:30 No Password /a/f2
-rw 2024-06-08 22:53:30 2024-06-08 22:53:30 No Password /f3
>>: ./fileSystemOper mySystem.dat del /a/c/f1
File /a/c/f1 deleted successfully.
>>: ./fileSystemOper mySystem.dat dump2fs
Dumping file system information
File System Name: linuxFile
Block Size: 1024 KB
Block Count: 4083
Free Blocks: 509
Number of Files: 2
Number of Directories: 2
Occupied Blocks:
File Name: a -> Blocks:
File Name: a/c -> Blocks: 1
File Name: /a/f2 -> Blocks: 3
File Name: /f3 -> Blocks: 4
>>: ./fileSystemOper mySystem.dat read /a/f2 linuxFile2.txt
fileSize: 1024
blockIndex: 1
blockSize: 1024
totalBlocks: 1
Read data from file /a/f2:
Content successfully written to linuxFile2.txt

```
>>: ./fileSystemOper mySystem.dat chmod /a/f2 -rw
Updated file details:
File Name: /a/f2
Permissions: RW
Password:
Attribute: File
Last Modified Date: 2024-06-08
Last Modified Time: 22:53:30
Creation Time: 22:53:30
Creation Date: 2024-06-08

>>: ./fileSystemOper mySystem.dat read /a/f2 linuxFile2.txt
fileSize: 1024
blockIndex: 1
blockSize: 1024
totalBlocks: 1
Read data from file /a/f2:

Content successfully written to linuxFile2.txt

>>: ./fileSystemOper mySystem.dat chmod /a/f2 +rw
Updated file details:
File Name: /a/f2
Permissions: RW
Password:
Attribute: File
Last Modified Date: 2024-06-08
Last Modified Time: 22:53:30
Creation Time: 22:53:30
Creation Date: 2024-06-08

>>: ./fileSystemOper mySystem.dat addpw /a/f2 1234
Updated file details:
File Name: /a/f2
Permissions: RW
Password: 1234
Attribute: File
Last Modified Date: 2024-06-08
Last Modified Time: 22:53:30
Creation Time: 22:53:30
Creation Date: 2024-06-08
```

```
>>: ./fileSystemOper mySystem.dat read /a/f2 linuxFile2.txt
Error: This file is password protected. Please provide a password.

>>: ./fileSystemOper mySystem.dat read /a/f2 linuxFile2.txt 1234
fileSize: 1024
blockIndex: 1
blockSize: 1024
totalBlocks: 1
Read data from file /a/f2:

Content successfully written to linuxFile2.txt
```