

HW2: FAT-12 File System

Ertugrul Kasikci 200104004097

Wednesday 5th June, 2024

Gebze Technical University
Department of Computer Engineering
CSE 312 / CSE 504 - Operating Systems
Spring 2024

Abstract

This project involves designing and implementing a simplified FAT-like file system in C++. The file system will utilize a FAT-12 structure, supporting block sizes of 0.5 KB and 1 KB. The file system will include various file attributes and support multiple operations similar to Linux file system commands.

Contents

1	Introduction	3
2	Design Specifications	3
2.1	Directory Table and Entries	3
2.2	Block Size and File System Size	3
2.3	Free Blocks Management	4
2.4	Permissions Handling	4
2.5	Password Protection	4
3	Implementation	5
3.1	File System Initialization and Utilities	5
3.2	Directory Operations	6
3.3	File Operations	7
3.4	Permission and Password Handling	7
4	Testing	7
4.1	Testing Commands	8
4.2	Command Explanations	8
4.2.1	Clean and Build	8
4.2.2	Create a File System	8
4.2.3	Make Directory	9
4.2.4	Write Directly to a File	9
4.2.5	Write from One File to Another	9
4.2.6	List Directory Contents	9
4.2.7	Delete a File	9
4.2.8	Dump File System Information	10
4.2.9	Write Directly to an Empty File	10
4.2.10	Read File Content	10
4.2.11	Change File Permissions	10
4.2.12	Add Password Protection	11
4.2.13	Print File System Details	11
5	Conclusion	12

1 Introduction

In this project, a simplified FAT-like file system is designed and implemented. The file system employs a FAT-12 structure to manage files and directories. This report outlines the design and implementation details, along with the functions used to handle various file system operations.

2 Design Specifications

2.1 Directory Table and Entries

The directory table is composed of multiple directory entries. Each entry contains the following attributes:

- char fileName[8];
- char extension[3];
- char attributes;
- char reserved[10];
- char timeField[2];
- char dateField[2];
- uint16_t firstBlockNumber;
- uint32_t size;

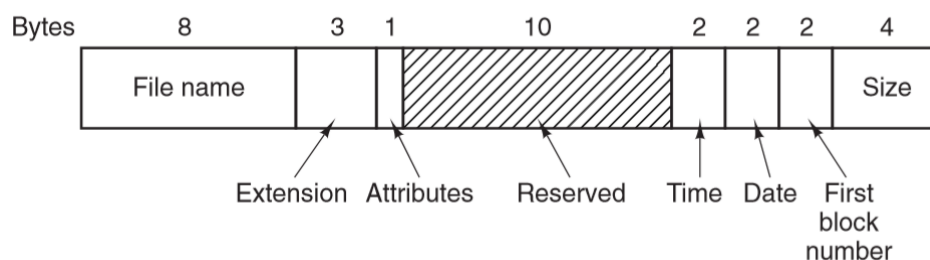


Figure 1: The MS-DOS directory entry.

The directory entry structure is the same with the MS-DOS directory entry as shown in Figure 1.

2.2 Block Size and File System Size

The file system is designed to support FAT-12 with a maximum of 2^{12} blocks. The total size of the file system varies according to the block size used. For instance, if the block size is 1 KB, the total file system size will be approximately 4 MB plus the space required for the FAT and directory entries.

As shown in Figure 2, the maximum partition size for FAT-12 with a block size of 1 KB is 4 MB.

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Figure 2: Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

2.3 Free Blocks Management

Free blocks are managed using a FAT-12 table, where each entry in the table corresponds to a block. The entry indicates whether the block is free or occupied and, if occupied, points to the next block in the file.

The FAT-12 table is implemented with the following structure:

```
static const int totalBlocks = 4096;
struct FATEntry
{
    bool isBusy;
    int nextBlock;
};
FATEntry FAT[totalBlocks];
```

2.4 Permissions Handling

Permissions are handled using attributes field, where specific bits are set to indicate read and write permissions (attributes also indicate if it is a file or folder).

- Read permission: Bit 0 (1 value means reading is permitted)
- Write permission: Bit 1 (1 value means writing is permitted)

2.5 Password Protection

Files can be protected with a password, which is stored in the reserved field. Any operation on a password-protected file requires the correct password to be provided. `void FileSystem::addPassword(const std::string &fileName, const std::string &password)` function is responsible for adding passwords.

FAT

	Busy	Next
0	0	
1	1	-1
2	1	6
3	1	5
4	1	-1
5	1	-1
6	1	3
7	0	

Figure 3: FAT-12 table structure.

3 Implementation

3.1 File System Initialization and Utilities

filesystemExists

```
bool filesystemExists(const std::string &fileName) const;
```

This function checks whether a file system exists by attempting to open the file. If the file can be opened successfully, the function returns **true**; otherwise, it returns **false**.

saveFileSystem

```
void saveFileSystem() const;
```

This function saves the current state of the file system to a file. It writes the block size, FAT entries, and directory entries to the file, ensuring that the file system can be reloaded later.

loadFileSystem

```
void loadFileSystem(const std::string &fileName);
```

This function loads a file system from a file. It reads the block size, FAT entries, and directory entries from the file, reconstructing the file system in memory.

```
File Name: lf, Size: 10, First Block: 4, Attributes: rw, Password: No, Last Mod
File Name: file3, Size: 10, First Block: 7, Attributes: rw, Password: No, Last M
File Name: file2, Size: 10, First Block: 6, Attributes: rw, Password: test1234,
File Name: lf2, Size: 10, First Block: 5, Attributes: rw, Password: No, Last Mod
```

Figure 4: Example of added password.

writeDirectoryEntryToPage

`void writeDirectoryEntryToPage(int block, const DirectoryEntry &dirEntry);`
 This function writes a directory entry to a specified block. It searches for an empty slot within the block and writes the entry, ensuring the directory structure is updated.

saveDirectoryEntry

`void saveDirectoryEntry(const DirectoryEntry &entry);`
 This function saves a directory entry to the file system. It ensures that the entry is correctly written to the appropriate location in the directory structure.

addDirectoryEntryToParent

`void addDirectoryEntryToParent(const DirectoryEntry &entry, int parentBlock);`
 This function adds a directory entry to the parent directory. It writes the entry to the parent block, updating the directory structure to include the new entry.

printBlockContents

`void printBlockContents() const;`
 This function prints the contents of each block in the file system. It helps in visualizing the data stored in each block and can be useful for debugging purposes.

dumpe2fs

`void dumpe2fs();`
 This function provides a summary of the file system, including the block count, free blocks, number of files and directories, and block size. It also lists all occupied blocks and their corresponding file names.

3.2 Directory Operations

listDirectory

`void listDirectory() const;`
 This function lists all directory entries in the file system, displaying the file name, size, first block, attributes, password protection status, and the last modification date and time.

printFileSystem

`void printFileSystem() const;`
 This function prints the entire file system, including the File Allocation Table (FAT) and all directory entries. It provides a comprehensive view of the current state of the file system.

makeDirectory

`void makeDirectory(const std::string &dirName);`
 This function creates a new directory. It first checks if the parent directory exists, allocates a block for the new directory, and then writes the new directory entry to the parent directory's block.

removeDirectory

`void removeDirectory(const std::string &dirName);`
 This function removes a directory. It verifies that the directory exists and is empty before removing it. If the directory is not empty, it returns an error message.

listDirectory (path)

```
void listDirectory(const std::string &path) const;
```

This function lists the contents of a specific directory given by the path. It displays information about each file and directory within the specified path.

printDirectoryPages

```
void printDirectoryPages();
```

This function prints the pages of the directory, displaying the contents of each directory block. It provides a detailed view of the directory structure within the file system.

3.3 File Operations

writeFile

```
void writeFile(const std::string &fileName, const std::string &content);
```

This function writes data to a file. It ensures that the parent directory has write permissions, allocates the necessary blocks, and writes the content to the file system. If the file already exists, it returns an error message.

readFile

```
void readFile(const std::string &fileName, const std::string &outputFile, const std::string &password = "");
```

This function reads data from a file and writes it to an output file. If the file is password protected, the correct password must be provided. It checks for read permissions before performing the read operation.

deleteFile

```
void deleteFile(const std::string &fileName);
```

This function deletes a file from the file system. It removes the directory entry and frees the blocks allocated to the file, ensuring that the file is completely removed from the system.

writeFileToFile

```
void writeFileToFile(const std::string &fileName, const std::string &linuxFileName);
```

This function copies the contents of a file within the file system to a specified Linux file. It ensures that the data is correctly transferred between the file system and the Linux file.

3.4 Permission and Password Handling

changeMode

```
void changeMode(const std::string &fileName, const std::string &permissions);
```

This function changes the read and write permissions of a file. It supports adding and removing read (r) and write (w) permissions using the appropriate permission flags.

addPassword

```
void addPassword(const std::string &fileName, const std::string &password);
```

This function adds a password to a file, providing an additional layer of protection. Subsequent operations on the file will require the correct password to be provided.

4 Testing

In this section, the testing process of the FAT-12 file system through a series of commands is detailed. Each command is explained along with the expected changes in the file system.

4.1 Testing Commands

```
#!/bin/bash

# Clean and build
make clean
make

# Create a file system
./makeFileSystem 1 fileSystem.data

# Run operations
./fileSystemOper fileSystem.data mkdir "/usr"
./fileSystemOper fileSystem.data mkdir "/usr/ysa"
./fileSystemOper fileSystem.data mkdir "/bin/ysa" # Should produce
./fileSystemOper fileSystem.data writeDirect "/usr/ysa/lf" "lf content" # Create lf f
./fileSystemOper fileSystem.data write "/usr/ysa/file1" "/usr/ysa/lf"
./fileSystemOper fileSystem.data write "/usr/file2" "/usr/ysa/lf"
./fileSystemOper fileSystem.data write "/file3" "/usr/ysa/lf"
./fileSystemOper fileSystem.data dir "/"
./fileSystemOper fileSystem.data del "/usr/ysa/file1"
./fileSystemOper fileSystem.data dumpe2fs
./fileSystemOper fileSystem.data writeDirect "/usr/lf2" "" # Create lf2 file
./fileSystemOper fileSystem.data read "/usr/file2" "/usr/lf2"
#cmp /usr/lf2 /usr/ysa/lf
./fileSystemOper fileSystem.data chmod "/usr/file2" "-rw"
./fileSystemOper fileSystem.data dumpe2fs
./fileSystemOper fileSystem.data read "/usr/file2" "/usr/lf2" # Should pro
./fileSystemOper fileSystem.data chmod "/usr/file2" "+rw"
./fileSystemOper fileSystem.data addpw "/usr/file2" "test1234"
./fileSystemOper fileSystem.data read "/usr/file2" "/usr/lf2" # Should pro
./fileSystemOper fileSystem.data read "/usr/file2" "/usr/lf2" "test1234" # Should be
./fileSystemOper fileSystem.data test

rm fileSystem.data
```

4.2 Command Explanations

4.2.1 Clean and Build

```
make clean
make
```

These commands clean any previous build files and compile the project again. This ensures that we are working with the latest version of the code.

4.2.2 Create a File System

```
./makeFileSystem 1 fileSystem.data
```


This command creates a new file system with a block size of 1 KB and stores it in the file `fileSystem.data`.

4.2.3 Make Directory

```
./fileSystemOper fileSystem.data mkdir "/usr"  
./fileSystemOper fileSystem.data mkdir "/usr/ysa"  
./fileSystemOper fileSystem.data mkdir "/bin/ysa" # Should produce an error
```

The first two commands create directories `/usr` and `/usr/ysa` respectively. The third command attempts to create a directory `/bin/ysa` but produces an error since `/bin` does not exist.

4.2.4 Write Directly to a File

```
./fileSystemOper fileSystem.data writeDirect "/usr/ysa/lf" "lf content"
```

This command creates a file `/usr/ysa/lf` and writes the content "lf content" directly to it.

4.2.5 Write from One File to Another

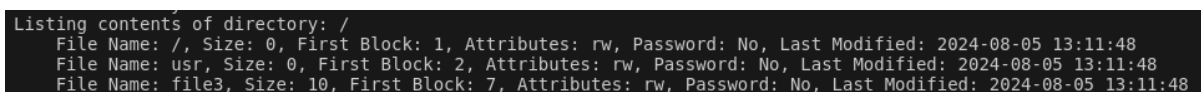
```
./fileSystemOper fileSystem.data write "/usr/ysa/file1" "/usr/ysa/lf"  
./fileSystemOper fileSystem.data write "/usr/file2" "/usr/ysa/lf"  
./fileSystemOper fileSystem.data write "/file3" "/usr/ysa/lf"
```

These commands copy the content of `/usr/ysa/lf` to `/usr/ysa/file1`, `/usr/file2`, and `/file3` respectively.

4.2.6 List Directory Contents

```
./fileSystemOper fileSystem.data dir "/"
```

This command lists the contents of the root directory. The output should show the files and directories present in the root.



```
Listing contents of directory: /  
File Name: /, Size: 0, First Block: 1, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:11:48  
File Name: usr, Size: 0, First Block: 2, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:11:48  
File Name: file3, Size: 10, First Block: 7, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:11:48
```

Figure 5: Output of the `dir` command.

4.2.7 Delete a File

```
./fileSystemOper fileSystem.data del "/usr/ysa/file1"
```

This command deletes the file `/usr/ysa/file1` from the file system.

```
Filesystem Summary:
Block count: 4096
Block size: 1024 bytes
Free blocks: 4089
Occupied blocks: 7
Number of files: 3
Number of directories: 3
Occupied blocks and file names:
Directory: / at block 1, Size: 0, First Block: 1, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:17:44
Directory: usr at block 2, Size: 0, First Block: 2, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:17:44
Directory: ysa at block 3, Size: 0, First Block: 3, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:17:44
File: lf at blocks: 4, Size: 10, First Block: 4, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:17:44
File: file2 at blocks: 6, Size: 10, First Block: 6, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:17:44
File: file3 at blocks: 7, Size: 10, First Block: 7, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:17:44
```

Figure 6: Output of the `dumpe2fs` command.

4.2.8 Dump File System Information

```
./filesystemOper filesystem.data dumpe2fs
```

This command provides detailed information about the file system, including the number of blocks, free blocks, and a list of all occupied blocks and their corresponding files.

4.2.9 Write Directly to an Empty File

```
./filesystemOper filesystem.data writeDirect "/usr/lf2" ""
```

This command creates an empty file `/usr/lf2`.

4.2.10 Read File Content

```
./filesystemOper filesystem.data read "/usr/file2" "/usr/lf2"
```

This command reads the content of `/usr/file2` and writes it to `/usr/lf2`.

4.2.11 Change File Permissions

```
./filesystemOper filesystem.data chmod "/usr/file2" "-rw"
./filesystemOper filesystem.data dumpe2fs
./filesystemOper filesystem.data read "/usr/file2" "/usr/lf2" # Should produce an error
./filesystemOper filesystem.data chmod "/usr/file2" "+rw"
```

The first command removes read and write permissions from `/usr/file2`. The second command shows the permissions are indeed changed. The third command attempts to read the file and produce an error. The fourth command restores the read and write permissions.

```

Permissions changed.
Filesystem Summary:
Block count: 4096
Block size: 1024 bytes
Free blocks: 4088
Occupied blocks: 8
Number of files: 4
Number of directories: 3
Occupied blocks and file names:
Directory: / at block 1, Size: 0, First Block: 1, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
Directory: usr at block 2, Size: 0, First Block: 2, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
Directory: ysa at block 3, Size: 0, First Block: 3, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File: lf at blocks: 4, Size: 10, First Block: 4, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File: file3 at blocks: 7, Size: 10, First Block: 7, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File: lf2 at blocks: 5, Size: 10, First Block: 5, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File: file2 at blocks: 6, Size: 10, First Block: 6, Attributes: --, Password: No, Last Modified: 2024-08-05 13:26:04
Read permission denied.

```

Figure 7: Output of the `dumpe2fs` command after the permission change.

4.2.12 Add Password Protection

```

./fileSystemOper fileSystem.data addpw "/usr/file2" "test1234"
./fileSystemOper fileSystem.data read "/usr/file2" "/usr/lf2" # Should produce an error
./fileSystemOper fileSystem.data read "/usr/file2" "/usr/lf2" "test1234" # Should be successful

```

The first command adds the password "test1234" to /usr/file2. The second command attempts to read the file without a password and should produce an error. The third command reads the file with the correct password.

```

Read permission denied.
Permissions changed.
Password added.
Password is incorrect.
Password is correct.

```

Figure 8: Output of the password change after the permission change.

4.2.13 Print File System Details

```
./fileSystemOper fileSystem.data test
```

This command prints detailed information about the file system, including the FAT table, directory pages, and block contents.

```

FAT Table (Occupied Blocks Only):
Block 0: isBusy = 1, nextBlock = -1
Block 1: isBusy = 1, nextBlock = -1
Block 2: isBusy = 1, nextBlock = -1
Block 3: isBusy = 1, nextBlock = -1
Block 4: isBusy = 1, nextBlock = -1
Block 5: isBusy = 1, nextBlock = -1
Block 6: isBusy = 1, nextBlock = -1
Block 7: isBusy = 1, nextBlock = -1
File Name: /, Size: 0, First Block: 1, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: usr, Size: 0, First Block: 2, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: ysa, Size: 0, First Block: 3, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: lf, Size: 10, First Block: 4, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: file3, Size: 10, First Block: 7, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: file2, Size: 10, First Block: 6, Attributes: rw, Password: test1234, Last Modified: 2024-08-05 13:26:04
File Name: lf2, Size: 10, First Block: 5, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
Directory Pages:
Directory: / at block 1
File Name: /, Size: 0, First Block: 1, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: usr, Size: 0, First Block: 2, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: file3, Size: 10, First Block: 7, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
Directory: usr at block 2
File Name: usr, Size: 0, First Block: 2, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: /, Size: 0, First Block: 1, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: ysa, Size: 0, First Block: 3, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: file2, Size: 10, First Block: 6, Attributes: rw, Password: test1234, Last Modified: 2024-08-05 13:26:04
File Name: lf2, Size: 10, First Block: 5, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
Directory: ysa at block 3
File Name: ysa, Size: 0, First Block: 3, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: usr, Size: 0, First Block: 2, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
File Name: lf, Size: 10, First Block: 4, Attributes: rw, Password: No, Last Modified: 2024-08-05 13:26:04
Block Contents:
Block 1:
/BkYusrBkYfile3#BkY

Block 2:
usrBkY/BkYysaBkYfile2#test1234BkY
lf2#BkY

Block 3:
ysaBkYusrBkYlf#BkY

Block 4:
lf content
Block 5:
lf content
Block 6:
lf content
Block 7:
lf content

```

Figure 9: Output of the `test` command.

5 Conclusion

The project successfully implements all the necessary commands and features as specified in the homework assignment. These include directory creation and removal, file reading and writing, permission changes, and password protection. The file system operations were verified through a series of test scripts to confirm that they work as expected.