Gebze Technical University

Department of Computer Engineering

CSE312 Operating Systems

Homework 2 Report

Burak Yıldırım
1901042609

Contents

1 Introduction **3**

2 Implementation **3**
2.1 Superblock Structure . . . 3
2.2 Directory Entry Structure and Directory Table . . . 4
2.2.1 Directory Entry Structure . . . 4
2.2.2 Directory Table . . . 5
2.3 Handling Free Blocks . . . 5
2.4 Handling Permissions . . . 5
2.4.1 Functions . . . 5
2.5 Handling Password Protection . . . 5
2.5.1 Functions . . . 5
2.6 File System Operations . . . 6
2.6.1 dir . . . 6
2.6.2 mkdir . . . 6
2.6.3 rmdir . . . 7
2.6.4 dumpe2fs . . . 7
2.6.5 write . . . 7
2.6.6 read . . . 8
2.6.7 del . . . 8
2.6.8 chmod . . . 9
2.6.9 addpw . . . 9

3 Testing **10**

1 Introduction

In this homework, we're expected to design and implement a simplified FAT like file system.

2 Implementation

2.1 Superblock Structure

The superblock contains all the key parameters of the file system, and it's stored in the first block of the file system.

```
struct SuperBlock {
    uint16_t blockSize;
    uint16_t blockCount;
    uint16_t fileCount;
    uint16_t directoryCount;
    uint16_t freeBlocks;
    uint16_t fatFirstBlock;
    uint16_t maxEntryPerBlock;
    DirectoryEntry rootDirectory;
} __attribute__((packed));
```

`__attribute__((packed))` is used to ensure that there is no padding applied by the compiler. The members of the structure are:

- `blockSize`: Size of each block in bytes.
- `blockCount`: Number of blocks.
- `fileCount`: Number of files.
- `directoryCount`: Number of directories.
- `freeBlocks`: Number of free blocks.
- `fatFirstBlock`: First block number of the FAT.
- `maxEntryPerBlock`: Number of maximum directory entries per block.
- `rootDirectory`: Directory entry for the root directory.

2.2 Directory Entry Structure and Directory Table

2.2.1 Directory Entry Structure

The directory entry contains the filename, extension, attributes, password, first block number, creation time, last modified time, and size of the file/directory. I designed a similar structure to the one in the textbook. The total size of the directory entry, and the sizes of the filename, extension, attributes, size, and first block number fields are taken from the textbook. I modified the structure to add the creation time, last modification time and password fields. To fit everything into 32 bytes, I used only the 32 bytes of the `time_t` for creation time and last modified time since it'll be valid until 2108, and limited the size of the password field to 6 bytes.

```
struct DirectoryEntry {
    array<uint8_t, MAX_FILENAME> filename;
    array<uint8_t, MAX_EXTENSION> extension;
    uint8_t attributes;
    array<uint8_t, MAX_PASSWORD> password;
    uint16_t firstBlock;
    uint32_t creationTime;
    uint32_t lastModifiedTime;
    uint32_t size;
} __attribute__((packed));
```

`__attribute__((packed))` is used to ensure that there is no padding applied by the compiler, therefore the struct is always 32 bytes in size. The members of the structure are:

- `filename`: Name of the entry. It's limited to `MAX_FILENAME` which is 8.
- `extension`: Extension of the entry. It's limited to `MAX_EXTENSION` which is 3.
- `attributes`: Attributes of the entry. It contains the following information:
  - Bit 1 (`0x01`): Read permission (1 if the entry is readable, 0 if not)
  - Bit 2 (`0x02`): Write permission (1 if the entry is writable, 0 if not)
  - Bit 3 (`0x04`): Type of the entry (1 for directory, 0 for file)
  - Bit 4 (`0x08`): Password protection (1 if the entry is password protected, 0 if not)
- `password`: Password for accessing the entry if it's password protected.
- `firstBlock`: First block number of the entry.
- `creationTime`: Creation time of the entry.
- `lastModifiedTime`: Last modified time of the entry.
- `size`: Size of the entry in bytes.

2.2.2 Directory Table

I do not have a separate directory table variable like an array of directory entries. Instead, if the type of the directory entry is a directory, then the blocks belonging to that entry simply store 16 or 32 directory entries each, depending on the block size.

2.3 Handling Free Blocks

To handle the free blocks, I used a File Allocation Table (FAT) with 4096 entries since the FAT12 has 2^{12} entries. Each entry of the FAT can have one of the following values:

- `FREE` (-2): indicates that this block is not occupied and can be used.
- `END_OF_FILE` (-1): indicates that this is the last block of a directory entry.
- Values between 2 - 4095: indicate the block number of the next block.

The FAT is stored in blocks 1 through 8 or 16, depending on the block size. I also store the number of free blocks in the superblock.

2.4 Handling Permissions

Permissions are handled using the `attributes` byte in the `DirectoryEntry` structure. Permissions are checked before any read or write operations to ensure the appropriate access control.

2.4.1 Functions

- `void changePermissions(const string &path, const string &permissions, const string &password)`: Changes the permissions of the given entry according to the given permissions.
- `bool isReadable(const uint8_t attributes)`: Checks if the given attributes have read permission.
- `bool isWritable(const uint8_t attributes)`: Checks if the given attributes have write permission.

2.5 Handling Password Protection

Passwords are stored in the `DirectoryEntry` structure. If a file is password-protected, the password field of the directory entry is set with that password. Otherwise the password field is empty. Operations on password-protected files require the correct password to proceed.

2.5.1 Functions

- `void addPassword(const string &path, const string &password)`: Adds password to the given file. If the path is a directory, it displays an error.

- `bool checkPassword(DirectoryEntry *entry, const string &password, const string &path, const FileSystemOperation &operation)`: Checks if the given entry is password-protected. Parameters `path` and `operation` are used for error messaging if needed. This function is called in `readFile`, `writeFile`, `deleteFile`, and `changePermissions` functions to check if the file is password-protected.

2.6 File System Operations

Before any of the operations listed below begin, the following sequence is performed:

- File system is opened.
- Superblock, FAT, and the data blocks are read.
- File system is closed.

2.6.1 dir

- **Function**: `void listDirectoryEntry(const string &path, const string &password)`
- **Details**:
  - Checks if the entry exists.
  - Checks if the entry is password-protected. This check always returns false for directories since the `addPassword` function gives an error for directories.
  - Checks if the entry has read permission.
  - Checks if the entry is a file. If it is, displays the details of the file. If not, traverses all the blocks of the directory and displays the details of the directory entries stored in those blocks. Details are the attributes, size, creation date, last modified date, and the name of the entry. Attributes are displayed in a Linux-like manner:
    - r: indicates that entry has read permission.
    - w: indicates that entry has write permission.
    - d: indicates that entry is a directory.
    - p: indicates that entry is password-protected.

2.6.2 mkdir

- **Function**: `void createDirectory(const string &path)`
- **Details**:
  - Checks if the directory already exists.
  - Ensures that the parent entry is present.
  - Confirms that the parent entry is a directory and not a file.
  - Finds an empty directory entry in the blocks of the parent directory, and a free block for the new directory.
  - Adds the dot directories to the first block of the new directory.
  - Updates the FAT.

2.6.3 rmdir

- **Function**: `void removeDirectory(const string &path)`
- **Details**:
  - Ensures the root directory cannot be removed.
  - Verifies the existence of the entry.
  - Confirms that the entry is indeed a directory.
  - Ensures that the parent directory has write permission.
  - Frees all the blocks used by the directory, and removes the directory entry from the parent directory's blocks.
  - If the block of the parent directory that the directory entry was stored becomes empty after removal, then that block is also freed.

2.6.4 dumpe2fs

- **Function**: `void printFileSystemInfo()`
- **Details**:
  - Recursively maps the blocks to filenames excluding the superblock and FAT blocks using the `generateBlockMapping` function.
  - Sorts the results by the block number.
  - Prints the superblock information and the block mapping information.

2.6.5 write

- **Function**: `void writeFile(const string &dest, const string &src, const string &password)`
- **Details**:
  - Verifies that the destination is not the root directory.
  - Verifies the existence of the source file.
  - Checks if the destination entry exists. If it does, the following checks are performed:
    - Confirms that the entry is not a directory.
    - Verifies if the entry is password-protected and checks the password if necessary.
    - Ensures the entry has read permission.
  - Ensures the parent directory is present.
  - Confirms the parent is a directory.
  - Ensures the parent directory has write permission.

  - If destination entry exists, frees all the blocks used by it except the first block, sets the size to 0. If not, finds an empty directory entry in the blocks of the parent directory, and a free block for the new file.
  - Copies the source file to the blocks of the destination file. New blocks are allocated as needed. After copying, size of the destination entry is set to total bytes copied, and the last modified time of the destination entry is set to the current time.

2.6.6 read

- **Function**: `void readFile(const string &src, const string &dest, const string &password)`
- **Details**:
  - Ensures the source is not the root directory.
  - Verifies the existence of the source file.
  - Confirms the source is not a directory.
  - Verifies if the source file is password-protected and checks the password if necessary.
  - Ensures the source file has read permission.
  - Opens the file specified as the destination for writing.
  - Reads data from the source file's blocks and writes them to the destination file.
  - Closes the destination file after writing.

2.6.7 del

- **Function**: `void deleteFile(const string &path, const string &password)`
- **Details**:
  - Ensures the entry is not the root directory.
  - Verifies the existence of the file.
  - Confirms the entry is not a directory.
  - Verifies if the file is password-protected and checks the password if necessary.
  - Ensures the parent directory has write permission.
  - Frees all the blocks used by the file, and removes the directory entry from the parent directory's blocks.
  - If the block of the parent directory that the directory entry was stored becomes empty after deletion, then that block is also freed.

2.6.8 chmod

- **Function**: `void changePermissions(const string &path, const string &permissions, const string &password)`
- **Details**:
  - Ensures the entry is not the root directory.
  - Verifies the existence of the file or directory.
  - Validates the permissions provided.
  - Verifies if the entry is password-protected and checks the password if necessary.
  - Updates the entry's permissions.
  - Sets the last modified time of the entry to the current time.

2.6.9 addpw

- **Function**: `void addPassword(const string &path, const string &password)`
- **Details**:
  - Ensures that the root directory cannot be password-protected.
  - Verifies the existence of the file or directory.
  - Ensures that directories cannot be password-protected.
  - Checks if the entry is already password-protected
  - Checks if the password exceeds 6 characters.
  - Sets the password bit in the attributes field in the entry.
  - Sets the password field of the entry to the given password.
  - Sets the last modified time of the entry to the current time.

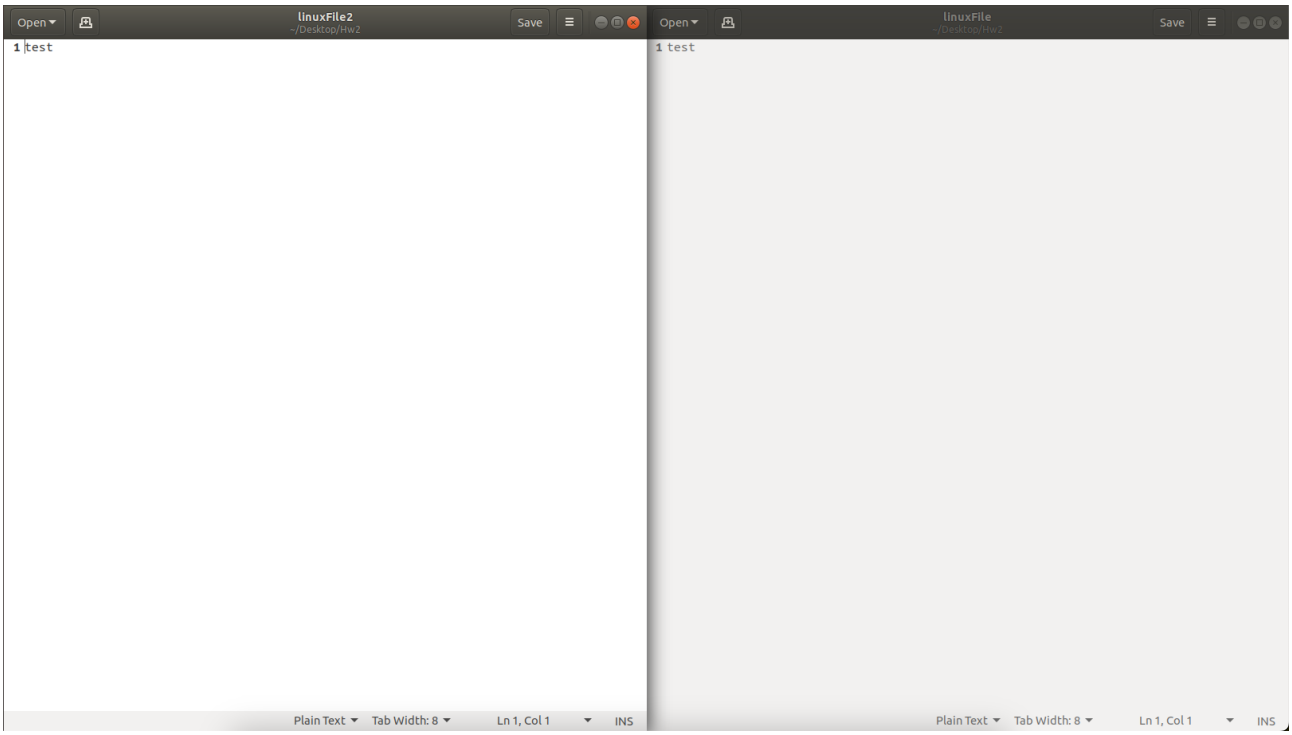
3 Testing

```
burakyildirim@ubuntu:~/Desktop/Hw2$ ./makeFileSystem 1 fileSystem.data
File system 'fileSystem.data' has been successfully created with a block size of 1 KB.
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data mkdir "\usr"
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data mkdir "\usr\ysa"
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data mkdir "\bin\ysa"
mkdir: \bin: No such file or directory
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data write "\usr\ysa\file1" linuxFile
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data write "\usr\file2" linuxFile
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data write "\file3" linuxFile
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data dir "\\"
```

Attributes	Size	Creation Date	Modification Date	Name
-drw	128	06/06/2024 18:16:08	06/06/2024 18:16:51	usr
--rw	4	06/06/2024 18:16:56	06/06/2024 18:16:56	file3

```
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data del "\usr\ysa\file1"
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data dumpe2fs
-----File System Information-----
Block Size: 1024
Block Count: 4096
Free Blocks: 4082
File Count: 2
Directory Count: 3

--------Block Information--------
Block #0: SuperBlock
Block #1: FAT
Block #2: FAT
Block #3: FAT
Block #4: FAT
Block #5: FAT
Block #6: FAT
Block #7: FAT
Block #8: FAT
Block #9: \
Block #10: \usr
Block #11: \usr\ysa
Block #13: \usr\file2
Block #14: \file3
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data read "\usr\file2" linuxFile2
burakyildirim@ubuntu:~/Desktop/Hw2$ cmp linuxFile linuxFile2
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data chmod "\usr\file2" -rw
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data read "\usr\file2" linuxFile2
read: \usr\file2: Permission denied
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data chmod "\usr\file2" +rw
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data addpw "\usr\file2" test12
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data read "\usr\file2" linuxFile2
read: \usr\file2: Password required
burakyildirim@ubuntu:~/Desktop/Hw2$ ./fileSystemOper fileSystem.data read "\usr\file2" linuxFile2 test12
burakyildirim@ubuntu:~/Desktop/Hw2$ cmp linuxFile linuxFile2
burakyildirim@ubuntu:~/Desktop/Hw2$
```


Open
linuxFile2
~/Desktop/Hw2
Save
1 test
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
Open
linuxFile
~/Desktop/Hw2
Save
1 test
Plain Text
Tab Width: 8
Ln 1, Col 1
INS