Operating Systems Homework #2
Part 1 & 2
Gebze Technical University - Computer Engineering

Student Name: ÇAĞRI YILDIZ
Course: CSE 312 / CSE 504 - Spring 2024
Submission Date: 08.06.2024

0. Introduction

This design report outlines the key structures and functionalities of a simplified FAT-like file system, developed in C. It details the architecture and mechanisms used to handle directory management, block allocation, file name handling, permissions, and password protection.

1. Part 1

1.1 Definitions and Constants

To establish a well-defined and manageable file system, specific constants are defined to set the boundaries and operational parameters of the system:

```
 8   #define MAX_BLOCKS 4096
 9   #define MAX_FILE_SYSTEM_SIZE (4 * 1024 * 1024) // 4MB
10   #define SUPER_BLOCK_SIZE 1024
11   #define DIRECTORY_ENTRIES 100
```

- **MAX_BLOCKS:** The maximum number of blocks that the file system can manage.
  #define MAX_BLOCKS 4096

- **MAX_FILE_SYSTEM_SIZE: Defines the total size of the file system, set to 4 MB.**
  #define MAX_FILE_SYSTEM_SIZE (4 * 1024 * 1024) // 4MB

- **SUPER_BLOCK_SIZE:** Specifies the size of the super block within the file system.
  #define SUPER_BLOCK_SIZE 1024

- **DIRECTORY_ENTRIES:** Sets the maximum number of directory entries the system can handle.
  #define DIRECTORY_ENTRIES 100

These constants are critical for ensuring that the file system does not exceed its capacity and operates within predefined limits, enhancing reliability and maintainability.

1.2 Directory Table and Directory Entries

The file system employs a DirectoryTable structure that includes an array of DirectoryEntry items, each representing a file within the system.

- Structure Definition:

```
typedef struct {
    char filename[255];
    unsigned int size;
    unsigned char owner_permission; // bit 0: read, bit 1: write
    time_t last_modification;
    time_t creation_time;
    char password[255];
    unsigned short first_block;
} DirectoryEntry;
```

- Directory Table:

```
typedef struct {
    DirectoryEntry entries[DIRECTORY_ENTRIES];
    unsigned int entry_count;
} DirectoryTable;
```

This setup manages all file metadata, ensuring that each file is indexed for quick access and manipulation within the file system.

1.3 Free Block Management

The FATEntry array constitutes the File Allocation Table (FAT), which maps the usage of data blocks across the file system.

- FAT Structure and Functionality:

```
typedef struct {
    char data[SUPER_BLOCK_SIZE];
} DataBlock;
```

- Managing Free Blocks:

Free blocks are identified when the next_block value is 0xFFFF. The allocate_block() function scans the FAT to find and allocate free blocks:

```
unsigned short allocate_block() {
    for (int i = 0; i < super_block.total_blocks; i++) {
        if (FAT[i].next_block == 0xFFFF) {
            return i;
        }
    }
    return 0xFFFF; // No free block found
}
```

1.4 Permissions

Handling Permissions:

- Permissions are managed using bitwise operations:

```
111     printf((entry->owner_permission & 0b01) ? "Read " : "");
112     printf((entry->owner_permission & 0b10) ? "Write" : "");
```

1.5 Password Protection

Passwords are stored directly within the directory entry and are used to control access to file operations based on string comparison methods.

1.6 Key Functions:

Initialization, File Creation, and State Management:

```
void init_file_system(float block_size) { ... }
void create_file(const char *filename, const char *password) { ... }
void save_file_system(const char *file_name) { ... }
void load_file_system(const char *file_name) { ... }
```

2. Part 2

2.1. Program Initialization and Execution

Initializing the File System:

The file system is initialized based on a command-line argument specifying the block size

```
175
176     float block_size = atof(argv[1]); // Convert command line argument to float
177     const char *file_name = argv[2];
```

2.2 Consistent File System Size

Maintaining a 4 MB File Size:

The size of mySystem.dat is consistently set to exactly 4 MB to meet design specifications, regardless of the actual data stored.

```
121     int fd = open(file_name, O_CREAT | O_WRONLY, 0666);
122     if (fd < 0) {
123         perror("Failed to create file system");
124         exit(EXIT_FAILURE);
125     }
```

2.3 File Operations and Management

Creating and Managing Files:

Files are added and managed within the file system using directory and FAT entries

```
180     create_file("example.txt", "password123");
```

3. Program ScreenShots