北京交通大學
BEIJING JIAOTONG UNIVERSITY

《操作系统》课程

实验报告

实验名称：	实验六 FAT 文件系统模拟设计与实现
姓　　名：	江家玮
学　　号：	22281188
日　　期：	2024.12.13

目录

1 开发环境

Visual Studio Code（VSCode）是一个流行的开源代码编辑器，广泛用于软件开发，尤其适合 Web 开发、Python 开发、C++开发等。它是由微软开发的，具有强大的功能和灵活的扩展性。以下是 VSCode 开发环境的简要介绍：

（1）核心特点

- ✓ 轻量级：VSCode 是一个轻量级的代码编辑器，启动速度快，占用资源少，适合快速开发和调试。
- ✓ 跨平台支持：支持 Windows、macOS 和 Linux 系统，可以在多种平台上使用。
- ✓ 多语言支持：内置对多种编程语言的支持，如 JavaScript、Python、C++、Java、Go、PHP 等。
- ✓ 智能代码补全：通过 IntelliSense 提供智能代码补全，帮助开发者更高效地编写代码。
- ✓ 内置终端：可以直接在编辑器内运行终端命令，便于开发过程中执行脚本、运行应用等。
- ✓ 调试功能：VSCode 提供了强大的调试功能，支持设置断点、查看变量、调用栈等，能够帮助开发者快速定位代码中的问题。

（2）扩展性

- ✓ 插件市场：VSCode 拥有一个庞大的插件市场，你可以根据自己的需要安装各种插件，增加对特定编程语言、框架或工具的支持。例如，Python、Node.js、Docker、Git 等都有相关插件。
- ✓ 自定义设置：你可以根据个人喜好调整编辑器的外观和行为，支持主题、字体、快捷键、工作区设置等的定制。
- ✓ 集成 Git：VSCode 集成了 Git，可以直接在编辑器内查看、提交和管理代码版本控制。

（3）常用功能

- ✓ 代码高亮：支持语法高亮显示，让代码更易读和理解。
- ✓ 代码片段：可以自定义代码片段，提高编码效率。

- ✓ 文件管理：支持多文件编辑、拖放文件、文件资源管理等，便于管理项目文件。
- ✓ 实时预览：对于 Web 开发，VSCode 可以通过插件支持实时预览，方便查看 HTML/CSS/JavaScript 的效果。

（4）调试与终端

- ✓ 集成调试：VSCode 内置调试工具，支持多种编程语言的调试，开发者可以设置断点、查看变量值、查看堆栈信息等。
- ✓ 集成终端：VSCode 提供内置终端功能，支持运行 Shell、PowerShell、Command Prompt 等命令，开发者无需切换到外部终端，便于多任务操作。

（5）工作区与项目管理

- ✓ 工作区：VSCode 支持“工作区”概念，你可以为不同的项目设置不同的配置和环境，确保项目之间的设置不会互相影响。
- ✓ 任务自动化：通过集成任务和自定义脚本，你可以将常见的操作（如构建、测试、部署）自动化，提高开发效率。

2 运行环境

VSCode 的运行环境是指支持和运行 Visual Studio Code 编辑器的硬件和软件环境。

（1）操作系统要求

VSCode 支持以下操作系统：

- ✓ Windows：Windows 7、Windows 8、Windows 10、Windows 11（32 位和 64 位版本都支持）。
- ✓ macOS：支持 macOS 10.11 或更高版本。
- ✓ Linux：支持大多数 Linux 发行版（如 Ubuntu、Debian、Fedora、Red Hat、Arch 等），通常需要 64 位操作系统。

（2）硬件要求

- ✓ 内存：至少 1 GB 的 RAM（推荐 2 GB 或更多）。
- ✓ 处理器：任何现代的处理器（Intel Core i3 或更高，AMD 同类处理器）。
- ✓ 存储：至少 200 MB 的可用磁盘空间来安装 VSCode。

VSCode 是一个跨平台、轻量级但功能强大的编辑器，能够在各种操作系统上运行。它的运行环境主要依赖于操作系统、硬件资源、必要的开发工具（如 Git、Node.js、编译器等）以及通过插件扩展的语言和框架支持。理解和配置这些运行环境能帮助开发者充分发挥 VSCode 的潜力，提升开发效率。

3 测试环境

VSCode 的测试环境是指在 VSCode 中配置和运行测试代码的环境，包括如何设置自动化测试框架、如何调试测试代码以及如何使用插件和工具来提高测试过程的效率。VSCode 本身并不包含完整的测试框架，但它为开发者提供了良好的支持，可以集成多种测试框架并进行调试。VSCode 支持多种编程语言的测试框架，包括 JavaScript、Python、Java、C++ 等，且通过安装插件，可以使 VSCode 成为一个强大的测试环境。本实验采用 C++的测试框架。

4 实验目的

探索、理解并掌握 FAT 文件系统的组织结构、设计原理和编程设计要旨。

5 实验内容

利用标准 C 语言（或结合汇编语言），分析、设计与实现 FAT 文件系统，支持 FAT 文件系统格式的模拟磁盘卷（暨磁盘映像文件）及其中的目录与文件的存取操作，包括磁盘卷格式化、创建目录、改变当前目录（或称为进入指定目录）、重命名目录、显示目录、删除目录、创建空文件、重命名文件、重定向写文件、显示文件内容、删除文件等，并开展与虚拟机平台上实用操作系统（譬如 DOS、Windows 或 Linux）之间的互操作测试验证。

6 实验代码分析

6.1 代码主要组成部分

该代码实现了一个简单的 FAT16 文件系统 模拟程序，主要用于演示基本的文件和目录管理操作。代码的主要组成部分如下：

(1) 文件系统数据结构定义：定义了两种数据结构

- ✓ DirectoryEntry 结构体：表示文件系统中的每个目录条目，包含文件名、文件所在的起始块、文件大小以及文件内容。
- ✓ FileSystem 结构体：表示整个文件系统，包含卷名、总块数、剩余空闲块数、文件目录数组和当前路径。

(2) 磁盘格式化功能：

- ✓ formatDisk 函数：通过用户输入卷名和总块数来初始化文件系统，分配目录空间并将所有目录条目初始化为空。格式化后，文件系统处于初始状态，用户可以进行目录和文件操作。

(3) 目录管理操作：

- ✓ 创建目录：createDirectory 函数用于创建一个新目录，将其添加到文件系统的目录数组中。
- ✓ 更改当前目录：changeDirectory 函数允许用户更改当前工作目录，将当前路径更新为目标目录。
- ✓ 重命名目录：renameDirectory 函数用于重命名一个已存在的目录。
- ✓ 删除目录：deleteDirectory 函数用于删除指定的目录，并将该目录条目的内容清空。

(4) 文件管理操作：

- ✓ 创建文件：createFile 函数用于在文件系统中创建新文件，分配一个磁盘块并将文件名和文件内容初始化为空。
- ✓ 重命名文件：renameFile 函数允许用户重命名文件。
- ✓ 写入文件：writeFile 函数用于向指定文件写入内容，并更新文件的大小和内容。
- ✓ 删除文件：deleteFile 函数用于删除指定文件并释放相应的磁盘块。

(5) 文件/目录内容显示：

- ✓ displayDirectory 函数显示当前文件系统中的所有目录，列出已创建的目录名称。
- ✓ displayFileContent 函数显示指定文件的内容。

(6) 主函数：

- ✓ main 函数提供了一个交互式菜单，用户可以通过输入不同的选项执行文件系统的操作，如格式化磁盘、创建目录、创建文件、重命名文件等。程序通过一个循环持续接收用户输入并执行相应操作，直到用户选择退出。

6.2 代码详解

6.2.1 全局数据结构

```
typedef struct {
    char filename[MAX_FILENAME_LENGTH];
    int start_block;
    int size;
    char data[MAX_FILE_SIZE];
} DirectoryEntry;

typedef struct {
    char volume_name[MAX_FILENAME_LENGTH];
    int total_blocks;
    int free_blocks;
    DirectoryEntry* directory;
    char current_path[MAX_FILENAME_LENGTH];
} FileSystem;

FileSystem fs;
```

- ✓ **功能：**
  - ■ 用于定义文件系统的基本结构，DirectoryEntry 表示目录项，FileSystem 表示文件系统，包括磁盘的基本信息和当前路径。
- ✓ **具体步骤：**
  - ■ 定义结构体 DirectoryEntry 和 FileSystem。
  - ■ DirectoryEntry 包含文件的名称、起始块、大小和数据。

- FileSystem 包含磁盘名称、总块数、剩余块数、目录条目和当前路径。
- 定义 fs 作为全局的文件系统对象。

6.2.2 formatDisk（格式化磁盘）

```
1. void formatDisk() {
2.     printf("Enter volume name: ");
3.     scanf("%s", fs.volume_name);
4.     printf("Enter total number of blocks: ");
5.     scanf("%d", &fs.total_blocks);
6.     fs.free_blocks = fs.total_blocks;
7.     fs.directory = (DirectoryEntry*)malloc(sizeof(DirectoryEntry) * fs.total_blocks);
8.     for (int i = 0; i < fs.total_blocks; i++) {
9.         strcpy(fs.directory[i].filename, "");
10.        fs.directory[i].start_block = -1;
11.        fs.directory[i].size = 0;
12.    }
13.    strcpy(fs.current_path, "/");
14.    printf("Disk formatted successfully.\n");
15.}
```

✓ **功能：**

- 初始化文件系统，格式化磁盘，清空所有的目录和文件条目。

✓ **具体步骤：**

- 输入磁盘名称和总块数，更新文件系统的 volume_name 和 total_blocks。
- 将 free_blocks 设置为总块数，表示所有块目前都可用。
- 使用 malloc 动态分配内存，为目录条目创建空间。
- 循环清空所有目录条目，设置文件名为空、起始块为 -1、文件大小为 0。
- 设置当前路径为根目录/。
- 输出格式化成功的提示。

6.2.3 createDirectory（创建目录）

```
1. void createDirectory() {
2.     char dirname[MAX_FILENAME_LENGTH];
3.     printf("Enter directory name: ");
4.     scanf("%s", dirname);
5.     for (int i = 0; i < fs.total_blocks; i++) {
```

```
        if (strcmp(fs.directory[i].filename, "") == 0) {
            strcpy(fs.directory[i].filename, dirname);
            fs.directory[i].start_block = -1;
            fs.directory[i].size = 0;
            printf("Directory created successfully.\n");
            return;
        }
    }
    printf("No free space for directory creation.\n");
}
```

✓ **功能：**

- 在文件系统中创建一个新目录。

✓ **具体步骤：**

- 输入目录名称。
- 遍历目录条目，找到第一个空闲位置（即文件名为空的条目）。
- 在该位置创建新目录，设置目录名称，起始块为-1，大小为 0。
- 输出目录创建成功的信息。
- 如果没有空闲位置，输出“没有可用空间”提示。

6.2.4 changeDirectory（更改当前目录）

```
void changeDirectory() {
    char dirname[MAX_FILENAME_LENGTH];
    printf("Enter directory name: ");
    scanf("%s", dirname);
    for (int i = 0; i < fs.total_blocks; i++) {
        if (strcmp(fs.directory[i].filename, dirname) == 0) {
            strcat(fs.current_path, dirname);
            strcat(fs.current_path, "/");
            printf("Directory changed to '%s'.\n", fs.current_path);
            return;
        }
    }
    printf("Directory '%s' not found.\n", dirname);
}
```

✓ **功能：**

- 更改当前的工作目录，并更新 current_path。

✓ **具体步骤：**

- 输入要切换到的目录名称。
- 遍历文件系统的目录条目，查找是否存在该目录。
- 如果目录存在，更新 current_path，将该目录名称附加到当前路径末尾。
- 输出目录切换成功的信息。
- 如果目录不存在，输出“目录未找到”的提示。

6.2.5 renameDirectory（重命名目录）

```
1. void renameDirectory() {
2.     char oldname[MAX_FILENAME_LENGTH], newname[MAX_FILENAME_LENGTH];
3.     printf("Enter old directory name: ");
4.     scanf("%s", oldname);
5.     printf("Enter new directory name: ");
6.     scanf("%s", newname);
7.     for (int i = 0; i < fs.total_blocks; i++) {
8.         if (strcmp(fs.directory[i].filename, oldname) == 0) {
9.             strcpy(fs.directory[i].filename, newname);
10.            printf("Directory renamed successfully.\n");
11.            return;
12.        }
13.    }
14.    printf("Directory '%s' not found.\n", oldname);
15.}
16.
```

✓ **功能：**

- 重命名一个现有的目录。

✓ **具体步骤：**

- 输入旧的目录名称和新的目录名称。
- 遍历目录条目，查找是否存在该目录。
- 如果存在，修改目录名称为新名称，并输出成功提示。
- 如果目录不存在，输出“目录未找到”的提示。

6.2.6 displayDirectory（显示当前目录）

```
1. void displayDirectory() {
2.     printf("Current Disk: %s\n", fs.volume_name);
3.     printf("Current Path: %s\n", fs.current_path);
```

```
    printf("Directory listing:\n");
    for (int i = 0; i < fs.total_blocks; i++) {
        if (strcmp(fs.directory[i].filename, "") != 0) {
            printf("- %s\n", fs.directory[i].filename);
        }
    }
}
```

✓ **功能：**

- 显示当前磁盘、当前路径以及目录中的所有文件和目录。

✓ **具体步骤：**

- 输出当前磁盘和当前路径。
- 遍历目录条目，列出所有非空的文件或目录名称。
- 如果目录中没有文件或目录，则不输出任何内容。

6.2.7 deleteDirectory（删除目录）

```
void deleteDirectory() {
    char dirname[MAX_FILENAME_LENGTH];
    printf("Enter directory name: ");
    scanf("%s", dirname);
    for (int i = 0; i < fs.total_blocks; i++) {
        if (strcmp(fs.directory[i].filename, dirname) == 0) {
            strcpy(fs.directory[i].filename, "");
            printf("Directory deleted successfully.\n");
            return;
        }
    }
    printf("Directory '%s' not found.\n", dirname);
}

```

✓ **功能：**

- 删除指定的目录。

✓ **具体步骤：**

- 输入要删除的目录名称。
- 遍历目录条目，查找是否存在该目录。
- 如果目录存在，清空该目录条目的文件名，表示目录已删除。
- 输出删除成功的信息。

- 如果目录不存在，输出“目录未找到”的提示。

6.2.8 createFile（创建文件）

```
void createFile() {
    char filename[MAX_FILENAME_LENGTH];
    printf("Enter file name: ");
    scanf("%s", filename);
    for (int i = 0; i < fs.total_blocks; i++) {
        if (strcmp(fs.directory[i].filename, "") == 0) {
            strcpy(fs.directory[i].filename, filename);
            fs.directory[i].start_block = i;
            fs.directory[i].size = 0;
            fs.directory[i].data[0] = '\0';
            fs.free_blocks--;
            printf("File created successfully.\n");
            return;
        }
    }
    printf("No free space for file creation.\n");
}

```

✓ **功能：**

- 创建一个新文件，并分配目录条目。

✓ **具体步骤：**

- 输入文件名称。
- 遍历文件系统的目录条目，找到第一个空闲位置。
- 创建新文件，并将文件名称、起始块、文件大小和内容初始化。
- 输出文件创建成功的信息。
- 如果没有可用空间，输出“没有可用空间”的提示。

6.2.9 renameFile（重命名文件）

```
void renameFile() {
    char oldname[MAX_FILENAME_LENGTH], newname[MAX_FILENAME_LENGTH];
    printf("Enter old file name: ");
    scanf("%s", oldname);
    printf("Enter new file name: ");
```

```
    scanf("%s", newname);
    for (int i = 0; i < fs.total_blocks; i++) {
        if (strcmp(fs.directory[i].filename, oldname) == 0) {
            strcpy(fs.directory[i].filename, newname);
            printf("File renamed successfully.\n");
            return;
        }
    }
    printf("File '%s' not found.\n", oldname);
}
```

✓ **功能：**

- 重命名文件。

✓ **具体步骤：**

- 输入旧文件名称和新文件名称。
- 查找文件是否存在。
- 如果文件存在，修改文件名称。
- 输出重命名成功的信息。
- 如果文件不存在，输出“文件未找到”的提示。

6.2.10 writeFile（写入文件内容）

```
void writeFile() {
    char filename[MAX_FILENAME_LENGTH];
    printf("Enter file name: ");
    scanf("%s", filename);
    for (int i = 0; i < fs.total_blocks; i++) {
        if (strcmp(fs.directory[i].filename, filename) == 0) {
            int block_index = fs.directory[i].start_block;

            if (block_index == -1) {
                printf("File '%s' not found.\n", filename);
                return;
            }
            printf("Enter file content:\n");
            char content[MAX_FILE_SIZE];
            scanf("%s", content);
            strcpy(fs.current_path, filename);
            fs.directory[i].size = strlen(content);
            strcpy(fs.directory[i].data, content);
            printf("File '%s' written successfully.\n", filename);
```

```
            return;
        }
    }

    printf("File '%s' not found.\n", filename);
}
```

✓ **功能：**

- 向文件写入内容。

✓ **具体步骤：**

- 输入要写入内容的文件名称。
- 查找文件是否存在。
- 如果文件存在，输入文件内容并更新文件的大小和内容字段。
- 输出文件写入成功的提示。

6.2.11 displayFileContent（显示文件内容）

```
void displayFileContent() {
    char filename[MAX_FILENAME_LENGTH];
    printf("Enter file name: ");
    scanf("%s", filename);
    for (int i = 0; i < fs.total_blocks; i++) {
        if (strcmp(fs.directory[i].filename, filename) == 0) {
            printf("Content of file '%s':\n%s\n", filename, fs.directory[i].data);
            return;
        }
    }
    printf("File '%s' not found.\n", filename);
}

```

✓ **功能：**

- 显示文件的内容。

✓ **具体步骤：**

- 输入文件名称。
- 查找文件是否存在。
- 如果文件存在，输出其内容。
- 如果文件不存在，输出“文件未找到”的提示。

6.2.12 deleteFile（删除文件）

```
void deleteFile() {
    char filename[MAX_FILENAME_LENGTH];
    printf("Enter file name: ");
    scanf("%s", filename);
    for (int i = 0; i < fs.total_blocks; i++) {
        if (strcmp(fs.directory[i].filename, filename) == 0) {
            strcpy(fs.directory[i].filename, "");
            fs.directory[i].start_block = -1;
            fs.directory[i].size = 0;
            fs.free_blocks++;
            printf("File deleted successfully.\n");
            return;
        }
    }
    printf("File '%s' not found.\n", filename);
}
```

✓ **功能：**

- 删除指定的文件。

✓ **具体步骤：**

- 输入要删除的文件名称。
- 查找文件是否存在。
- 如果文件存在，清空文件条目的数据，表示文件已删除。
- 输出文件删除成功的提示。
- 如果文件不存在，输出“文件未找到”的提示。

6.2.13 main 函数

```
int main() {
    int choice;
    printf("---------------------------------- FAT16 File System ----------------------------------\n");
    printf("1. Format Disk        2. Create Directory    3. Change Directory    4. Rename Directory\n");
    printf("5. Display Directory   6. Delete Directory    7. Create File         8. Rename File\n");
    printf("9. Write File         10. Display File       11. Delete File         0. Exit\n");
    do {
        printf("\nCurrent Disk: %s   Current Path: %s\n", fs.volume_name, fs.current_path);
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: formatDisk(); break;
```

```
            case 2: createDirectory(); break;
            case 3: changeDirectory(); break;
            case 4: renameDirectory(); break;
            case 5: displayDirectory(); break;
            case 6: deleteDirectory(); break;
            case 7: createFile(); break;
            case 8: renameFile(); break;
            case 9: writeFile(); break;
            case 10: displayFileContent(); break;
            case 11: deleteFile(); break;
            case 0: break;
            default: printf("Invalid choice.\n");
        }
    } while (choice != 0);
    free(fs.directory);
    return 0;
}
```

✓ **功能：**

- 提供一个菜单驱动的用户界面，允许用户进行文件系统的管理操作。

✓ **具体步骤：**

- 打印菜单，列出所有可用操作。
- 循环显示菜单并读取用户输入的操作。
- 根据用户的选择，调用相应的函数。
- 用户选择退出时，释放资源并结束程序。

7 关键数据结构和算法流程

7.1 关键数据结构

（1）DirectoryEntry：

✓ filename[MAX_FILENAME_LENGTH]: 文件名，存储文件的名称。

✓ start_block: 文件数据起始的磁盘块索引。

✓ size: 文件大小（以字节为单位）。

✓ data[MAX_FILE_SIZE]: 文件内容的数据存储区域。

（2）FileSystem：

✓ volume_name[MAX_FILENAME_LENGTH]: 磁盘的卷标名称。

- ✓ total_blocks: 磁盘的总块数。
- ✓ free_blocks: 当前可用的磁盘块数。
- ✓ directory: 指向磁盘目录的指针，存储所有目录项。
- ✓ current_path[MAX_FILENAME_LENGTH]: 当前工作路径，表示文件系统的当前目录。

7.2 核心算法流程

（1）初始化：

- ✓ 格式化磁盘：用户输入磁盘的卷名和磁盘块数，初始化文件系统并分配相应的内存空间存储磁盘目录。
- ✓ 每个磁盘块对应一个 DirectoryEntry 结构，表示一个文件或目录的元数据。
- ✓ 初始化 current_path 为根目录 /。

（2）格式化磁盘：

- ✓ 在 formatDisk 函数中，用户提供卷标名称和磁盘块数，系统创建磁盘并分配目录项。
- ✓ 所有的目录项初始时为空，且文件大小为 0。

（3）创建目录：

- ✓ 用户输入目录名称，系统遍历目录表查找空闲位置，将目录名称和相关数据填入空目录项，分配相应的磁盘块。
- ✓ 目录创建成功后，更新 free_blocks 数量。

（4）更改目录：

- ✓ 用户输入目标目录名称，系统检查该目录是否存在，若存在则更新当前路径 current_path。
- ✓ 路径格式为：/root/dir1/dir2/。

（5）重命名目录：

- ✓ 用户输入旧目录名称和新目录名称，系统查找该目录项并修改目录名称。

（6）删除目录：

- ✓ 用户输入目录名称，系统查找该目录并删除对应的目录项，释放磁盘空间。

（7）创建文件：

- ✓ 用户输入文件名，系统查找空闲目录项，创建文件并分配磁盘块。
- ✓ 文件的起始块为该目录项的 start_block。

（8）重命名文件：

✓ 用户输入旧文件名称和新文件名称，系统查找该文件并修改文件名。

（9）写文件：

✓ 用户输入文件名称并编辑文件内容，系统根据文件大小更新该文件的内容和大小。

✓ 文件数据存储在 data 字段中。

（10）显示文件内容：

✓ 用户输入文件名称，系统查找文件并显示其内容。

（11）删除文件：

✓ 用户输入文件名称，系统查找文件并删除，释放磁盘块。

8 实验结果展示

```
---------------------------------- FAT16 File System ----------------------------------
1. Format Disk         2. Create Directory    3. Change Directory    4. Rename Directory
5. Display Directory   6. Delete Directory    7. Create File         8. Rename File
9. Write File         10. Display File       11. Delete File         0. Exit

Current Disk:    Current Path:
Enter your choice: 1
Enter volume name: test_1
Enter total number of blocks: 256
Disk formatted successfully.
```

图 8-1 磁盘卷格式化

```
Current Disk: test_1   Current Path: /
Enter your choice: 2
Enter directory name: TRAIN_1
Directory created successfully.
```

图 8-2 创建目录

```
Current Disk: test_1   Current Path: /
Enter your choice: 3
Enter directory name: TRAIN_1
Directory changed to '/TRAIN_1/'.
```

图 8-3 改变当前目录

```
Current Disk: test_1   Current Path: /TRAIN_1/
Enter your choice: 4
Enter old directory name: TRAIN_1
Enter new directory name: train_1
Directory renamed successfully.
```

图 8-4 重命名目录

```
Current Disk: test_1   Current Path: /TRAIN_1/
Enter your choice: 5
Current Disk: test_1
Current Path: /TRAIN_1/
Directory listing:
- train_1
```

图 8-5 显示目录

```
Current Disk: test_1   Current Path: /TRAIN_1/
Enter your choice: train_1
Enter directory name: Directory deleted successfully.
```

图 8-6 删除目录

```
Current Disk: test_1   Current Path: /TRAIN_1/
Enter your choice: 7
Enter file name: book
File created successfully.
```

图 8-7 创建空文件

```
Current Disk: test_1   Current Path: /TRAIN_1/
Enter your choice: 8
Enter old file name: 1
Enter new file name: kk
File renamed successfully.
```

图 8-8 重命名文件

```
Current Disk: test_1   Current Path: /TRAIN_1/
Enter your choice: 9
Enter file name: kk
Enter file content:
kklikeshim
File 'kk' written successfully.
```

图 8-9 重定向写文件

```
Current Disk: test_1   Current Path: kk
Enter your choice: 10
Enter file name: kk
Content of file 'kk':
kklikeshim
```

图 8-10 显示文件内容

```
Current Disk: test_1   Current Path: kk
Enter your choice: 11
Enter file name: kk
File deleted successfully.
```

图 8-11 删除文件

```
kk@kk-virtual-machine:~/桌面$ sudo mkdir /mnt/mydisk/mydir
kk@kk-virtual-machine:~/桌面$ echo "Hello,FAT file system!" > /mnt/mydisk/myfile.txt
bash: /mnt/mydisk/myfile.txt: 权限不够
kk@kk-virtual-machine:~/桌面$ sudo echo "Hello,FAT file system!" > /mnt/mydisk/myfile.txt
bash: /mnt/mydisk/myfile.txt: 权限不够
kk@kk-virtual-machine:~/桌面$ ls -ld /mnt/mydisk
drwxr-xr-x 3 root root 512  1月  1  1970 /mnt/mydisk
kk@kk-virtual-machine:~/桌面$ echo "Hello,FAT file system!" | sudo tee  /mnt/mydisk/myfile.tx
Hello,FAT file system!
kk@kk-virtual-machine:~/桌面$ ls /mnt/mydisk
mydir  myfile.txt
kk@kk-virtual-machine:~/桌面$ echo "kk is a good girl" | sudo tee  /mnt/mydisk/myfile.txt
kk is a good girl
kk@kk-virtual-machine:~/桌面$ mv /mnt/mydisk/myfile.txt /mnt/mydisk/mynewfile.txt
mv: 无法将 '/mnt/mydisk/myfile.txt' 移动至 '/mnt/mydisk/mynewfile.txt': 权限不够
kk@kk-virtual-machine:~/桌面$ sudo mv /mnt/mydisk/myfile.txt /mnt/mydisk/mynewfile.txt
kk@kk-virtual-machine:~/桌面$ sudo mv /mnt/mydisk/mydir /mnt/mydisk/newdir
kk@kk-virtual-machine:~/桌面$ ls /mnt/mydisk
mynewfile.txt  newdir
kk@kk-virtual-machine:~/桌面$ sudo rm /mnt/mydisk/mynewfile.txt
kk@kk-virtual-machine:~/桌面$ sudo rmdir /mnt/mydisk/newdir
kk@kk-virtual-machine:~/桌面$ sudo umount /mnt/mydisk
```

图 8-12 文件系统交互性测试

9 技术难点及解决方案

9.1 内存管理与动态分配问题

（1）技术难点

在此实验中，文件系统使用动态分配内存来模拟磁盘和文件的存储，这会涉及如何有效管理内存，特别是在创建、删除文件和目录时。如果内存分配不当，可能导致内存泄漏或访问无效内存的问题。

（2）解决方案：

- ✓ 在程序初始化时，使用 malloc 动态分配内存并创建一个固定大小的目录（DirectoryEntry 数组）。程序结束时，使用 free 释放已分配的内存，以避免内存泄漏。
- ✓ 在文件和目录的创建、删除过程中，始终小心地检查内存是否被正确分配或释放。使用 strcpy 等字符串操作时，确保不会发生缓冲区溢出。
- ✓ 使用适当的边界检查和错误处理机制，防止内存被非法访问。例如，在创建文件或目录之前，检查是否还有空余的磁盘块。

9.2 文件与目录的索引和查找

（1）技术难点

文件和目录的管理依赖于数组索引。在模拟磁盘的过程中，如何在大量目录项中高效地查找特定的文件或目录是一个技术挑战。特别是当磁盘空间有限或文件系统结构不够优化时，查找操作可能变得低效。

（2）解决方案

- ✓ 线性查找：对于当前实验，文件和目录的查找采用线性查找的方式。虽然这种方法在磁盘空间较大时效率较低，但在本实验的规模下，查找效率是可接受的。
- ✓ 优化索引结构：可以采用哈希表或树结构（如 B 树或红黑树）来优化文件和目录的查找速度，尤其是在文件系统规模增大时。这会提高查找文件的效率，但需要更复杂的内存管理和算法实现。
- ✓ 分配空闲块的策略：为提高文件系统的效率，可以引入一个专门管理空闲块

的结构，比如空闲块位图或链表，以便更快速地分配和回收磁盘块。

9.3 文件系统的路径管理和当前路径跟踪

（1）技术难点

文件系统中的路径管理是一个复杂的任务，特别是在进行路径跳转和相对路径/绝对路径解析时，容易出错。例如，如何正确解析并处理当前路径（current_path）以及如何在目录结构中执行路径操作（如切换目录、删除文件等）是一个常见的挑战。

（2）解决方案

- ✓ 路径分隔符统一性：确保路径的分隔符一致性（如使用斜杠 /），并正确处理不同操作系统中的路径规范。
- ✓ 相对路径与绝对路径的处理：在 changeDirectory 和 displayDirectory 等函数中，通过合适的字符串操作（如 strcat 和 strcmp）来解析和操作路径。在切换目录时，始终检查路径是否有效，避免出现路径越界或访问不存在的目录。
- ✓ 路径栈管理：在复杂的路径操作中，可以使用栈结构来管理当前路径的状态。每次进入新的目录时，将当前路径推入栈中，退出时弹出栈顶路径，保持路径的完整性和准确性。

10 实验心得体会

通过这次 FAT16 文件系统的实验，我深入理解了文件系统的基本构成与工作原理。实验让我意识到，在实现一个简化版的文件系统时，内存管理和数据结构的选择是至关重要的。在模拟磁盘的过程中，如何高效管理文件目录、实现文件创建、删除、重命名等基本操作，涉及到内存的动态分配和释放。这要求我们时刻注意资源的管理，防止出现内存泄漏或无效内存访问的情况。同时，路径管理和当前路径的跟踪也给我带来了很大的挑战。在实现过程中，我学会了如何处理相对路径和绝对路径，并且掌握了如何通过字符串操作来模拟文件系统的路径解析，这让我对文件系统的目录结构有了更深的理解。尽管实验的规模较小，但通

过这些操作，我体会到文件系统内部复杂的实现逻辑及其背后的原理。这次实验不仅提升了我对文件系统工作机制的理解，也锻炼了我在编码过程中解决问题的能力。