操作系统课程设计实验报告

实验题目：简单文件系统的实现

姓　　名：王翔宇

学　　号：22200215

专　　业：网络空间安全

班　　级：22270311

老师姓名：张祯

日　　期：2024　年　6　月　12　日

目　　录

(大家注意，目录是自动生成的，页码从正文部分开始，当同学们把正文写完后，只需要右击目录，选择更新域，目录会自动更新)

一 题目介绍

本实验通过具体编写文件存储空间的管理、文件物理结构、目录结构和文件操作的实现，加深对文件系统内部数据结构、功能以及实现过程的理解。

任务描述

在内存中开辟一个虚拟磁盘空间作为文件存储分区，在其上实现一个简单的基于多级目录的单用户单任务系统中的文件系统。在退出该文件系统的使用时，应将虚拟磁盘上的内容以一个文件的方式保存到磁盘上，以便下次可以再将它恢复到内存的虚拟磁盘中。

文件物理结构可采用显式链接或其他结构。

空闲磁盘空间的管理可选择 FAT 表、位示图或其他办法。

文件目录结构采用多级目录结构。为简单起见，可以不使用索引结点，每个目录项应包含文件名、物理地址、长度等信息，还可以通过目录项实现对文件的读写保护。

二 实验思路

一、实验目标

理解文件系统基本概念：了解文件系统的基本组成部分和工作原理。

实现基本功能：实现文件和目录的基本操作，如创建、删除、读取、写入等。

模拟磁盘管理：使用 FAT（文件分配表）来管理磁盘块。

文件系统持久化：实现文件系统的保存和加载功能，以便在程序重启后恢复文件系统状态。

二、实验步骤

环境准备：

选择 C 语言作为编程语言。

准备一个简单的命令行接口，用于输入和执行文件系统命令。

数据结构设计：

定义目录、文件和 FAT 表的结构。

模拟磁盘块的数组。

功能模块实现：

文件系统初始化：

初始化根目录。

初始化 FAT 表。

设置当前目录。

文件系统格式化：

重置文件系统，包括 FAT 表、根目录和磁盘块。

目录操作：
创建、删除、浏览和导航目录。
文件操作：
创建、删除、打开、关闭、读写文件。
文件系统持久化：
保存和加载文件系统状态。
三、具体实现步骤
数据结构定义：

使用结构体定义目录、文件和 FAT 表项。
使用二维数组模拟磁盘块。
文件系统初始化和格式化：

编写初始化函数，设置根目录和 FAT 表。
编写格式化函数，重置文件系统状态。
目录操作：

创建目录：在当前目录下添加新的子目录。
删除目录：删除指定的子目录。
浏览目录：列出当前目录下的所有文件和子目录。
导航目录：改变当前目录到指定子目录或父目录。

文件操作：

创建文件：在当前目录下创建新的文件，并分配磁盘块。
删除文件：删除指定文件并释放其占用的磁盘块。
打开文件：标记文件为打开状态。
关闭文件：标记文件为关闭状态。
写入文件：将数据写入文件的磁盘块中。
读取文件：从文件的磁盘块中读取数据。
文件系统持久化：

保存文件系统：将当前文件系统的状态写入磁盘文件中。
加载文件系统：从磁盘文件中读取文件系统的状态。
用户交互接口：

实现一个简单的命令行接口，用于输入和执行文件系统命令。
支持基本的文件系统操作命令，如 mkdir、rmdir、ls、cd、create、rm、write、read、format 和 exit 等。

三 遇到问题及解决方法

问题：定义的数据结构可能不够灵活或不足以支持文件系统的所有功能。
解决方法：仔细分析文件系统的需求，重新设计数据结构，确保其足够灵活和完备。
文件系统初始化问题：

问题：文件系统初始化时可能存在资源分配或状态设置不正确的问题。
解决方法：仔细检查初始化函数，确保正确地分配资源和设置初始状态。
文件系统格式化问题：

问题：格式化文件系统时可能会出现磁盘数据未正确清除或 FAT 表未正确重置的问题。
解决方法：在格式化函数中确保正确地清除磁盘数据并重置 FAT 表。

四 核心代码及实验结果展示

4.1 实验小组分工

全部

4.2 核心代码及实验结果

宏定义以及结构体：

```
#define MAX_NAME_LEN 100
#define MAX_FILE_SIZE 1024
#define MAX_FILES 100
#define MAX_DISK_BLOCKS 1000
#define SAVE_FILE "filesystem_.dat"

// FAT table entry
typedef struct FATEntry
{
        int nextBlock;
} FATEntry;

// File structure
typedef struct File
{
        char name[MAX_NAME_LEN];
        int startBlock;
        int length;
        int isOpen;
        struct File *next;
} File;

// Directory structure
typedef struct Directory
{
        char name[MAX_NAME_LEN];
        struct Directory *parent;
        struct Directory *subdirs;
        File *files;
        struct Directory *next;
} Directory;
```

FAT 表和虚拟磁盘的定义以及所用函数声明：

```
// FAT table
FATEntry FAT[MAX_DISK_BLOCKS];
char disk[MAX_DISK_BLOCKS][MAX_FILE_SIZE];
int freeBlockIndex = 0;

Directory *root;
Directory *currentDir;
char currentPath[MAX_NAME_LEN * MAX_FILES] = "~";

void initialize();
void my_format();
void my_mkdir(const char *dirname);
void my_rmdir(const char *dirname);
void my_ls();
void my_cd(const char *dirname);
void my_create(const char *filename);
void my_open(const char *filename);
void my_close(const char *filename);
void my_write(const char *filename, const char *content);
void my_read(const char *filename);
void my_rm(const char *filename);
void my_exitsys();
void save_filesystem();
void load_filesystem();
void update_path();
int is_block_free(int block);
void save(Directory *dir, FILE *file);
Directory *create(Directory *parent, FILE *file);
```

磁盘分配函数：

```
int allocate_block()
{
        for (int i = freeBlockIndex; i < MAX_DISK_BLOCKS; ++i)
                {
                        if (FAT[i].nextBlock == -1 && is_block_free(i))
                                {
                                        freeBlockIndex = i + 1;
                                        return i;
                                }
                }
        for (int i = 0; i < freeBlockIndex; ++i)
                {
                        if (FAT[i].nextBlock == -1 && is_block_free(i))
                                {
                                        freeBlockIndex = i + 1;
                                        return i;
                                }
                }
        return -1;
}

int is_block_free(int block)
{
        for (int i = 0; i < MAX_DISK_BLOCKS; ++i)
S               {
                        if (FAT[i].nextBlock == block)
                                {
                                        return 0;
                                }
                }
        return 1;
-- 插入 --
```

文件系统初始化，格式化函数，退出函数：

```
void initialize()
{
        for (int i = 0; i < MAX_DISK_BLOCKS; ++i)
                {
                        FAT[i].nextBlock = -1;
                }

        root = (Directory *)malloc(sizeof(Directory));
        strcpy(root->name, "/");
        root->parent = NULL;
        root->subdirs = NULL;
        root->files = NULL;
        root->next = NULL;
        currentDir = root;
        strcpy(currentPath, "~");
}

void my_format()
{
        free(root);
        initialize();
        printf("Filesystem formatted.\n");
}
```

```
void my_exitsys()
{
        save_filesystem();
        printf("文件系统已保存\n");
        exit(0);
}
```

目录相关的操作：

创建目录：

```
void my_mkdir(const char *dirname)
{
        Directory *newDir = (Directory *)malloc(sizeof(Directory));
        strcpy(newDir->name, dirname);
        newDir->parent = currentDir;
        newDir->subdirs = NULL;
        newDir->files = NULL;
        newDir->next = currentDir->subdirs;
        currentDir->subdirs = newDir;
}
```

删除目录：

```
void my_rmdir(const char *dirname)
{
        Directory *prev = NULL, *curr = currentDir->subdirs;
        while (curr != NULL && strcmp(curr->name, dirname) != 0)
                {
                        prev = curr;
                        curr = curr->next;
                }
        if (curr == NULL)
                {
                        printf("Directory %s does not exist.\n", dirname);
                        return;
                }
        if (prev == NULL)
                {
                        currentDir->subdirs = curr->next;
                }
        else
                {
                        prev->next = curr->next;
                }
        free(curr);
}
```

切换目录：

```
void my_cd(const char *dirname)
{
        if (strcmp(dirname, "..") == 0)
                {
                        if (currentDir->parent != NULL)
S                               {
                                        currentDir = currentDir->parent;
                                        update_path();
                                }
                        return;
                }
        Directory *subdir = currentDir->subdirs;
        while (subdir != NULL)
                {
                        if (strcmp(subdir->name, dirname) == 0)
                                {
                                        currentDir = subdir;
                                        update_path();
                                        return;
                                }
                        subdir = subdir->next;
                }
        printf("Directory %s does not exist.\n", dirname);
}
```

列出当前目录下文件和子目录：

```
void my_ls()
{
        Directory *subdir = currentDir->subdirs;
        while (subdir != NULL)
                {
                        printf("%s\n", subdir->name);
                        subdir = subdir->next;
                }
        File *file = currentDir->files;
        while (file != NULL)
                {
                        printf("%s\n", file->name);
                        file = file->next;
                }
}
```

更新当前目录路径：

```
void update_path()
{
        if (currentDir == root)
                {
                        strcpy(currentPath, "~");
                }
        else
                {
                        Directory *temp = currentDir;
                        char tempPath[MAX_NAME_LEN * MAX_FILES] = "";
                        while (temp != root)
                                {
                                        char tempName[MAX_NAME_LEN];
                                        strcpy(tempName, temp->name);
                                        strcat(tempName, "/");
                                        strcat(tempName, tempPath);
                                        strcpy(tempPath, tempName);
                                        temp = temp->parent;
                                }
                        strcpy(currentPath, "~/");
                        strcat(currentPath, tempPath);
                        currentPath[strlen(currentPath) - 1] = '\0';
                }
}
```

文件相关的操作：

创建文件：

```
void my_create(const char *filename)
{
        File *newFile = (File *)malloc(sizeof(File));
        strcpy(newFile->name, filename);
        newFile->startBlock = allocate_block();
        if (newFile->startBlock == -1)
                {
                        printf("No free blocks available.\n");
                        free(newFile);
                        return;
                }
        newFile->length = 0;
        newFile->isOpen = 0;
        newFile->next = currentDir->files;
        currentDir->files = newFile;
        printf("File %s created at block %d.\n", filename, newFile->startBlock);
}
```

打开和关闭文件：

```
void my_open(const char *filename)
{
        File *file = currentDir->files;
        while (file != NULL)
                {
                        if (strcmp(file->name, filename) == 0)
                                {
                                        file->isOpen = 1;
                                        printf("File %s opened.\n", filename);
                                        return;
                                }
                        file = file->next;
                }
        printf("File %s does not exist.\n", filename);
}

void my_close(const char *filename)
{
        File *file = currentDir->files;
        while (file != NULL)
                {
                        if (strcmp(file->name, filename) == 0)
                                {
                                        file->isOpen = 0;
                                        printf("File %s closed.\n", filename);
                                        return;
                                }
                        file = file->next;
                }
        printf("File %s does not exist.\n", filename);
}
```

写文件：

```
void my_write(const char *filename, const char *content)
{
        File *file = currentDir->files;
        while (file != NULL)
                {
                        if (strcmp(file->name, filename) == 0)
                                {
                                        if (!file->isOpen)
                                                {
                                                        printf("File %s is not open.\n", filename);
                                                        return;
                                                }
                                        int blockIndex = file->startBlock;
                                        int offset = 0;
                                        int contentLength = strlen(content);
                                        while (offset < contentLength)
                                                {
                                                        int remainingBytes = MAX_FILE_SIZE - (file->length % MAX_FILE_SIZE);
                                                        int writeLength = (contentLength - offset < remainingBytes) ? contentLength - offset : remainingBytes;
                                                        memcpy(disk[blockIndex] + (file->length % MAX_FILE_SIZE), content + offset, writeLength);
                                                        offset += writeLength;
                                                        file->length += writeLength;
                                                        if (offset < contentLength)
                                                                {
                                                                        if (FAT[blockIndex].nextBlock == -1)
                                                                                {
                                                                                        int newBlock = allocate_block();
                                                                                        if (newBlock == -1)
                                                                                                {
                                                                                                        printf("No free blocks available.\n");
                                                                                                        return;
                                                                                                        printf("No free blocks available.\n");
                                                                                                        return;
                                                                                                }
                                                                                        FAT[blockIndex].nextBlock = newBlock;
                                                                                        blockIndex = newBlock;
                                                                                        FAT[blockIndex].nextBlock = -1;
                                                                                }
                                                                        else
                                                                                {
                                                                                        blockIndex = FAT[blockIndex].nextBlock;
                                                                                }
                                                                }
                                                }

                                        printf("Written to file %s.\n", filename);
                                        return;
                                }
                        file = file->next;
                }
        printf("File %s does not exist.\n", filename);
}
```

读文件：

```
void my_read(const char *filename)
{
        File *file = currentDir->files;
        while (file != NULL)
                {
                        if (strcmp(file->name, filename) == 0)
                                {
                                        if (!file->isOpen)
                                                {
                                                        printf("File %s is not open.\n", filename);
                                                        return;
                                                }
                                        int blockIndex = file->startBlock;
                                        int offset = 0;
                                        while (blockIndex != -1 && offset < file->length)
                                                {
                                                        int readLength = (file->length - offset < MAX_FILE_SIZE) ? file->length - offset : MAX_FILE_SIZE;
                                                        fwrite(disk[blockIndex], 1, readLength, stdout);
                                                        offset += readLength;
                                                        blockIndex = FAT[blockIndex].nextBlock;
                                                }
                                        printf("\n");
                                        return;
                                }
                        file = file->next;
                }
        printf("File %s does not exist.\n", filename);
}
```

删除文件：

```
void my_rm(const char *filename)
{
        File *prev = NULL, *curr = currentDir->files;
        while (curr != NULL && strcmp(curr->name, filename) != 0)
                {
                        prev = curr;
                        curr = curr->next;
                }
        if (curr == NULL)
                {
                        printf("File %s does not exist.\n", filename);
                        return;
                }
        if (prev == NULL)
                {
                        currentDir->files = curr->next;
                }
        else
                {
                        prev->next = curr->next;
                }

        int blockIndex = curr->startBlock;
        while (blockIndex != -1)
                {
                        int nextBlock = FAT[blockIndex].nextBlock;
                        FAT[blockIndex].nextBlock = -1;
                        blockIndex = nextBlock;
                }

        free(curr);
```

文件系统的保存和装载函数

保存：

```
void save_filesystem()
{
        FILE *file = fopen(SAVE_FILE, "wb");
        if (file == NULL)
                {
                        printf("Error saving filesystem.\n");
                        return;
                }

        fwrite(&freeBlockIndex, sizeof(int), 1, file);
        fwrite(FAT, sizeof(FATEntry), MAX_DISK_BLOCKS, file);
        fwrite(disk, sizeof(char), MAX_DISK_BLOCKS * MAX_FILE_SIZE, file);//保存全局信息

        save(root, file);

        fclose(file);
        printf("Filesystem saved.\n");
}
```

```
//save() 函数递归地保存目录和文件信息，包括目录名、子目录数、文件名，文件数及文件的起始块和长度。
void save(Directory *dir, FILE *file)
{
        int subdirCount = 0, fileCount = 0;
        Directory *subdir = dir->subdirs;
        while (subdir)
                {
                        subdirCount++;
                        subdir = subdir->next;
                }

        File *currFile = dir->files;
        while (currFile)
                {
                        fileCount++;
                        currFile = currFile->next;
                }

        fwrite(dir->name, sizeof(char), MAX_NAME_LEN, file);
        fwrite(&subdirCount, sizeof(int), 1, file);
        fwrite(&fileCount, sizeof(int), 1, file);

        currFile = dir->files;
        while (currFile)
                {
                        fwrite(currFile->name, sizeof(char), MAX_NAME_LEN, file);
                        fwrite(&currFile->startBlock, sizeof(int), 1, file);
                        fwrite(&currFile->length, sizeof(int), 1, file);
                        currFile = currFile->next;
                }
```

```
                }

        subdir = dir->subdirs;
        while (subdir)
                {
                        save(subdir, file);
                        subdir = subdir->next;
                }
}
```

装载：

```
void load_filesystem()
{
        FILE *file = fopen(SAVE_FILE, "rb");
        if (file == NULL)
                {
                        initialize();
                        printf("没有现有系统，初始化一个新系统\n");
                        return;
                }

        fread(&freeBlockIndex, sizeof(int), 1, file);
        fread(FAT, sizeof(FATEntry), MAX_DISK_BLOCKS, file);
        fread(disk, sizeof(char), MAX_DISK_BLOCKS * MAX_FILE_SIZE, file);

        root = create(NULL, file);
        currentDir = root;
        strcpy(currentPath, "~");

        fclose(file);
        printf("Filesystem loaded.\n");
}
```

```
//create() 函数递归地从文件中加载目录和文件信息。
Directory *create(Directory *parent, FILE *file)
{
        Directory *dir = (Directory *)malloc(sizeof(Directory));
        fread(dir->name, sizeof(char), MAX_NAME_LEN, file);
        dir->parent = parent;

        int subdirCount, fileCount;
        fread(&subdirCount, sizeof(int), 1, file);
        fread(&fileCount, sizeof(int), 1, file);

        dir->subdirs = NULL;
        dir->files = NULL;

        for (int i = 0; i < fileCount; ++i)
                {
                        File *newFile = (File *)malloc(sizeof(File));
                        fread(newFile->name, sizeof(char), MAX_NAME_LEN, file);
                        fread(&newFile->startBlock, sizeof(int), 1, file);
                        fread(&newFile->length, sizeof(int), 1, file);
                        newFile->isOpen = 0;
                        newFile->next = dir->files;
                        dir->files = newFile;
                }

        for (int i = 0; i < subdirCount; ++i)
                {
                        Directory *subdir = create(dir, file);
                        subdir->next = dir->subdirs;
                        dir->subdirs = subdir;
                }
```

```
        for (int i = 0; i < subdirCount; ++i)
                {
                        Directory *subdir = create(dir, file);
                        subdir->next = dir->subdirs;
                        dir->subdirs = subdir;
                }

        return dir;
}
```

主函数：

```
int main()
{
        char command[256];
        char name[256];
        char content[MAX_FILE_SIZE];

        load_filesystem();

        while (1)
                {
                        printf("%s>", currentPath);
                        fgets(command, sizeof(command), stdin);
                        command[strcspn(command, "\n")] = 0;

                        if (strcmp(command, "exit") == 0)

                                {
                                        my_exitsys();
                                        break;
                                }
                        else if (strcmp(command, "format") == 0)

                                {
                                        my_format();
                                }
                        else if (strncmp(command, "mkdir ", 6) == 0)
                                {
                                        sscanf(command + 6, "%s", name);
                                        my_mkdir(name);
                                }
                        else if (strncmp(command, "rmdir ", 6) == 0)
```

```
                        else if (strncmp(command, "rmdir ", 6) == 0)
                                {
                                        sscanf(command + 6, "%s", name);
                                        my_rmdir(name);
                                }
                        else if (strcmp(command, "ls") == 0)
                                {
                                        my_ls();
                                }
                        else if (strncmp(command, "cd ", 3) == 0)
                                {
                                        sscanf(command + 3, "%s", name);
                                        my_cd(name);
                                }
                        else if (strncmp(command, "create ", 7) == 0)
                                {
                                        sscanf(command + 7, "%s", name);
                                        my_create(name);
                                }
                        else if (strncmp(command, "open ", 5) == 0)
                                {
                                        sscanf(command + 5, "%s", name);
                                        my_open(name);
                                }
                        else if (strncmp(command, "close ", 6) == 0)
                                {
                                        sscanf(command + 6, "%s", name);
                                        my_close(name);
                                }
```

```
                else if (strncmp(command, "write ", 6) == 0)
                        {
                                sscanf(command + 6, "%s", name);
                                printf("Enter content: ");
                                fgets(content, sizeof(content), stdin);
                                content[strcspn(content, "\n")] = 0;
                                my_write(name, content);
                        }
                else if (strncmp(command, "read ", 5) == 0)
                        {
                                sscanf(command + 5, "%s", name);
                                my_read(name);
                        }
                else if (strncmp(command, "rm ", 3) == 0)
                        {
                                sscanf(command + 3, "%s", name);
                                my_rm(name);
                        }
                else
                        {
                                printf("Unknown command.\n");
                        }
                }

        return 0;
}
```

实验结果：

```
Filesystem loaded.
~>format
Filesystem formatted.
~>ls
~>mkdir 1
~>mkdir 2
~>ls
2
1
~>create a.txt
File a.txt created at block 0.
~>open a.txt
File a.txt opened.
~>write a.txt
Enter content: hello world!
Written to file a.txt.
~>read a.txt
hello world!
~>close a.txt
File a.txt closed.
~>ls
2
1
a.txt
~>cd 1
~/1>cd ..
~>cd 2
~/2>cd ..
~>exit
Filesystem saved.
文件系统已保存
haha@haha-virtual-machine:~/桌面/five$
```

```
haha@haha-virtual-machine:~/桌面/five$ ./filesystem
Filesystem loaded.
~>ls
1
2
a.txt
~>
```

五 个人实验改进与总结

5.1 个人实验改进

再进行该实验时，采用 FAT 表来表示文件在磁盘快上存放的位置，定义了一个 FAT 结构体，结构体中的内容是文件下一个磁盘块的块号。此外，我采取另外一种保存方式，将文件和目录分开保存，整个文件系统在一个数据文件上进行，在保存文件系统的时候，将所有内容 FAT 表、磁盘数组以及递归对目录和文件进行保存。在读取文件系统的时候，也递归读取出保存的所有全局变量，目录和文件。

5.2 个人实验总结

本次实验成功实现了一个基本的虚拟文件系统，包括文件和目录的创建、删除、读取、写入、打开、关闭等基本功能。通过采用 FAT（文件分配表）结构管理磁盘块，并实现了文件系统的持久化存储，使得文件系统数据在程序重启后仍然有效。实验过程中，进一步优化了文件系统的存储和读取性能，解决了磁盘块分配和释放的问题，为未来的功能扩展打下了坚实基础。通过此次实验，我深入理解了文件系统的基本原理和实现方法，提升了编程能力和解决问题的能力。

六 参考文献

1. 文件系统的设计与实现（操作系统课程设计）_文件安全系统设计与实现-CSDN 博客
2. 操作系统课程设计----模拟文件管理系统（c 语言）_操作系统 | 文件系统实验 c 语言版-CSDN 博客