湖南大學
HUNAN UNIVERSITY

操作系统实验报告

课 程 名 称：操作系统
实验项目名称：操作系统内核编程实验
专 业 班 级：软件 2203
姓　　名：白旭
学　　号：202226010306
指 导 教 师：周军海
完 成 时 间：2024 年 5 月 29 日

信息科学与工程学院

实验题目：实验六 文件系统

实验目的：

- **使用文件系统注册/注销函数，注册一个文件系统类型，名称为"mrfs"或其他自定义名称；**
- **此文件系统至少需要拥有以下功能：**
  （1）ls:查看当前目录下的文件和文件夹信息命令。
  （2）cd:进入下级目录命令。
  （3）mv:移动文件命令
  （4）touch：新建文件命令
  （5）mkdir：新建文件夹命令
  （6）rm:删除文件命令
  （7）rmdir:删除文件夹命令
  （8）read：从某文件内读取信息命令
  （9）write：向某文件内写入信息命令
  （10）exit：退出文件系统命令
- **思考题：**
  虚拟文件系统（VFS）是什么、不是什么以及为什么是这样？

实验环境：

- Ubuntu 20.04

实验内容及操作步骤：

1. 编写代码

   1) file.h

```cpp
#ifndef FILE_H
#define FILE_H

#include <string>

/**
 * @class File
 * @brief 表示文件系统中的一个文件。
 * File 类封装了文件的属性和操作，包括文件名和内容。
 */
class File {
public:
    std::string name;    ///< 文件的名称。
    std::string content; ///< 文件的内容。

    /**
     * @brief 构造一个新的 File 对象。
     * 这个构造函数使用指定的名称初始化一个 File 对象。
     * 文件内容初始为空。
     * @param name 文件的名称。
     */
    File(std::string name);

    /**
     * @brief 销毁 File 对象。
     * 这个析构函数清理 File 对象使用的任何资源。
     */
    ~File();
};

#endif // FILE_H
```

#ifndef FILE_H 和 #define FILE_H 用于防止重复包含头文件

#include <string> 包含了 std::string 类型的定义

class File 定义了表示文件的类

public 访问说明符下定义了类的成员变量和方法

成员变量

std::string name：文件的名称

std::string content：文件的内容

成员函数

File(std::string name)：构造函数，用指定的文件名初始化 File 对象，内容初始化为空

~File()：析构函数，清理 File 对象使用的资源

2) file.cpp

```cpp
#include "File.h"

/**
 * @brief 使用指定的名称构造一个新的 File 对象。
 *
 * 文件内容初始为空。
 *
 * @param name 文件的名称。
 */
File::File(std::string name) : name(name) {
    // 构造函数实现
}

/**
 * @brief 销毁 File 对象。
 *
 * 这个析构函数清理 File 对象使用的任何资源。
 */
File::~File() {
    // 析构函数实现
}
```

#include "File.h"包含了 File 类的头文件，以实现文件类的功能

构造函数使用成员初始化列表(: name(name))初始化文件对象名称

析构函数没有具体实现，因为这个简单示例中并没有涉及到需要清理的资源

成员函数

File::File(std::string name)：构造函数，用指定的文件名初始化 File 对象，内容初始化为空

File::~File()：析构函数，清理 File 对象使用的资源。

3) disk.h

```cpp
#ifndef DISK_H
#define DISK_H

#include <vector>
#include <string>
#include <unordered_map>
#include "File.h"

/**
 * @class Directory
 * @brief 表示文件系统中的一个目录。
 *
 * Directory 类包含了目录的名称以及目录中包含的文件和子目录。
 */
class Directory {
public:
    std::string name;    ///< 目录的名称。
    std::unordered_map<std::string, File*> files; ///< 目录中的文件。
    std::unordered_map<std::string, Directory*> directories; ///< 目录中的子目录。

    Directory(std::string name);  ///< 构造一个新的 Directory 对象。
    ~Directory();  ///< 销毁 Directory 对象。
};

/**
 * @class Disk
 * @brief 表示一个简单的内存文件系统。
 *
 * Disk 类包含了文件系统的根目录和当前目录，并提供了基本的文件和目录操作。
 */
```

```cpp
class Disk {
public:
    Directory* root; ///< 文件系统的根目录。
    Directory* currentDirectory; ///< 当前工作目录。

    Disk();  ///< 构造一个新的 Disk 对象。
    ~Disk();  ///< 销毁 Disk 对象。

    void ls();  ///< 显示当前目录下的文件和子目录。
    void cd(std::string dirName);  ///< 切换到指定的目录。
    void mv(std::string oldName, std::string newDirPath);  ///< 移动文件或目录到指定路径。
    void touch(std::string fileName);  ///< 创建一个新的文件。
    void mkdir(std::string dirName);  ///< 创建一个新的目录。
    void rm(std::string fileName);  ///< 删除指定的文件。
    void rmdir(std::string dirName);  ///< 删除指定的目录。
    void read(std::string fileName);  ///< 读取指定文件的内容并显示。
    void write(std::string fileName, std::string content);  ///< 向指定文件写入内容。
    void exit();  ///< 退出文件系统。

private:
    void deleteDirectory(Directory* dir);  ///< 删除指定的目录及其内容。
    void deleteFile(File* file);  ///< 删除指定的文件。
    Directory* navigateToDirectory(const std::string& path);  ///< 导航到指定路径的目录。
};

#endif // DISK_H
```

Directory 类表示文件系统中的一个目录，包含目录名称、文件和子目录的映射

Disk 类表示一个简单的内存文件系统，包含根目录、当前工作目录以及基本的文件和目录操作函数；Disk 类包含了对文件和目录进行操作的函数，如显示当前目录内容、切换目录、移动文件、创建文件和目录、删除文件和目录等；Disk 类包含了一些私有函数，用于实现文件和目录的删除、导航等辅助功能

4) disk.cpp

```cpp
#include "Disk.h"
#include <iostream>

// Directory 类的构造函数，初始化目录名称
Directory::Directory(std::string name) : name(name) {}

// Directory 类的析构函数，清理目录中的文件和子目录
Directory::~Directory() {
    // 清理目录中的文件
    for (auto& file : files) {
        delete file.second;
    }
    // 清理目录中的子目录
    for (auto& dir : directories) {
        delete dir.second;
    }
}

// Disk 类的构造函数，创建文件系统的根目录和当前目录
Disk::Disk() {
    root = new Directory("/");  // 根目录
    currentDirectory = root;    // 当前目录初始为根目录
}

// Disk 类的析构函数，清理文件系统的根目录及其内容
Disk::~Disk() {
    deleteDirectory(root);  // 递归清理根目录及其内容
}
```

```cpp
// 显示当前目录下的文件和子目录
void Disk::ls() {
    // 显示子目录
    for (auto& dir : currentDirectory->directories) {
        if (dir.first != "..") {  // 排除父目录
            std::cout << dir.first << "/ ";
        }
    }
    // 显示文件
    for (auto& file : currentDirectory->files) {
        std::cout << file.first << " ";
    }
    std::cout << std::endl;
}
```

```cpp
// 切换到指定的目录
void Disk::cd(std::string dirName) {
    if (dirName == "..") {  // 返回上级目录
        if (currentDirectory != root) {
            currentDirectory = currentDirectory->directories[".."];
        }
    } else if (currentDirectory->directories.find(dirName) != currentDirectory->directories.end()) {
        currentDirectory = currentDirectory->directories[dirName];  // 进入子目录
    } else {
        std::cout << "Directory not found!" << std::endl;  // 目录不存在
    }
}
```

```cpp
// 移动文件或目录到指定路径
void Disk::mv(std::string oldName, std::string newDirPath) {
    // 导航到目标目录
    Directory* targetDir = navigateToDirectory(newDirPath);
    if (targetDir == nullptr) {
        std::cout << "Target directory not found!" << std::endl;
        return;
    }

    // 检查是否为文件
    if (currentDirectory->files.find(oldName) != currentDirectory->files.end()) {
        File* file = currentDirectory->files[oldName];
        currentDirectory->files.erase(oldName);
        targetDir->files[oldName] = file;  // 移动文件
    }
    // 检查是否为目录
    else if (currentDirectory->directories.find(oldName) != currentDirectory->directories.end()) {
        Directory* dir = currentDirectory->directories[oldName];
        currentDirectory->directories.erase(oldName);
        dir->directories[".."] = targetDir;  // 更新父目录
        targetDir->directories[oldName] = dir;  // 移动目录
    } else {
        std::cout << "File or Directory not found!" << std::endl;  // 文件或目录不存在
    }
}

// 创建一个新的文件
void Disk::touch(std::string fileName) {
    if (currentDirectory->files.find(fileName) == currentDirectory->files.end()) {
        currentDirectory->files[fileName] = new File(fileName);  // 创建新文件
    } else {
        std::cout << "File already exists!" << std::endl;  // 文件已存在
    }
}
```

```cpp
// 创建一个新的目录
void Disk::mkdir(std::string dirName) {
    if (currentDirectory->directories.find(dirName) == currentDirectory->directories.end()) {
        Directory* newDir = new Directory(dirName);  // 创建新目录
        newDir->directories[".."] = currentDirectory;  // 设置父目录
        currentDirectory->directories[dirName] = newDir;
    } else {
        std::cout << "Directory already exists!" << std::endl;  // 目录已存在
    }
}

// 删除指定的文件
void Disk::rm(std::string fileName) {
    if (currentDirectory->files.find(fileName) != currentDirectory->files.end()) {
        delete currentDirectory->files[fileName];  // 删除文件
        currentDirectory->files.erase(fileName);
    } else {
        std::cout << "File not found!" << std::endl;  // 文件不存在
    }
}
```

```cpp
// 删除指定的目录
void Disk::rmdir(std::string dirName) {
    if (currentDirectory->directories.find(dirName) != currentDirectory->directories.end()) {
        Directory* dir = currentDirectory->directories[dirName];
        currentDirectory->directories.erase(dirName);
        deleteDirectory(dir);  // 递归删除目录
    } else {
        std::cout << "Directory not found!" << std::endl;  // 目录不存在
    }
}

// 读取指定文件的内容并显示
void Disk::read(std::string fileName) {
    if (currentDirectory->files.find(fileName) != currentDirectory->files.end()) {
        std::cout << currentDirectory->files[fileName]->content << std::endl;  // 显示文件内容
    } else {
        std::cout << "File not found!" << std::endl;  // 文件不存在
    }
}
```

```cpp
// 向指定文件写入内容
void Disk::write(std::string fileName, std::string content) {
    if (currentDirectory->files.find(fileName) != currentDirectory->files.end()) {
        currentDirectory->files[fileName]->content = content;  // 写入文件内容
    } else {
        std::cout << "File not found!" << std::endl;  // 文件不存在
    }
}

// 退出文件系统
void Disk::exit() {
    std::cout << "Exiting file system." << std::endl;  // 退出文件系统
}

// 递归删除目录及其内容
void Disk::deleteDirectory(Directory* dir) {

    // 清理目录中的文件
    for (auto& file : dir->files) {
        delete file.second;
    }
    dir->files.clear();  // 清空文件列表

    // 递归清理子目录
    for (auto& subdir : dir->directories) {
        if (subdir.first != "..") {  // 排除父目录
            deleteDirectory(subdir.second);
        }
    }
    dir->directories.clear();  // 清空子目录列表

    delete dir;  // 清理目录对象
}
```

```cpp
// 导航到指定路径的目录
Directory* Disk::navigateToDirectory(const std::string& path) {
    Directory* dir = currentDirectory;
    size_t start = 0;
    size_t end = path.find('/');
    while (end != std::string::npos) {
        std::string dirName = path.substr(start, end - start);
        if (dir->directories.find(dirName) != dir->directories.end()) {
            dir = dir->directories[dirName];  // 导航到子目录
        } else {
            return nullptr;  // 路径不存在
        }
        start = end + 1;  // 更新起始位置
        end = path.find('/', start);  // 查找下一个斜杠
    }
    std::string dirName = path.substr(start);  // 获取路径中的目录名称
    if (dirName == "..") {
        return dir->directories[".."];  // 返回上级目录
    }
    if (dir->directories.find(dirName) != dir->directories.end()) {
        return dir->directories[dirName];  // 返回目标目录
    } else {
        return nullptr;  // 路径不存在
    }
}

// 删除指定的文件
void Disk::deleteFile(File* file) {
    delete file;  // 清理文件对象
}
```

这段代码实现了 Disk 类的各种文件和目录操作函数的定义，包括显示当前目录内容、切换目录、移动文件、创建文件和目录、删除文件和目录等。同时还实现了 Directory 类的构造函数、析构函数以及清理目录中的文件和子目录的功能

Disk 类的构造函数创建了文件系统的根目录和当前目录，并将当前目录初始设置为根目录

Disk 类的析构函数递归地清理文件系统的根目录及其内容

ls()函数用于显示当前目录下的文件和子目录

cd()函数用于切换到指定的目录

mv()函数用于移动文件或目录到指定路径

touch()函数用于创建一个新的文件

mkdir()函数用于创建一个新的目录

rm()函数用于删除指定的文件

rmdir()函数用于删除指定的目录

read()函数用于读取指定文件的内容并显示

write()函数用于向指定文件写入内容

exit()函数用于退出文件系统

5) my_shell.cpp

```cpp
#include <iostream>
#include <sstream>
#include "Disk.h"

// 解析用户输入的命令并调用相应的磁盘操作函数
void parseCommand(Disk &disk, const std::string &command) {
    std::istringstream iss(command);  // 使用字符串流解析命令
    std::string cmd, arg1, arg2;
    iss >> cmd >> arg1;  // 提取命令及参数
    std::getline(iss, arg2);  // 获取可能存在的第二个参数
    if (!arg2.empty() && arg2[0] == ' ') {
        arg2 = arg2.substr(1);  // 去除参数前的空格
    }

    // 根据命令调用对应的磁盘操作函数
    if (cmd == "ls") {
        disk.ls();
    } else if (cmd == "cd") {
        disk.cd(arg1);
    } else if (cmd == "mv") {
        disk.mv(arg1, arg2);
    } else if (cmd == "touch") {
        disk.touch(arg1);
    } else if (cmd == "mkdir") {
        disk.mkdir(arg1);
    } else if (cmd == "rm") {
        disk.rm(arg1);
    } else if (cmd == "rmdir") {
        disk.rmdir(arg1);
    } else if (cmd == "read") {
        disk.read(arg1);
    } else if (cmd == "write") {
        disk.write(arg1, arg2);
    } else if (cmd == "exit") {
        disk.exit();
    } else {
        std::cout << "Unknown command: " << cmd << std::endl;  // 未知命令
    }
}
```

```cpp
int main() {
    Disk disk;  // 创建磁盘对象
    std::string command;
    while (true) {
        std::cout << "> ";
        std::getline(std::cin, command);  // 获取用户输入的命令
        if (command == "exit") {  // 如果输入为 exit，则退出循环
            break;
        }
        parseCommand(disk, command);  // 解析并执行用户命令
    }
    return 0;
}
```

parseCommand()：解析用户输入的命令并调用相应的磁盘操作函数

main()：创建磁盘对象并进入一个循环，不断接收用户输入的命令并调用 parseCommand()函数执行

6) makefile

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall

all: my_shell

my_shell: my_shell.o Disk.o File.o
    $(CXX) $(CXXFLAGS) -o my_shell my_shell.o Disk.o File.o

my_shell.o: my_shell.cpp Disk.h File.h
    $(CXX) $(CXXFLAGS) -c my_shell.cpp

Disk.o: Disk.cpp Disk.h File.h
    $(CXX) $(CXXFLAGS) -c Disk.cpp

File.o: File.cpp File.h
    $(CXX) $(CXXFLAGS) -c File.cpp

clean:
    rm -f my_shell my_shell.o Disk.o File.o
```

CXX 定义了 C++ 编译器为 g++

CXXFLAGS 定义了编译选项，包括 C++11 标准和启用所有警告

all 是默认目标，依赖于 my_shell 目标

my_shell 目标依赖于 my_shell.o、Disk.o 和 File.o 这三个目标文件

clean 目标用于清理生成的可执行文件和中间目标文件

2. 模块功能验证

1) my_shell

实验结果及分析：

1. 实验结果与预期相符
2. 分析：

mkdir test：创建一个名为 test 的目录

touch file1：创建一个名为 file1 的空文件

ls：列出当前目录内容，显示 test/目录和 file1 文件

cd test：进入 test 目录

touch file2：在 test 目录中创建一个名为 file2 的空文件

ls：列出当前目录（test 目录）内容，显示 file2 文件

write file2 hello2：向 file2 文件写入内容 hello2

read file2：读取并显示 file2 文件内容，输出 hello2

rm file2：删除 file2 文件

ls：列出当前目录（test 目录）内容，此时目录为空

cd ..：返回上一级目录

mv file1 test：将 file1 文件移动到 test 目录

ls：列出当前目录内容，显示 test/目录

cd test：进入 test 目录
ls：列出当前目录（test 目录）内容，显示 file1 文件
cd ..：返回上一级目录
rmdir test：删除 test 目录（假设 test 目录已经空）
ls：列出当前目录内容，test 目录已经被删除，目录为空
exit：退出 my_shell 程序

收获与体会：

1. 对文件系统的基本原理有了更深入的了解
2. 熟悉了文件系统的基本操作

思考题：

1. 虚拟文件系统（VFS）是什么、不是什么以及为什么是这样？

   虚拟文件系统（VFS）是操作系统中的一个重要组件，它提供了一个统一的接口，使得不同类型的文件系统（如磁盘文件系统、网络文件系统等）能够以统一的方式被应用程序访问。VFS 抽象了底层文件系统的细节，为应用程序提供了一个通用的文件操作接口，使得应用程序可以独立于具体的文件系统实现。

   VFS 不是一个具体的文件系统，而是一个抽象层。它并不直接管理实际的文件存储，而是通过与底层文件系统交互来实现文件的读写等操作。因此，VFS 可以支持多种不同类型的文件系统，包括本地文件系统、网络文件系统、虚拟文件系统等，而不需要修改应用程序的代码。

   VFS 的设计使得操作系统和应用程序能够更加灵活和可扩展。它提供了一个统一的接口，使得应用程序不需要关心底层文件系统的细节，从而可以更容易地适应不同的文件系统实现。同时，VFS 的存在也提高了系统的安全性和稳定性，因为它可以在不影响应用程序的情况下替换底层文件系统的实现。

实验成绩	