课 程 设 计 报 告

设计题目：简单文件系统的设计与实现

【本报告为Github专版，抹去了部分个人信息】

班　　级：计算机2001班

组长学号：20205960

组长姓名：AlexHoring

指导教师：乔建忠

设计时间：2023 年 6 月

设计分工

组长学号及姓名：20205960-AlexHoring

分工：文件系统整体设计、虚拟磁盘实现、部分基本文件操作实现

组员 1 学号及姓名：20205925-HZ

分工：文件操作几乎所有基础与新增功能的具体实现

组员 2 学号及姓名：20205944-ZKY

分工：命令行 UI 交互，登录系统设计

摘　要

文件是具有文件名的一组关联信息的集合，通常文件由若干记录组成。文件系统是操作系统与管理文件相关的软件与数据的集合。本报告具体叙述了操作系统课程设计任务要求设计的文件系统的具体要求，以及我们实现的简单文件系统模拟程序的基本层次结构、应用的数据结构、采用的算法，最后展示了我们实现的文件系统运行结果。

本报告所叙述的文件系统实现了多用户多级目录，提供账户控制、目录管理、文件管理、文件读写等功能，并为权限控制功能留下了接口，可供日后拓展。该文件系统交互界面类似 Linux 命令行，上手容易，基本功能完善，足够普通用户正常使用。

通过课程设计，小组成员加深了对文件系统的理解，锻炼了代码、调试能力。

关键词：操作系统，文件系统，索引文件，树形目录，空闲块列表，分层结构

目　录

1 概述

本次课程设计选择的题目是“简单文件系统的设计与实现”，整个课程设计项目在 Windows 11 操作系统下使用 C++环境从零开始全新开发。项目使用的编译环境是 Mingw-w64 11.0 与 CMake 3.12，使用的集成开发环境是 CLion 2023.1 教育认证版，使用 Git 进行协作。

项目创建了一个 disk.zhl 文件作为磁盘的模拟，使用仿 Linux Terminal 风格的界面设计，用户通过输入命令与程序交互，实现文件系统的各种功能。我们设计的文件系统支持的命令与操作如表 1.1 所示。

表 1.1 文件系统支持的命令及其解释

命令	用法	含义	备注
login	无	用户登录	启动程序后自动执行
mkdir	mkdir <目录名>	创建目录	所有命令的目录名或者文件名均可选择绝对或相对路径
cd	cd <目录名>	切换目录	
ls	ls {<目录名>}	列举目录下所有项目	无参默认当前目录
touch	touch <文件名>	创建文件	
open	open <选项> <文件名>	打开文件	选项有 r（读）或 w（写）
write	write <文件名> <内容>	将指定内容写入文件	
read	read <文件名> <字符数>	从文件中读出字符	字符不足时读到文件结尾
close	close <文件名>	关闭文件	
rm	rm <文件名>	删除文件	
rmdir	rmdir <目录名>	删除目录	无法删除根目录
logout	logout{ <选项>}	退出程序	选项有 e（退出）与 s（切换用户），无参默认为 s
format	format	格式化	操作后工作目录回到根目录
rename	rename <旧名> <新名>	重命名文件或者目录	新名不能带“/”符号
mv	mv <源位置> <目的目录>	移动文件或者目录	
chmod	chmod <选项> <权限> <路径名>	设置指定文件或目录的访问权限	选项有 o（小组）u（其余用户）a（所有），目前无用
seek	seek <文件名> <选项> <偏移>	将指定文件的读写指针移动到指定位置	选项有 b（开头）c（当前位置），要求文件已打开

整个项目使用分层结构进行设计。项目共分为四层，最底层 DiskDriver 作为磁盘驱动，支持对虚拟磁盘（也就是 disk.zhl）文件进行操作，包括设置读写头，写入数据、读取数据等操作；上一层 FileSystem 是基础文件系统层，包括磁盘挂载、文件系统格式化、磁盘块分配回收、磁盘块读写、身份认证、权限

认证等功能；接着是 UserInterface 用户接口层，包括表 1.1 列举的所有命令的具体实现；最后顶层 Shell 是与用户的交互界面，负责将用户输入的命令转化为 UserInterface 层所需的参数形式，进行功能的调用。同时，本项目支持数据持久化，在退出登录、重启电脑甚至更换虚拟磁盘文件 disk.zhl 后，都可以正常读取并操作虚拟磁盘中的内容。

本项目实现了课程设计要求中列举的全部基础功能，并且增加了若干文件系统可以有的额外功能。

2 课程设计任务及要求

2.1 设计任务

利用课堂所学知识，设计并实现一个多用户、多级目录结构文件系统，可以参考 DOS 的 FAT 文件系统或者 UNIX 的文件系统。

2.2 设计要求

文件系统的基本功能包括：多用户、多级目录、具有 login （用户登录）、系统初始化（建文件卷、提供登录模块）、文件的创建、文件的打开、文件的读、文件的写、文件关闭、删除文件、创建目录、改变目录、列出文件目录、退出、格式化。

以上是基本内容，可以根据实际文件系统提供的命令和系统调用，自己增加和实现附加的功能。

3 算法及数据结构

3.1 算法总体思想与流程

我们团队设计的文件系统模拟程序采用了分层的思想，其分为四个层次，最底层是磁盘驱动层，主要对磁盘的寻道、写入、读取操作进行模拟；接着是

基本文件系统层，主要支持文件系统的建立、空闲块的分配与回收、用户信息的访问与用户权限的设置等；然后是用户接口层，这层主要实现了表 1.1 中的所有命令；最上层是用户界面层，该层直接面向用户，其主要职责是将用户输入的命令“翻译”成用户接口层可以识别的形式，然后调用下层的对应接口，并处理返回结果，呈现给用户。每一层都只能直接调用其下层提供的服务，并且向上层提供操作的接口。具体的层次结构与数据交互如图 3.1 所示。

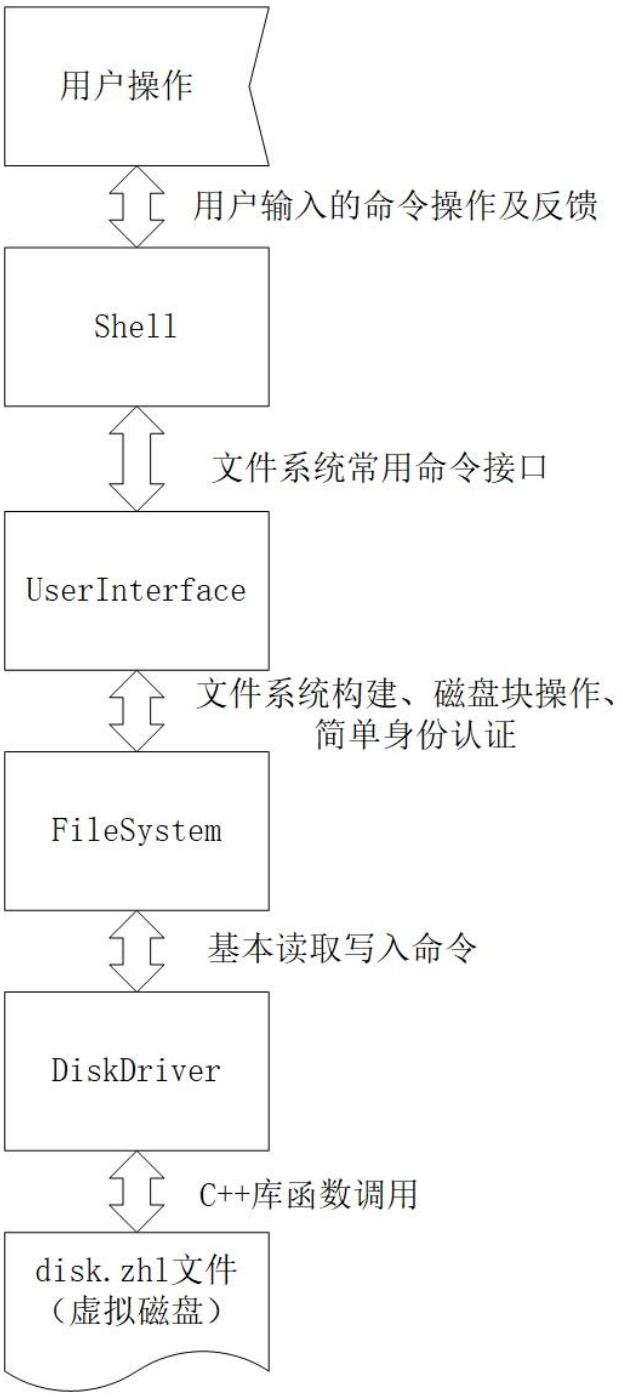

图 3.1 整个文件系统模拟程序的层次结构图

3.2 磁盘驱动模块

3.2.1 功能

磁盘驱动模块的功能是模拟计算机系统对磁盘的操作。根据所学知识，无论是机械硬盘，还是近年来兴起的固态硬盘，其共同的特点都是：可以定位到某一个位置，并从该位置开始读出或者写入数据。因此，作为对文件系统的模拟，我们从硬盘的操作中抽象出“定位”、“读取数据”、“写入数据”操作，在程序的运行环境中新建虚拟磁盘文件 disk.zhl，利用 C++语言的文件流 fstream 中相关函数来模拟上述三种操作。

3.2.2 数据结构

磁盘驱动程序的数据结构基础定义及其详细解释如表 3.1 所示。

表 3.1 磁盘驱动程序类 DiskDriver 的数据结构定义

成员变量	
变量名	**说明**
diskName	静态变量，存储虚拟磁盘文件名，方便拓展
disk	虚拟磁盘对象，使用 C++ fstream 类打开文件来模拟
isOpen	判断磁盘是否打开
成员函数	
函数名	**说明**
open()	打开虚拟磁盘文件，返回值为是否打开成功
close()	关闭虚拟磁盘文件，返回值为是否关闭成功
init(sz)	创建未格式化的容量为 sz 的虚拟磁盘文件，单位为 Byte
seekStart(sz)	将读写头移动到距起始 sz 字节处
seekCurrent(sz)	将读写头移动到距当前位置 sz 字节处
read(buf,sz)	从当前位置读出 sz 字节到 buf 缓冲区
write(buf,sz)	从当前位置将 buf 缓冲区中的 sz 字节写入文件

为了模拟的效果，我们打开文件的方式为二进制读写方式，所有数据以二

进制方式写入磁盘文件。这样虽然降低了文件的可读性，但是做到了对磁盘的深度还原。例如，当创建一个大小为 16MB 的虚拟磁盘时，生成的虚拟磁盘文件则不偏不倚恰好是 16MB。这也为上层文件系统模块的部分操作带来了方便。

3.2.3 算法

磁盘驱动模块大部分只是单纯调用 C++库函数，没有什么特别的算法可言。值得一提的是 init 函数，在 init 函数新创建磁盘时，系统将整个虚拟磁盘文件用二进制 0 来填充，直到磁盘大小达到要求的大小位置。然后，在文件的开头用二进制方式写入磁盘的大小，该数据占 4 个字节。这样做是考虑到“磁盘有一个基本引导区，其中存放着基本的信息”这一事实，而进行的模拟。

3.3 基本文件系统模块

3.3.1 功能

基本文件系统模块的功能是在磁盘上创建文件系统，并实现文件系统相关的操作。参考实际的文件系统实现，我们的文件系统将磁盘划分为“引导与超级块区”、“空闲块列表区”与“数据区”三大分区，具体将在 3.3.2 节予以介绍。该模块实现的功能包括格式化磁盘（在磁盘上建立文件系统）、磁盘块分配回收、磁盘块内容读写、用户认证与用户组策略设置等。本模块的所有功能都是在磁盘驱动模块提供的全部功能基础上实现的，体现了软件开发中常见的分层思想。

3.3.2 数据结构

在介绍文件系统模块的数据结构前，有必要介绍一下我们设计的文件系统。我们设计的文件系统参考了 Windows 操作系统下常见的 FAT 文件系统与 Linux 中常用的 ext 文件系统，在此基础上做了简化与修改。

文件系统的 0 号块是“引导与超级块区”，存放磁盘的最大容量、格式化标记、块大小、根目录所在磁盘块号、空闲块数、空闲块栈顶指针、空闲块栈指

针的块内偏移、磁盘可用容量、用户列表、信赖矩阵等信息。接下来的连续若干块是“空闲块列表区”，其连续存放空闲块的块号。剩下的区域是“数据区”，存放各种类型的文件。在我们的设计中，目录也被视作一种文件。

基本文件系统如图 3.2 所示。

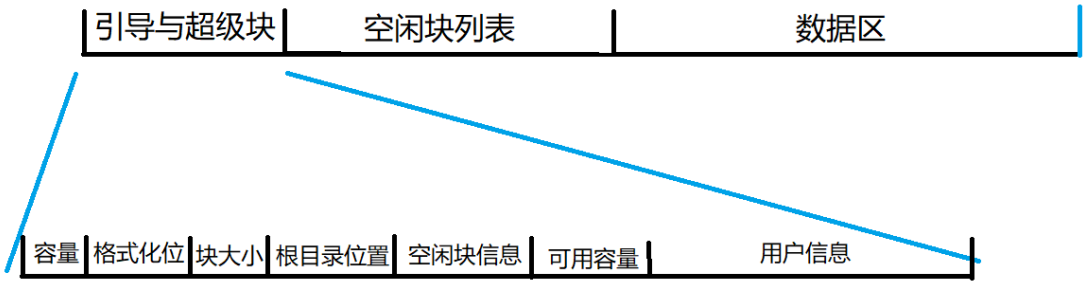

图 3.2 基本文件系统图示

每一个文件都有一个 i 节点，i 节点存放文件体所在的磁盘块、信息标记、文件创建者、文件大小、文件创建时间、文件上次修改时间等信息。其中，信息标记用一个 8 位无符号整数来表示，其中高 2 位 00 表示文件，01 表示目录，10 被定义为软链接（但是截至目前未实现），11 未定义，中间 3 位以 rwx 格式表示信赖者的访问权限，低 3 位表示其余用户访问权限。关于访问权限，在超级块中存在两个结构：一个是用户列表，一个是用户的信赖矩阵。在超级块的信赖矩阵 trustMatrix 中，对于两个不同的用户 a 和 b，trustMatrix[a][b]代表在 a 的视角下 b 的信赖状态，若值为 1 代表对 a 而言 b 可信任，为 0 则代表对 a 而言 b 仅仅是普通用户，该功能将被用于访问控制，但是限于时间，上层模块未实现相关功能。

对于一个目录而言，目录占用的空间为一个磁盘块。一个目录分为若干目录项，一个目录项占 16 字节，其中 4 字节表示该目录项 i 节点所在的磁盘块，12 字节表示该目录项指向文件的文件名。因此，本文件系统中文件名不能超过 11 个字符（C++字符串以空字符结尾），且文件与目录不可重名。由此可见，我们的目录采用目前常见的树形目录结构。

对于普通文件而言，我们设计的文件系统对于文件采用索引存储，i 节点指向的文件体是文件的索引表，索引表每一项指向一个磁盘块，一个索引表占据一个磁盘块。索引表的最后一项是下一索引表所在磁盘块，因此文件的大小理

论上不受限制。并且，由此可见，我们的文件采用单级索引结构。

与文件大小可以几乎无限制，相反，一个目录所包含的目录项有限，在块大小为 4096Byte 的情况下，一个目录仅能存放 254 个不同的文件，余下的两个目录项分别存储当前目录“./”与上级目录“../”。这是我们的文件系统设计初期的一个失误。

文件与目录结构的简要介绍如图 3.3 所示。

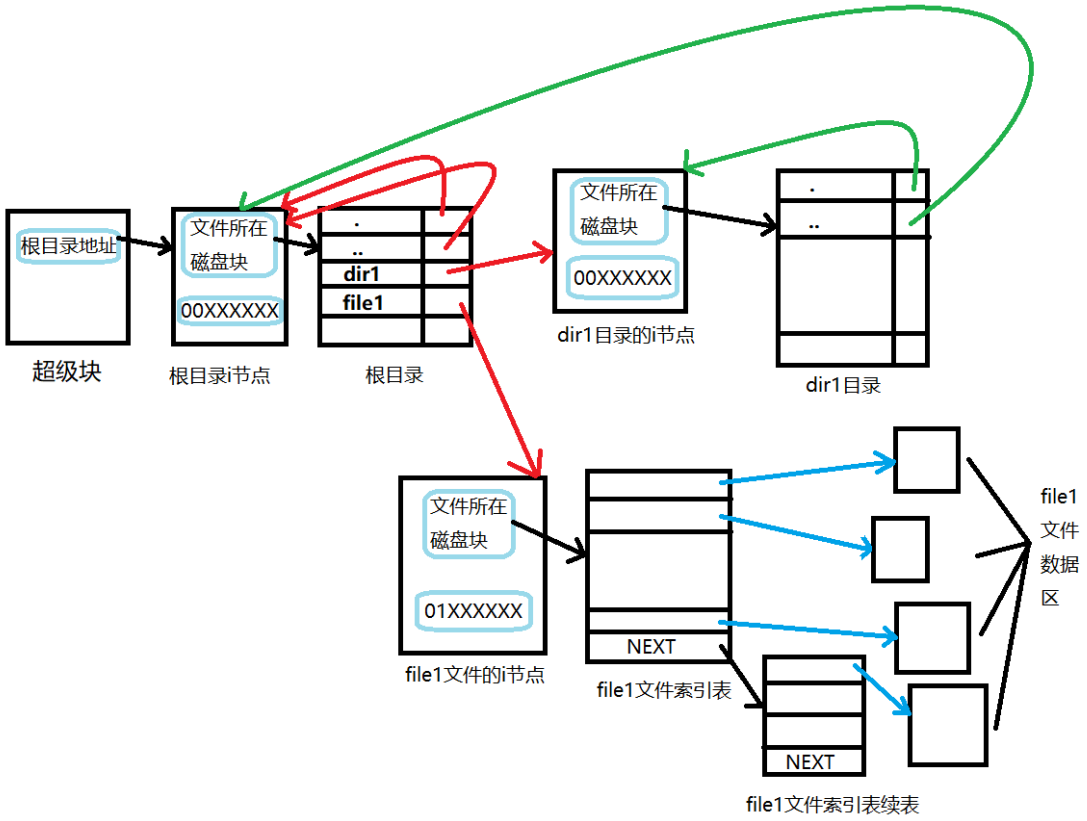

图 3.3 文件与目录的结构

基本文件系统的数据结构基础定义及其详细解释如表 3.2 所示。

表 3.2 文件系统类 FileSystem 的数据结构定义

成员变量	
变量名	**说明**
capacity	读取到的磁盘最大容量
isUnformatted	磁盘是否未格式化标记
isOpen	磁盘是否打开标记
blockSize	块大小
systemInfo	文件系统超级块信息

成员函数	
函数名	**说明**
createDisk(sz)	创建一个大小为 sz 的磁盘，单位 Byte
format(bsize)	将块大小设定为 bsize 进行格式化
mount()	尝试挂载硬盘，返回是否挂载成功
blockAllocate()	分配空闲磁盘块
blockFree(bno)	回收磁盘块
read(bno,offset,buf,sz)	从 bno 号磁盘块偏移 offset 出读入 sz 字节到缓冲区 buf
write(bno,offset,buf,sz)	将缓冲区 buf 中 sz 字节数据写入 bno 号磁盘块偏移 offset 处
readNext(buf,sz)	从当前位置读 sz 字节到 buf 缓冲区
writeNext(buf,sz)	从当前位置写入来自 buf 的 sz 字节数据
locale(bno,offset)	将读写头定位到 bno 磁盘块偏移 offset 处
userVerify(uname,pwd)	利用用户名 uname 与密码 pwd 认证身份
grantTrustuser(cusr,tusr)	为用户 cusr 添加信任用户 tusr
revokeTrustUser(cusr,tusr)	回收 cusr 对 tusr 的信任权限
verifyTrustUser(cuid,tuid)	测试对 cuid 而言 tuid 是否可信任
getUser(uid)	根据 uid 读取用户信息
getRootLocation()	获取根目录所在磁盘块
update()	更新超级块信息

3.3.3 算法

（1）读写磁盘块相关算法

文件系统层给上层模块提供基于磁盘块与偏移量的数据读写方式。为了将其转化为磁盘驱动层能够识别的基于偏移量的读写方式，只需要利用下列转换公式：

$$磁盘块号 \times 磁盘块大小 + 偏移量 = 实际偏移量$$

就可以转化为基于虚拟磁盘文件起始位置的定位与读写。

（2）磁盘块分配与回收算法

前面提到，磁盘块的分配与回收基于空闲块列表这一数据结构，该方案来自参考文献[4]。简单来说，就是将空闲磁盘块号按照从小到大顺序写入一个特定的连续区域，并且记录第一个可用的磁盘块号所在的位置。抽象来看，就是一个栈结构。对于块大小为 4096Byte 的磁盘块、块号用 4Byte 来表示的文件系统来说，空闲块列表所占用的空间不超过总容量的 0.1%，因此对于空间利用率来说，这是一个非常令人满意的方案。

在磁盘块分配与回收过程中，若是将整个空闲块列表调入内存，所需要的空间是非常大的。因此，我们的方案是：仅将部分空闲块调入内存。具体来说，将栈顶所在的磁盘块调入内存，因此在内存中只需要开辟一个磁盘块大小的空间。当在内存的部分已满或已空时，就与磁盘进行一轮数据交换，写入或读出一个磁盘块的数据。这个方案的好处是均摊下来能够实现几乎“零时间”的磁盘分配，并且保证尽量少的磁盘空间占用。该方案的缺点是对于栈底的块，可能很长时间不会被调用，靠前的磁盘块因为频繁的磁盘分配与回收可能导致写入量过大，造成读写分配不均匀，而且，在磁盘块边缘频繁读写可能造成多次磁盘访问，造成局部时间内分配效率下降。

磁盘块分配回收算法流程图如图 3.4 所示。

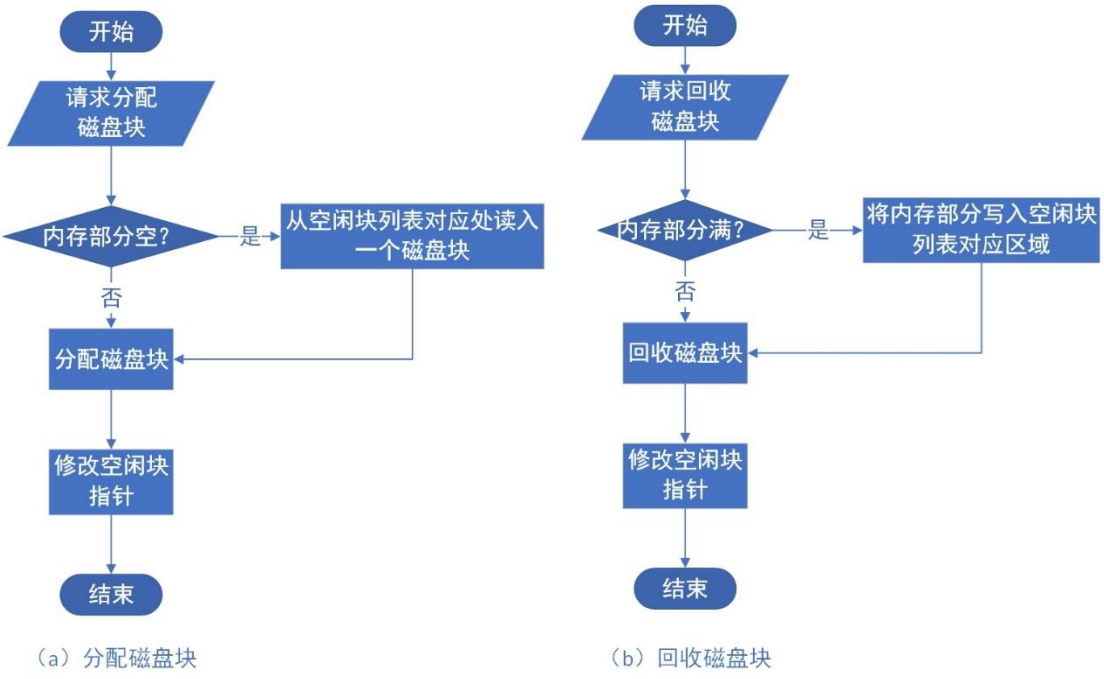

图 3.4 磁盘块分配回收算法流程图

（3）用户认证算法

在超级块区存在一个用户列表以及一个可信矩阵。根据课程设计基本要求，文件系统实现了一个简单的 8 用户管理。用户 ID、默认用户名与默认密码在格式化时由系统分配，默认所有人的可信矩阵项均为 0，即均为普通用户。与用户认证相关的算法只需在用户列表中顺序查找、指定行与列在可信矩阵中读取与写入就可以了。

（4）挂载与格式化算法

在虚拟磁盘文件起始处 4 字节为容量标记，接着 1 字节为格式化标记，该标记全 1 表示未格式化，全 0 表示已格式化。挂载时，先读入这两个数据进行判断，若已格式化，接着读入超级块、初始化空闲块列表的内存区即可。

挂载算法流程图如图 3.5 所示。

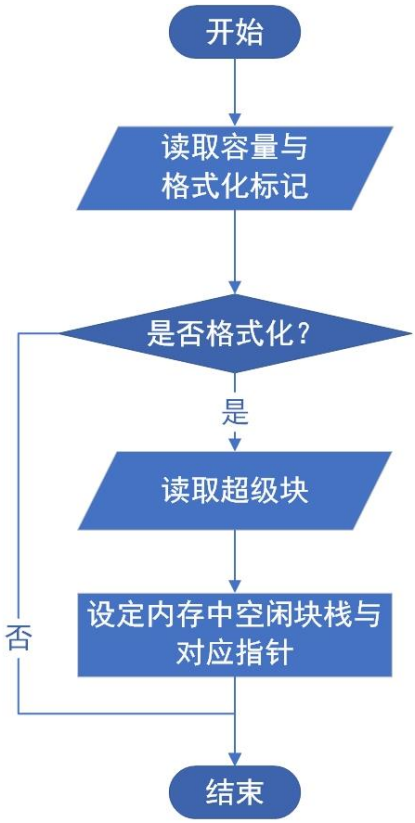

图 3.5 挂载算法流程图

当挂载且目标磁盘未被格式化后，就要进行格式化操作。格式化操作首先写入格式化标记与块大小，接着计算磁盘总块数，借此得到空闲块列表的大小。根目录 i 节点被分配在数据区的第一块磁盘块，也就是空闲块列表栈底后第一个磁盘块。根目录本体被初始化到 i 节点的下一块磁盘块。接着，将空闲

块号写入空闲块列表，并定位空闲块栈顶指针。最后，将用户名、密码、用户ID 等信息写入超级块中，格式化工作就结束了。

格式化算法的流程图如图 3.6 所示。

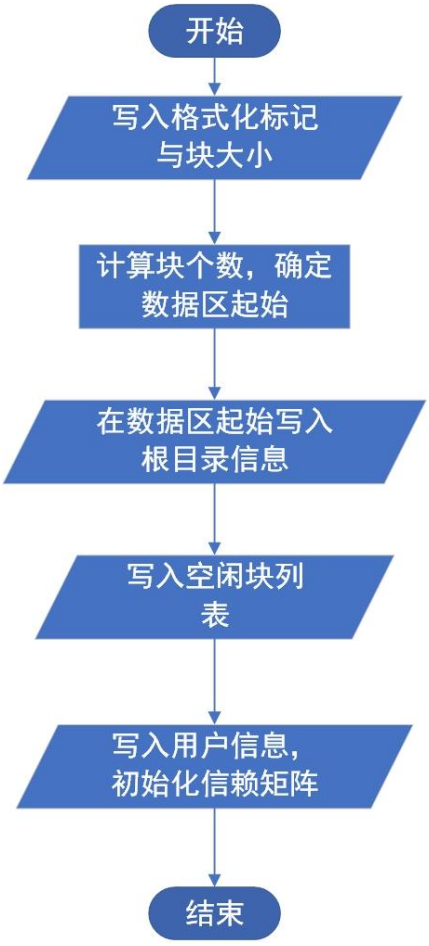

图 3.6 格式化算法流程图

（5）其他算法

对于表 3.2 提到的其他函数的实现算法，大部分还是以简单的调用下层接口与返回指定值为主。特别地，对于 update 函数，其作用是将超级块信息写入磁盘对应区域，并清除超级块修改标记。由于写入文件、分配回收磁盘块等操作会修改超级块中的相关信息，而频繁写入磁盘不是一个好主意，因此设计了 update 函数，上级模块可以设置在进行 10 次或者其他次数以后执行一次 update 函数，将数据持久化，以避免频繁的磁盘写入操作。

3.4 用户接口模块

3.4.1 功能

用户接口模块主要实现文件管理系统的所有业务逻辑。

这里的业务逻辑包括：文件系统的初始化；mkdir 命令——在指定路径下创建文件夹；ls 命令——显示指定路径下的所有文件和文件夹；touch 命令——在指定路径下创建文件；cd 命令——进入指定路径下的文件夹；rm 命令——删除指定路径下的文件；rmdir 命令——删除指定路径下的文件夹；mv 命令——移动指定路径下的文件或者文件夹；raname 命令——将指定路径下的文件或者文件夹重命名；format 命令——格式化整个文件系统；chmod——设置指定路径下的文件的读写权限和用户组权限；open 命令——将指定路径下的文件用指定的方式（读或写）打开；close 命令——关闭指定路径下的文件；setCursor 命令——将当前文件的文件指针设置偏移。

3.4.2 数据结构

用户接口模块的数据结构基础定义及其详细解释如表 3.3 所示。

表 3.3 用户接口模块类 UserInterface 的数据结构定义

成员变量	
变量名	**说明**
directory	文件系统当前所处目录
nowDiretoryDisk	文件系统当前所处目录所在的磁盘号
fileSystem	下层系统接口
fileOpenTable	文件打开表
成员函数	
函数名	**说明**
initialize()	文件系统初始化
mkdir(src, directoryName)	在 src 指出的目录下创建名为 directoryName 的文件夹
ls(src)	显示 src 指出的目录下的所有文件和目录

touch(src, fileName)	在 src 指出的目录下创建名为 fileName 的文件
cd(directoryName);	进入当前目录下名为 directoryName 的文件夹
rm(uid, src, fileName)	以 uid 的用户身份删除 src 指出的目录下的名为 fileName 的文件
rmdir(uid, src, dirName)	以 uid 的用户身份删除 src 指出的目录下的名为 dirName 的文件夹
mv(src, des)	将 src 路径指出的文件或者文件夹移动到 des 路径指出的文件或者文件夹
rename(src, newName)	将 src 路径指出的文件或者文件夹重命名为 newName
format()	将整个文件系统格式化
chmod(who, how, src)	将 src 路径指出的文件对 who 用户设置 how 权限
open(how, src)	将 src 路径指出的文件以 how 方式打开并设置文件打开表
close(src)	将 src 路径指出的文件关闭并设置文件打开表
read(uid, src, buf,sz)	以 uid 的用户的身份将 src 路径指出的文件读取 sz 字节到 buf 数组中
write(uid, src, buf,sz)	以 uid 的用户的身份 buf 数组中的数据写入 sz 字节到 src 路径指出的文件

关于文件的打开与关闭操作，需要做出一些补充：在用户接口模块中有一个名为“文件打开表”的结构，里面存储了打开文件的文件名、打开方式（读、写、读写）、文件 i 节点所在磁盘块等信息。当打开文件时，文件会被加入文件打开表，关闭文件时文件信息会被移除。对于文件的 read 与 write 操作只能针对打开的文件，若操作指出的文件没有被打开，则会向控制台输出一个错误命令。

3.4.3 算法

1. **Initialize 命令文件系统初始化函数**

   算法流程：首先尝试挂载，如果挂载失败则格式化否则继续；如果格式化失败则创建新磁盘否则继续；如果挂载成功则读入根节点所在磁盘块并将将根目录信息写入当前目录。

2. mkdir 命令创建文件夹

算法主要流程是找到指定路径，并在该路径下创建新目录，分配新 i 节点，并写入磁盘。算法流程图如图 3.7 所示。

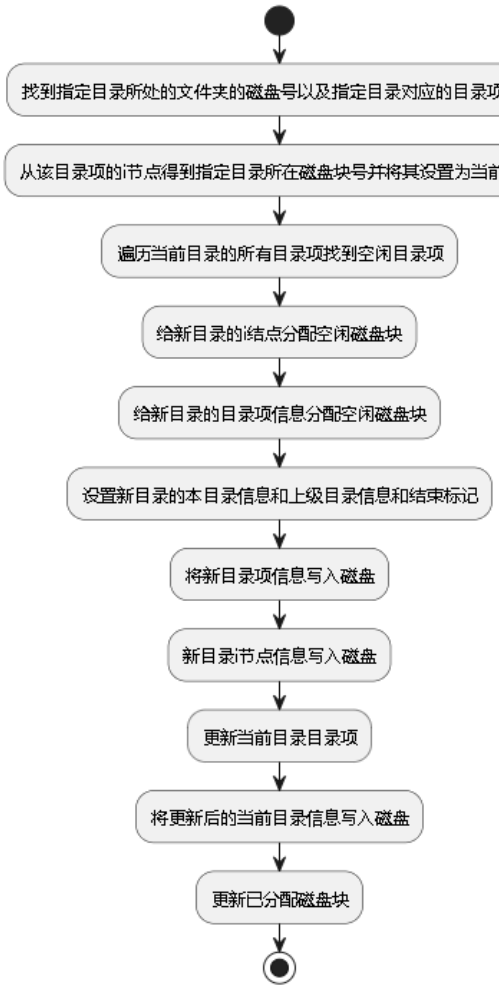

图 3.7 mkdir 命令流程图

3. ls 命令显示指定路径下的所有文件和文件夹

算法主要流程是找到指定路径，并打印该路径下的所有文件名和目录名。算法流程图如图 3.8 所示。

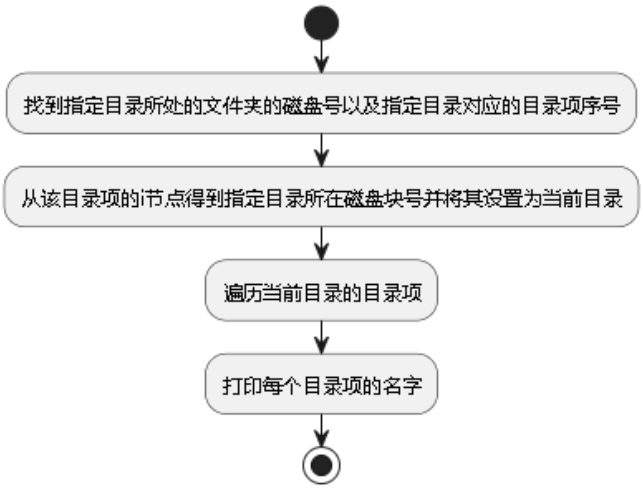

图 3.8 ls 命令流程图

4. touch 命令在指定路径下创建文件

算法主要流程是找到指定路径，并在该路径下创建新文件，分配新 i 节点，并写入磁盘。算法流程图如图 3.9 所示。

图 3.9 touch 命令流程图

5. cd 命令进入当前目录下的指定目录

算法主要流程是找到指定目录，并将指定目录设置为当前目录。算法流程图如图 3.10 所示。

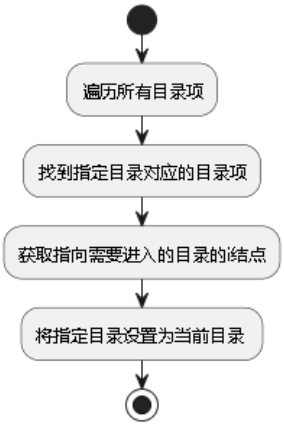

图 3.10 cd 命令流程图

6. rm 命令删除指定路径下的文件

算法主要流程是找到指定目录，并在指定目录下寻找指定目录项，并根据该目录项找到文件索引表，删除所有文件。算法流程图如图 3.11 所示。

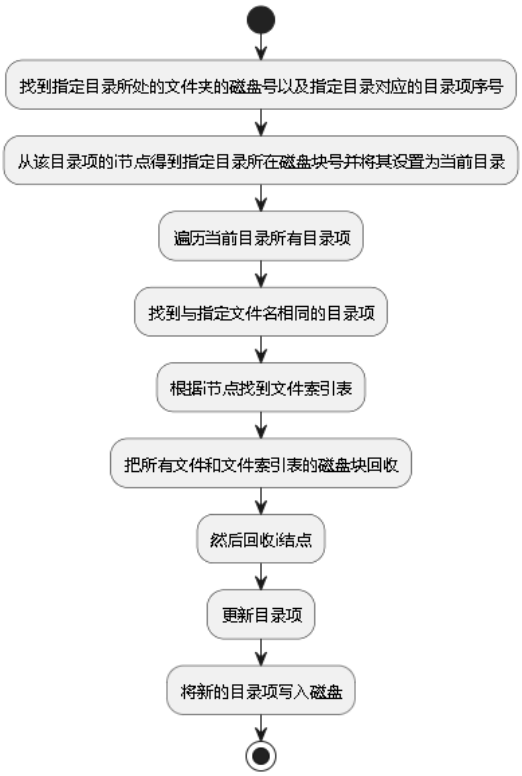

图 3.11 rm 命令流程图

7. **rmdir 命令删除指定路径下的文件夹**

算法流程：首先找到指定目录所处的文件夹的磁盘号以及指定目录对应的目录项序号，然后从该目录项的 i 节点得到指定目录所在磁盘块号并将其设置为当前目录，遍历当前目录所有目录项，找到与指定目录名相同的目录项，保存当前目录所在磁盘块，进入指定目录，一个目录的目录项如果只有前两项（当前目录与上级目录），那么该层目录递归结束，回收本层目录所有磁盘块；如果还有其他项，则遍历所有目录项，如果是文件,就使用 rm 接口删除文件;如果是目录,就递归调用 rmdir：保存当前目录，进入下一目录，回收该 i 结点，递归删除目录，递归返回时要重置当前目录。所有目录项回收完毕，回收指定目录的 i 结点所在磁盘块，重置当前目录，更新目录项，将新的目录项写入磁盘。

8. **mv 命令移动指定路径下的文件或者目录**

算法流程：查找源文件或者目录的 i 结点，查找目的目录所在的目录所在的磁盘块号以及对应目录项编号，在目标目录查找空目录项，保存当前目录,设置当前目录为被移动的文件所在的目录，更新被移动的文件所在的目录，并将被移动的文件所在目录项的新信息写入磁盘，在查找到的目标目录的空目录项写入源文件或者目录的 i 节点，将目标目录的新信息写入磁盘，更新磁盘信息。

9. **rename 命令将指定路径下的文件或目录重命名**

算法流程：找到需要被改名的文件或者目录所在的目录,和其对应的目录项编号，将对应目录项的名字改为新名字，将新目录信息写入磁盘。

10. **chmod 命令设置指定路径下的文件的对应用户的读写权限**

算法流程：找到需要被设置的文件所在的目录,和其对应的目录项编号，从对应目录项取出 i 节点，设置 i 节点的 flag 属性（高 2 位 00 表示文件，01 表示目录，10 表示软链接，中间 3 位以 rwx 格式表示信赖者的访问权限，低 3 位表示其余用户访问权限），然后写回 i 节点。

3.5 用户界面模块

3.5.1 功能

用户界面模块主要实现了文件系统与用户的直接交互。该模块接收用户输入并且对用户输入的命令进行解析，分解得到可以让用户接口模块直接识别的参数，用于完成用户指令。同时对文件系统的界面进行了规范和美化，用户在输入命令时可以看到当前登录的用户的用户名以及当前文件系统所处的文件夹以及从根目录到当前文件夹的整个路径。

3.5.2 数据结构

用户界面模块的数据结构基础定义及其详细解释如表 3.4 所示。

表 3.4 用户界面模块类 Shell 的数据结构定义

成员变量	
变量名	**说明**
cmd	接收用户输入的整行命令
user	当前登录用户名
isExit	是否退出标记
nowPath	当前从根目录开始的路径
成员函数	
函数名	**说明**
split_path()	分割用户输入的整行命令，得到下层可以解析的参数
running_shell()	界面主程序
cmd_cd()	cd 命令处理程序
cmd_ls()	ls 命令处理程序
cmd_mkdir()	mkdir 命令处理程序
cmd_touch()	touch 命令处理程序
cmd_rmdir()	rmdir 命令处理程序
cmd_rm()	rm 命令处理程序

cmd_mv()	mv 命令处理程序
cmd_rename()	rename 命令处理程序
cmd_format()	format 命令处理程序
cmd_chmod()	chmod 命令处理程序
cmd_open()	open 命令处理程序
cmd_close()	close 命令处理程序
cmd_login()	登录处理程序
cmd_logout()	logout 命令处理程序
cmd_read()	read 命令处理程序
cmd_write()	write 命令处理程序
cmd_seek()	seek 命令处理程序

3.5.3 算法

1. **split_path(string &path) 解析分割用户输入的整行命令**

   算法流程：依次分析用户输入的整行命令中的每个字符，如果遇到连续的'/'视为一个分割符，直到遇到非'/'，从该字符开始到下一个'/'之间的所有字符组成用户命令中的一个一个参数。将所有参数保存并返回。算法流程图如图 3.12 所示。

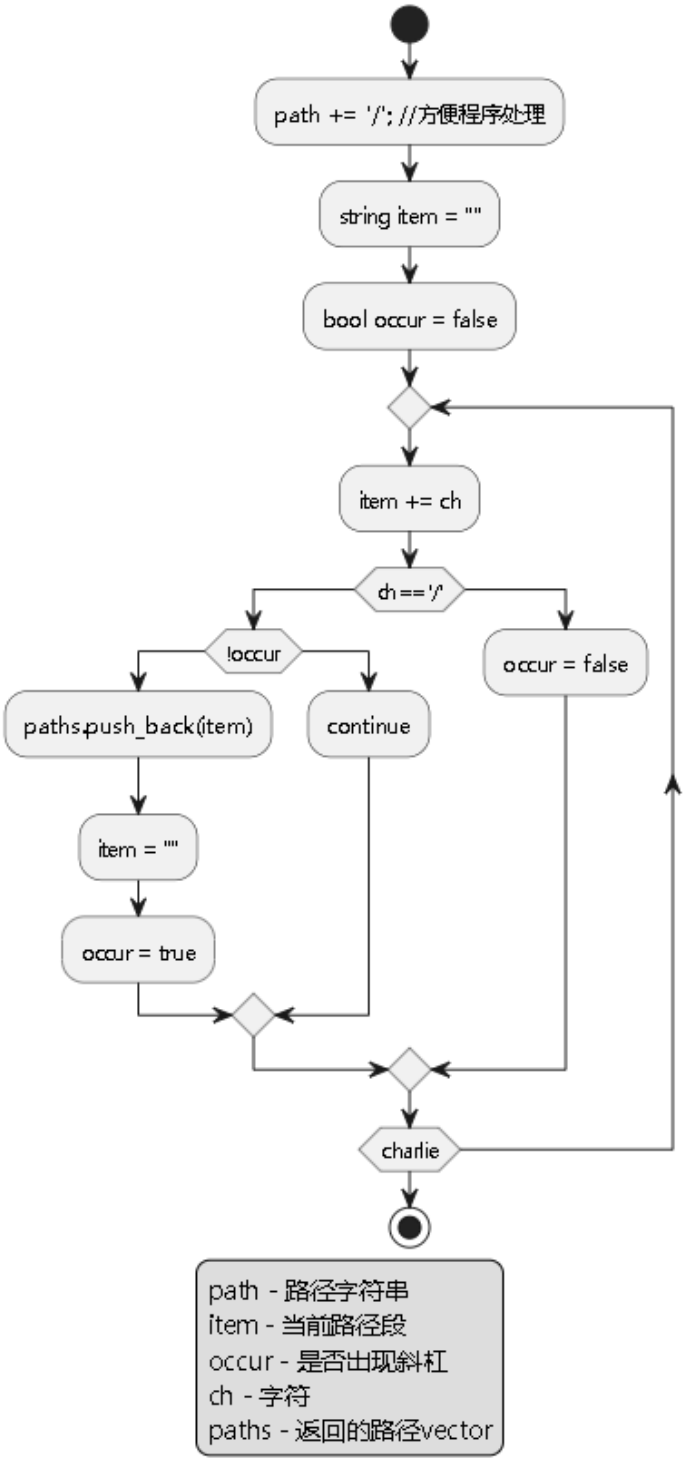

图 3.12 命令解析函数流程图

2. running_shell()用户界面主程序

算法流程: 打开文件系统后，首先登录。登录成功后，循环接收用户输入的命令，并且每次接收用户输入之前都输出命令行提示符前缀。接

收完毕用户输入后，将用户输入的命令进行分解，得到能被用户接口模块识别的参数。并且根据用户输入的命令名，调用用户接口模块中的相应功能。直到用户输入 logout 命令，且附带的参数为-e 时，文件系统才退出关闭。

3. 其他成员函数

其余的功能函数主要是给下层用户接口模块提供参数并且调用对应功能，这里就不再赘述，详细实现参见 3.4 节。

4 程序设计与实现

4.1 程序流程图

程序启动时，新建一个 Shell 对象，这个对象提供了用户界面。在初始化时，该对象初始化 UserInterface 对象，UserInterface 对象初始化 FileSystem 对象，FileSystem 对象初始化 DiskDriver 对象。启动时，用户界面向用户接口层发出初始化命令，用户接口层执行初始化，尝试挂载磁盘。若没有找到虚拟磁盘文件，则新建一个虚拟磁盘，由用户指定磁盘大小。若虚拟磁盘未格式化，则格式化磁盘。若虚拟磁盘文件已经被格式化，用户界面提示登录。登录认证成功后，用户可以键入各种各样的命令，命令经用户界面解析后传递给用户接口层。当用户执行退出命令时，所有驻留在内存中的数据写入磁盘，执行退出操作。

完整的程序流程图如图 4.1 所示。

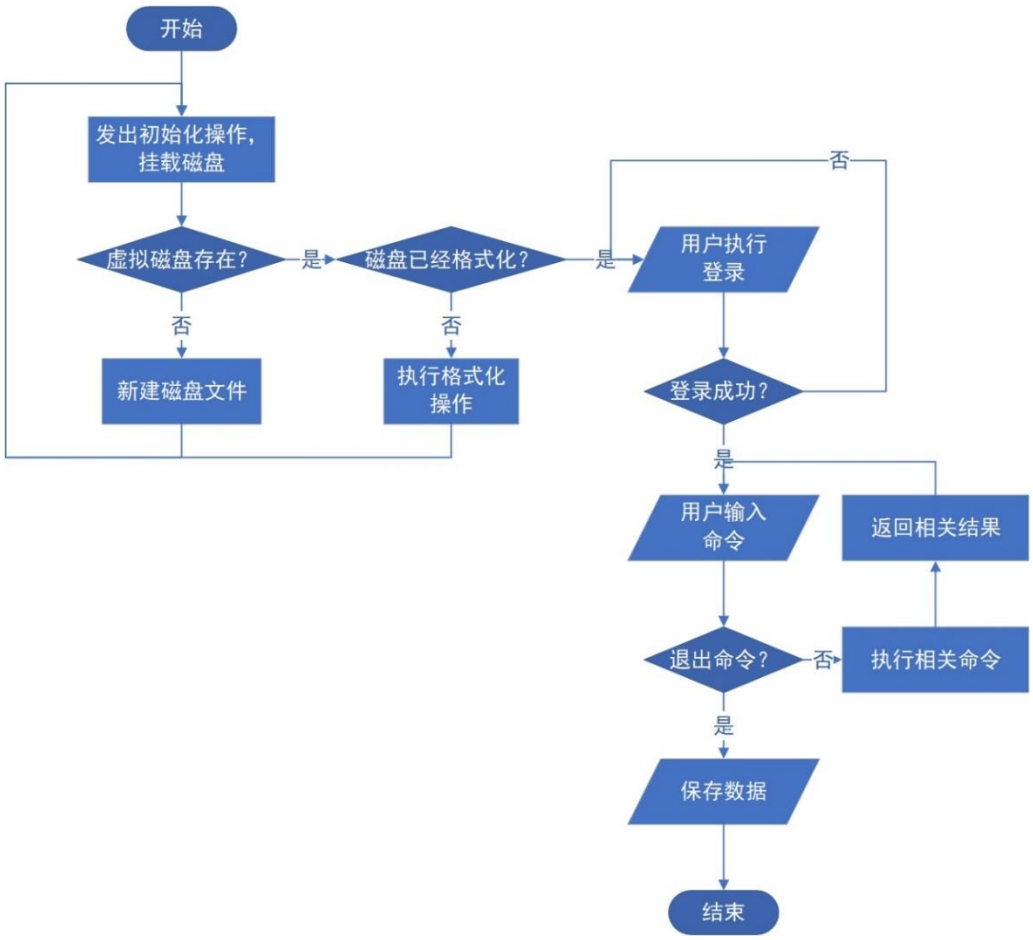

图 4.1 文件系统总体操作流程图

4.2 程序说明

（1）本程序为命令行程序，在 Windows 环境下使用 C++语言从零开始独立开发，使用的编译环境是 Mingw-w64 11.0 与 CMake 3.12，使用的集成开发环境是 CLion 2023.1 教育认证版。程序开发过程使用 Git 进行协作，代码托管在 Github 平台，地址为：https://github.com/zhangshifen38/FileSystem。

（2）要编译运行程序，可以使用 CLion 打开项目文件夹，配置好兼容的运行环境后编译运行，即可在内部控制台上进行操作。也可以手动使用 CMake 工具生成 Makefile 后进行编译。

（3）系统的使用方式和 Linux 的终端类似，支持的命令及用法在表 1.1 中已经给出，这里不再赘述。另外，由于编码等诸多原因，本程序未对中文进行支持。

4.3 实验结果

文件系统各操作执行结果如图 4.2-4.16 所示。

```
                +--------------------------------------+
                |    Simple FileSystem Simulation      |
                +--------------------------------------+
mount failed!
begin format!
format failed because there is no disk.
Start creating a disk, please input disk size(MB):
```

图 4.2 无虚拟磁盘情况下启动文件系统，提示创建磁盘，用户输入磁盘大小

```
                +--------------------------------------+
                |    Simple FileSystem Simulation      |
                +--------------------------------------+
mount failed!
begin format!
format failed because there is no disk.
Start creating a disk, please input disk size(MB):16
disk create success!
format success!
Login as:
```

图 4.3 设定磁盘大小后磁盘创建、格式化成功，进入登录步骤

```
                +--------------------------------------+
                |    Simple FileSystem Simulation      |
                +--------------------------------------+
mount failed!
begin format!
format failed because there is no disk.
Start creating a disk, please input disk size(MB):16
disk create success!
format success!
Login as: user1
Password: 123456
FileSystem@user1:/ $
```

图 4.4 登录成功，等待输入命令。默认用户名 user1-user8，默认密码 123456

```
FileSystem@user1:/ $ls
.       ..
FileSystem@user1:/ $mkdir dir1
FileSystem@user1:/ $touch file1
FileSystem@user1:/ $ls
.       ..      dir1    file1
FileSystem@user1:/ $touch dir1
touch:cannot create file 'dir1':File exists
FileSystem@user1:/ $
```

图 4.5 ls 命令、mkdir 命令与 touch 命令执行结果，目录用蓝色字表示，目录文件不同名

```
FileSystem@user1:/ $touch ./dir1/file1.1
FileSystem@user1:/ $ls /dir1
.       ..      file1.1
FileSystem@user1:/ $cd dir1
FileSystem@user1:/dir1/ $ls
.       ..      file1.1
FileSystem@user1:/dir1/ $
```

图 4.6 ls 命令、touch 命令、mkdir 命令与 cd 命令支持绝对路径与相对路径

```
FileSystem@user1:/dir1/ $mkdir ../dir2
FileSystem@user1:/dir1/ $cd ../
FileSystem@user1:/ $ls
.       ..      dir1    file1   dir2
FileSystem@user1:/ $cd dir2
FileSystem@user1:/dir2/ $rmdir ../dir1
FileSystem@user1:/dir2/ $ls ../
.       ..      file1   dir2
FileSystem@user1:/dir2/ $
```

图 4.7 rmdir 命令递归删除目录

```
FileSystem@user1:/dir2/ $rm /file1
FileSystem@user1:/dir2/ $cd ../
FileSystem@user1:/ $ls
.       ..      dir2
FileSystem@user1:/ $
```

图 4.8 rm 命令删除文件

```
FileSystem@user1:/ $cd dir2
FileSystem@user1:/dir2/ $touch file2
FileSystem@user1:/dir2/ $open -rw file2
FileSystem@user1:/dir2/ $read file2 50
read:

FileSystem@user1:/dir2/ $
```

图 4.9 open 命令打开文件，read 命令读至多 50 个字节，若不足读到结尾。默认空文件

```
FileSystem@user1:/dir2/ $read file2 50
read:

FileSystem@user1:/dir2/ $write file2 operating-system-project
FileSystem@user1:/dir2/ $seek file2 -b 5
FileSystem@user1:/dir2/ $read file2 20
read:
ting-system-project
FileSystem@user1:/dir2/ $
```

图 4.10 write 命令向文件中写入一句话，seek 命令移动文件指针到下标 5 处，读出数据

```
FileSystem@user1:/dir2/ $close file2
FileSystem@user1:/dir2/ $rename file2 fileT
FileSystem@user1:/dir2/ $ls
.       ..      fileT
```

图 4.11 close 命令关闭文件，rename 命令重命名文件为 fileT

```
FileSystem@user1:/dir2/ $mkdir /dir3
FileSystem@user1:/dir2/ $mv fileT ../dir3
FileSystem@user1:/dir2/ $ls
.       ..
FileSystem@user1:/dir2/ $ls /dir3
.       ..      fileT
FileSystem@user1:/dir2/ $
```

图 4.12 mv 命令移动文件到新建的目录 dir3

```
FileSystem@user1:/dir2/ $cd /dir3
FileSystem@user1:/dir3/ $open -wr fileT
FileSystem@user1:/dir3/ $read fileT 50
read:
operating-system-project
FileSystem@user1:/dir3/ $close fileT
FileSystem@user1:/dir3/ $
```

图 4.13 读取 fileT 中的内容，可以验证其经过重命名、移动后文件内容不变

```
FileSystem@user1:/dir3/ $logout
Login as: user8
Password: 123456
FileSystem@user8:/ $cd /dir3
FileSystem@user8:/dir3/ $ls
.       ..      fileT
FileSystem@user8:/dir3/ $
```

图 4.14 logout 命令登出当前用户，切换账号，数据存在

```
FileSystem@user8:/dir3/ $logout -e
Bye!

进程已结束,退出代码0
```

图 4.15 logout 的-e 参数代表结束模拟程序

```
+-------------------------------------+
|    Simple FileSystem Simulation     |
+-------------------------------------+
Login as: user4
Password: 123456
FileSystem@user4:/ $ls
.       ..      dir2    dir3
FileSystem@user4:/ $cd dir3
FileSystem@user4:/dir3/ $open -rw fileT
FileSystem@user4:/dir3/ $seek fileT -b 5
FileSystem@user4:/dir3/ $read fileT 6
read:
ting-s
FileSystem@user4:/dir3/ $close fileT
FileSystem@user4:/dir3/ $
```

图 4.16 重新启动程序，原数据依旧存在，实现了数据的持久化

5 结论

本次课程设计，我们团队花费 2 周的时间，设计了一个简单的模拟文件系统，系统架构合适、功能近乎完善、可拓展性强。文件系统采用树形目录结构，利用基于栈的空闲块列表方式组织管理磁盘块，文件主体采用索引结构。模拟程序操作界面为类 Linux 命令行，开发方式为从零开始全新开发，实现了全部的基本功能，并且附带实现了部分文件系统应该有的附加功能。

我们设计的文件系统仍然存在一些问题，包括但不限于：没有为 i 节点设立专门的储存空间，将 i 节点储存于磁盘的数据区，一定程度上造成了磁盘资源的浪费；单个目录容量有限，在 4096Byte 的块大小下，仅支持 256 个目录项；部分功能的实现仅考虑了能否实现，没有考虑实现的时间复杂度；等等。

通过课程设计，我们小组对于文件系统有了更加深入的理解，对文件、目录的存储结构以及文件操作有了更加清晰的认知。

6 参考文献

[1] 张尧学, 宋虹, 张高, 等. 计算机操作系统教程. 4 版. 北京:清华大学出版社, 2013.

[2] 张尧学, 等. 计算机操作系统教程习题解答与实验指导. 4 版. 北京:清华大学出版社, 2013.

[3] 汤小丹, 梁红兵, 哲凤屏, 汤子瀛, 等. 计算机操作系统. 4 版, 西安:西安电子科技大学出版社, 2014.

[4] （美）William Stallings, 等. 陈向群, 陈渝, 等译. 操作系统——精髓与设计原理. 9 版. 北京:电子工业出版社, 2020.

7 收获、体会和建议

7.1 组长收获、体会和建议

在课程设计开始时，由于老师提供的示例代码思路和我们小组不一致，修改起来诸多不便，于是决定重新开发一个模拟文件系统程序。在课程设计过程中，作为文件系统的主要设计者与底层模块的实现者，我广泛地查阅资料，通过不断的修改完善，最终实现了这个相对完整的文件系统，为编写应用层代码的组员提供了强有力的支撑。

本次课程设计系统性地帮助我复习了操作系统课上所学习的文件系统知识，并且在应用于实践的过程中，我也巩固了对 C++语言一些特性的使用，理解了二进制方式读写文件的原理与方法。在设计过程中，我们也不可避免地遇到了困难，但是在小组合作与指导教师的帮助下，问题一个个都得到了解决。

操作系统课程设计或许是大学本科阶段的最后一门课程设计，因此，我也格外珍惜这次机会，最后的结构也是令我满意的，在验收时，老师也给予了正面的评价，我觉得，我这两周的努力没有白费。

7.2 组员 1 收获、体会和建议

在本次操作系统课程设计中，我负责的是文件系统的所有命令业务逻辑的实现，包括常见的 cd、touch、mkdir、rm 等等命令的实现。由于我负责的功能的实现调用了下层提供的文件系统底层接口，首次采用分层设计的软件架构方法，让我整个课程设计的完成更加得心应手。课程设计的过程中，我不仅进一步掌握了操作系统的课程知识，也学习了一些课外的关于 linux 系统的知识，获益匪浅。在文件系统的所有命令的实现当中，我遇到的最困难的应该是 rmdir 命令的实现，因为该命令要删除指定目录下的所有内容，如果遇到目录要进行递归操作，实现对整个目录中所有内容的完整的递归删除与安全返回还原现场难度不小，过程中遇到了比如递归返回条件错误，递归顺序错误等等的问题，在大量调试后才最终完美实现需求功能。本次课程设计将课堂知识与实践紧密结合，我认为安排的十分合理，没有其他建议。

7.3 组员 2 收获、体会和建议

在本次操作系统课程设计中，我在团队中担任的角色是负责命令行 UI 的设计，主要交互提示词如 FileSystem@{UserName}+路径，类似 Linux Terminal 中命令输入之前的提示，还设计了对于用户输入的路径的分词操作，主要用于删去多余的“/”并将路径进行存储，此外还设计了 login 的正确访问的后续路径显示和登陆失败的容错，退出登录的相关操作等等，即使在团队中我的作用没有很答，没有设计文件系统中的重难点部分，但我还是从队友的代码之中学习到了很多操作系统知识，底层实现的思路和代码实现。在本次课设中，我在收获到了知识的同时也对团队协作能力的掌握更进一步，在这次课设中收获满满！