

操作系统课程 设计

目 录

1	设计内容	1
1.1	设计意义	1
1.2	设计说明	1
1.3	设计要求	2
1.3.1	问题描述	2
1.3.2	基本要求	2
1.3.3	指令设计	2
2	数据结构设计	3
2.1	磁盘块设计	3
2.2	主文件目录 MFD	4
2.2.1	结构定义	4
2.2.2	描述说明	4
2.3	用户文件目录 UFD	4
2.3.1	结构定义	4
2.3.2	描述说明	5
2.4	打开文件目录 OFD	5
2.4.1	结构定义	5
2.4.2	描述说明	5
2.5	数据块 Block	6
2.5.1	结构定义	6
2.5.2	描述说明	6
3	算法设计	7
3.1	功能模块图	7
3.2	算法描述	8
3.2.1	磁盘块分配算法	8
3.2.2	磁盘块回收算法	9
3.2.3	读文件	9
3.2.4	写文件	10
3.2.5	创建文件	11
3.2.6	删除文件	12
3.2.7	文件拷贝	13
3.2.8	属性修改	13
3.2.9	其他操作	14
3.3	算法思路	14
3.3.1	实现方法	14
3.3.2	设计思想	16

4	测试数据及程序运行情况	18
4.1	系统功能测试	18
4.1.1	用户注册与登录	18
4.1.2	创建和删除文件	18
4.1.3	读写文件	19
4.1.4	文件拷贝	20
4.1.5	属性修改	21
4.1.6	其他操作	23
4.2	输入验证测试	24
4.2.1	指令解析	24
4.2.2	非法输入	25
4.3	鲁棒性测试	27
4.3.1	用户注册数量边界测试	27
4.3.2	大规模数据测试	27
5	设计过程中出现的问题及解决方法	28
5.1	技术问题	28
5.1.1	指令解析	28
5.1.2	指针定位	28
5.2	细节问题	29
5.2.1	文件打开失败	29
5.2.2	未进行异常处理	29
5.2.3	fgets 函数	30
6	扩展功能与算法对比	31
6.1	扩展功能实现	31
6.2	算法对比	31
6.2.1	连续分配与链接分配	31
6.2.2	单级目录与两级目录	32
7	自我评析与总结	33
	参考文献	34

1 设计内容

1.1 设计意义

文件系统是用于组织、存储和管理计算机文件的一种系统软件。设计文件系统的主要意义在于提供了一种有序、结构化的方式来组织和管理计算机系统中的数据。通过层次结构的目录和文件形式，文件系统使用户能够方便地进行文件和目录操作，包括创建、删除、查看和修改文件，以及在不同层次的目录之间切换。此外，文件系统设计支持用户管理和权限控制，确保只有经过身份验证和授权的用户才能访问特定的数据，提高了信息安全性。这种设计有助于简化用户操作，降低系统复杂性，提高系统的可用性和稳定性，为用户提供了一个简单而完整的文件操作环境。

1.2 设计说明

本课程设计实现了两级文件目录系统（如图 1.1 所示），包括主文件目录（MFD）和用户文件目录（UFD），形成了一种层级结构。在 MFD 中，用户信息被存储，而在 UFD 中，每个用户拥有独立的文件目录，用于存储个人文件信息。

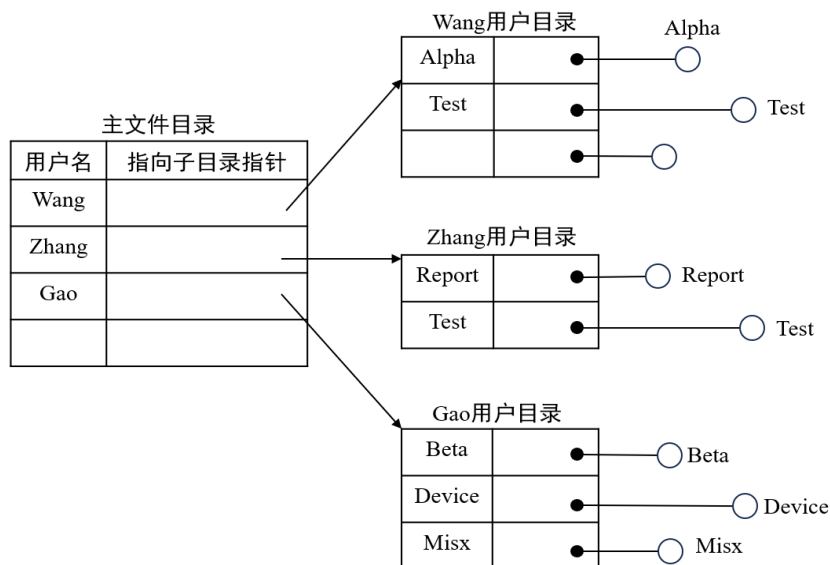


图 1.1 两级文件目录系统图

通过指令实现了一系列基本功能，如用户注册、登录、文件的创建、删除、查看信息，以及修改密码等操作。文件系统还支持目录结构的管理，用户可以从根目录到用户目录进行切换。这样的设计使得文件系统具备用户管理、文件管理、权限控制、目录结构维护等基本功能，为用户提供了一套简单而完整的文件操作环境。

1.3 设计要求

1.3.1 问题描述

在任一 OS 下，建立一个大文件，把它假象成一张盘，在其中实现一个简单的小型文件系统。

1.3.2 基本要求

该小型文件系统没有子目录机制，文件连续分配，不考虑分区。做一个简单的操作界面，提供四条简单的命令：简单的 ls、cat、cp、rd。

本进一步增强文件系统功能：文件系统不连续分配，可以有子目录机制，（如两级子目录机制）。

1.3.3 指令设计

设计要求必须实现下面 4 条指令：

- (1) ls: 打印目录下的所有文件信息。
- (2) cat <filename>: 打印用户指定文件信息，需用户输入文件名。
- (3) cp <srcFilename> <desFilename>: 复制源文件内容至目标文件。
- (4) rd <filename>: 删除当前目录下用户指定的文件，需要用户输入文件名。

在原有设计要求设计的指令外，为了实现文件系统的完整性和方便用户的操作与使用，额外扩展 11 条指令：

- (1) exit: 退出当前系统。
- (2) save: 将当前文件数据同步到磁盘。
- (3) cls: 用户认为当前屏幕内容过多可以使用该指令。
- (4) register <username> <password>: 用户注册。
- (5) login <username> <password>: 用户登录系统。
- (6) logout: 退出当前用户目录至根目录。
- (7) passwd <oldPwd> <newPwd>: 用户修改账户密码。
- (8) create <filename> <mode>: 在当前目录下创建文件，需用户输入文件名。
- (9) chmod <filename> <mode>: 修改用户指定文件的权限，需用户输入文件名。
- (10) rename <srcFilename> <desFilename>: 修改用户指定的文件名。
- (11) write <filename> <-m> <nbytes>: 向指定文件写指定字节长度的内容。

2 数据结构设计

2.1 磁盘块设计

该文件系统的磁盘区域划分为系统区和用户区。在系统区中，包括 MFD 块、UFD 块和 FOD 块，主要用于存放系统数据，以供文件系统管理和组织文件使用；与此同时，用户区由剩余的磁盘块组成，主要以 Block 为单位存放用户的实际数据。具体的磁盘块分布情况如图 2.1 所示。

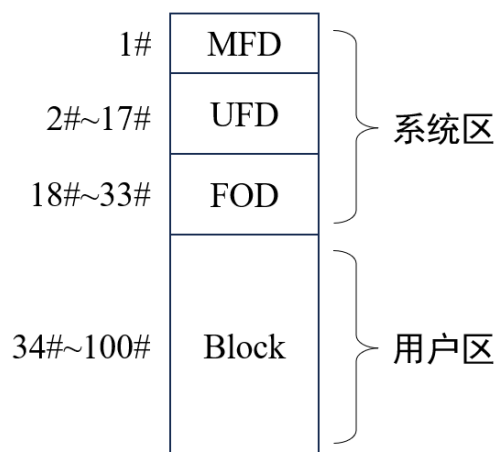


图 2.1 磁盘块分布图

MFD (Master File Directory) 块存储整个文件系统的主目录信息，UFD (User File Directory) 块存储用户文件的目录信息，而 FOD (File Open Directory) 块则用于存储已打开文件的信息。Block 数据块则用于存储实际的文件内容。磁盘块之间通过换行符 ('\n') 进行分割。如图 2.2 所示：

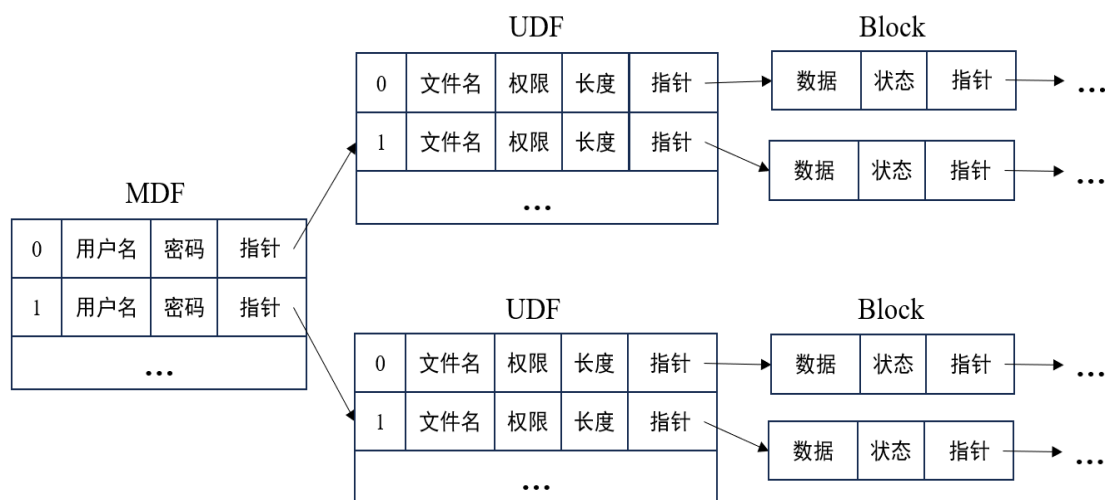


图 2.2 磁盘块设计图

2.2 主文件目录 MFD

2.2.1 结构定义

```
// MFD 结构体定义，表示主文件目录项
typedef struct mfd
{
    char username[14]; // 用户名，14 个字节
    char password[14]; // 用户密码，14 个字节
    int next;           // 该用户的 UFD 所在的盘号 4 个字节
} MFD;
```

2.2.2 描述说明

每个 MFD 项占 32 字节，其中包含用户名（14 字节）、用户密码（14 字节）和链接到该用户的 UFD 所在的盘号（4 字节）。整个文件系统最多可容纳 16 个用户，每个用户的信息占用一个 MFD 项。因此，1 个磁盘块可存放 $512/32=16$ 个 MFD(用户)，即该文件系统最多可管理 16 个用户。如下表：

表 2.1 MFD 示例表

username	password	next
uiui	123	1
popo	123	2

表 2.1 描述了一个文件系统中主目录文件（MFD）的结构。每个 MFD 项占用 32 字节，包括用户名、用户密码和链接到用户文件目录（UFD）所在盘号的信息。在示例表中，有两个用户，分别为"uiui"和"popo"，每个用户的信息存储在一个 MFD 项中，包括用户名、密码和 UFD 的盘号。整个文件系统最多可容纳 16 个用户，每个用户的信息占用一个 MFD 项。这种结构有助于有效地组织和管理用户信息以及与其关联的文件目录。

2.3 用户文件目录 UFD

2.3.1 结构定义

```
// UFD 结构体定义，表示用户文件目录项
typedef struct ufd
{
    char filename[20]; // 文件名，20 个字节
    int mode;           // 文件权限，0 - 只读；1 - 只写；2 - 可读可写
    int length;         // 文件长度（字节数）
    int address;         // 文件起始磁盘块地址
} UFD;
```

2.3.2 描述说明

每个 UFD 结构体占用 32 字节，包括文件名（20 字节）、文件权限（4 字节）、文件长度（4 字节）和文件起始磁盘块地址（4 字节）。假设每个用户需要一个独立的 UFD 块，因此总共需要 16 个 UFD 块。如下表：

表 2.2 UFD 示例表

filename	mode	length	address
1.txt	2	278	0
2.txt	2	43	2

表 2.2 描述了一个文件系统中用户文件目录（UFD）的结构。每个 UFD 结构体占用 32 字节，包括文件名、文件权限、文件长度和文件起始磁盘块地址。在示例表中，有两个文件，分别为 "1.txt" 和 "2.txt"。每个文件占用一个 UFD 块，其中包括文件名、权限、长度以及起始磁盘块地址的信息。以 "1.txt" 为例，其权限为 2，长度为 278 字节，起始磁盘块地址为 0。而 "2.txt" 的权限为 2，长度为 43 字节，起始磁盘块地址为 2。总共有 16 个 UFD 块，每个用户拥有一个独立的 UFD 块，用于存储其文件信息。这种结构有助于有效地管理每个用户的文件目录信息。

2.4 打开文件目录 OFD

2.4.1 结构定义

```
// ofd 结构体定义，表示用户打开文件项
typedef struct ofd
{
    char filename[20]; // 文件名，20 个字节
    int mode;           // 文件权限，0 - 只读；1 - 只写；2 - 可读可写
    int state;          // 文件状态，0 未打开，1 打开
    int write_point;    // 写指针位置
} OFD;
```

2.4.2 描述说明

每个 OFD 结构体占用 32 字节，包括文件名（20 字节）、文件权限（4 字节）、文件状态（4 字节）和写指针位置（4 字节）。假设每个用户可能同时打开一个文件，因此总共需要 16 个 OFD 块。如下表：

表 2.3 OFD 示例表

filename	mode	state
1.txt	1	0
2.txt	2	1

表 2.3 描述了一个文件系统中的打开文件目录（OFD）的结构。每个 OFD 结构体占用 32 字节，包括文件名、文件权限、文件状态和写指针位置。在示例表中，有两个打开的文件，分别为 "1.txt" 和 "2.txt"。每个文件占用一个 OFD 块，其中包括文件名、权限、状态以及写指针位置的信息。以 "1.txt" 为例，其权限为 1，状态为 0，表示文件当前未被修改，写指针位置为 0。而 "2.txt" 的权限为 2，状态为 1，表示文件当前处于修改状态，写指针位置为 1。总共有 16 个 OFD 块，每个用户可能同时打开一个文件，用于存储其打开文件的相关信息。这种结构有助于有效地跟踪和管理正在打开的文件。

2.5 数据块 Block

2.5.1 结构定义

```
// Block 结构体定义，表示磁盘上的一个磁盘块
typedef struct block{
    char data[248];    // 存储数据的数组，大小为 248 字节
    int is_rest;        // 标识是否空闲，0 表示空闲，1 表示占用
    int next;          // 下一个磁盘块的编号
} Block;
```

2.5.2 描述说明

每个 Block 结构体占用 256 字节，包括链接编号（4 字节）、空闲标识（4 字节）和数据内容（248 字节）。一个文件可占据多个磁盘块，如下表：

表 2.4 Block 示例表

index（隐含）	data	is_rest	next
0	This is is an	1	1
1	example of a data	1	2
2	representation	1	2
3	Null（空数据）	0	Null（空数据）
4	Null（空数据）	0	Null（空数据）

表 2.4 描述一个文件系统中数据块组织。每个数据块占用 32 字节，包括数据、标志位以及下一个数据块的索引。在示例表中，每个数据块包含数据、标志位（1 表示有数据，0 表示空数据），以及指向下一个数据块的索引。

数据块的索引（index）为 0 的数据块的下一个数据块（next）为 1，索引为 1 的数据块的下一个数据块为 2，索引为 2 的数据块的下一个数据块为 2。当索引等于下一个数据块时，表示已经到达链表的尾端，遍历结束。通过把这三个数据块的数据字段连接起来，可以得到完整的句子：“This is an example of a data representation”。而索引为 3 和 4 的数据块的 is_rest 字段均为 0，表示它们都是空闲数据块。

3 算法设计

3.1 功能模块图

在这个两级文件目录系统（如图 3.1）中，主文件目录和用户文件目录构成了系统的基本结构。首先，主文件目录下的功能模块包括用户注册、用户登录和文件操作。用户可以通过注册功能创建新的账户，通过登录功能进入系统。文件操作则涵盖了一系列基本的文件管理功能，如读写文件、拷贝文件、创删文件等。

在用户文件目录层级下，用户可以进行更加个性化的文件操作，包括对文件的读写、拷贝、创删等。此外，用户还可以选择进行用户登出操作，以安全退出当前账户，保护个人信息和文件安全。文件操作的丰富性使得用户可以方便地管理自己的文件资源。

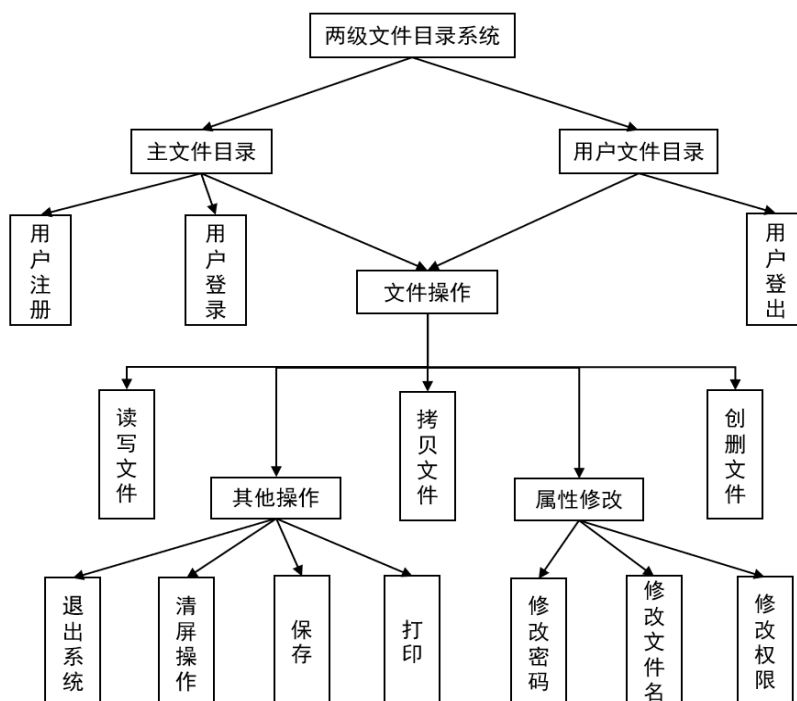


图 3.1 功能模块图

文件操作的核心功能包括读写文件、拷贝文件、创删文件。读写文件允许用户查看和编辑文件内容，拷贝文件支持用户在不同目录之间复制文件，创删文件则使用户能够新建文件或删除不再需要的文件。这些基本操作为用户提供了高度的文件管理灵活性。

系统还提供了一些其他操作，例如退出系统、保存、打印目录下文件信息、清屏以及登出。退出系统功能允许用户安全地关闭整个应用程序，保存操作用于确保用户的修改和更新得以保存。打印目录下文件信息则提供了查看文件列表的功能，清屏操作则用于清除屏幕上的信息，提供更好的用户体验。登出

操作可在用户使用完毕后安全退出当前账户。

最后，文件和用户的属性修改功能进一步增强了系统的灵活性和个性化。用户可以修改自己的密码，更改文件名，以及管理文件的权限，从而更好地适应不同用户的需求。这种两级文件目录系统的设计使得用户可以高效、方便地进行文件管理和个人信息维护。

3.2 算法描述

3.2.1 磁盘块分配算法

文件的创建和删除涉及对磁盘空间的分配和回收，因此必须建立一个有效的数据结构来追踪和记录磁盘的使用情况。为此，本系统将所有数据块 Block 组织成一张空闲物理块表，其中每个 Block 的 is_rest 字段用于标识数据块是否空闲，从而对磁盘空间进行管理。

在进行磁盘块分配之前，系统首先根据用户的请求计算所需的分配块数。例如，若用户请求 1000 字节的空間，且用户区的磁盘块大小为 256 字节，则请求分配块数为 $\lceil 1000/256 \rceil = 4$ 。系统根据这一计算得知至少需要分配 4 个磁盘块以满足用户的空间需求。

然后采用首次适应算法，即选择第一个满足要求的可用空闲块进行分配。具体而言，系统将检索空闲物理块表的第 i 项，并判断第 i 个磁盘块是否处于空闲状态。如果第 i 个磁盘块空闲，则为文件分配该物理块，并相应地更新物理块状态。然而，如果第 i 个磁盘块已经被占用，系统将在空闲物理块表中寻找下一个可用的物理块，直至找到足够的空闲磁盘块。流程图如图 3.2 所示。

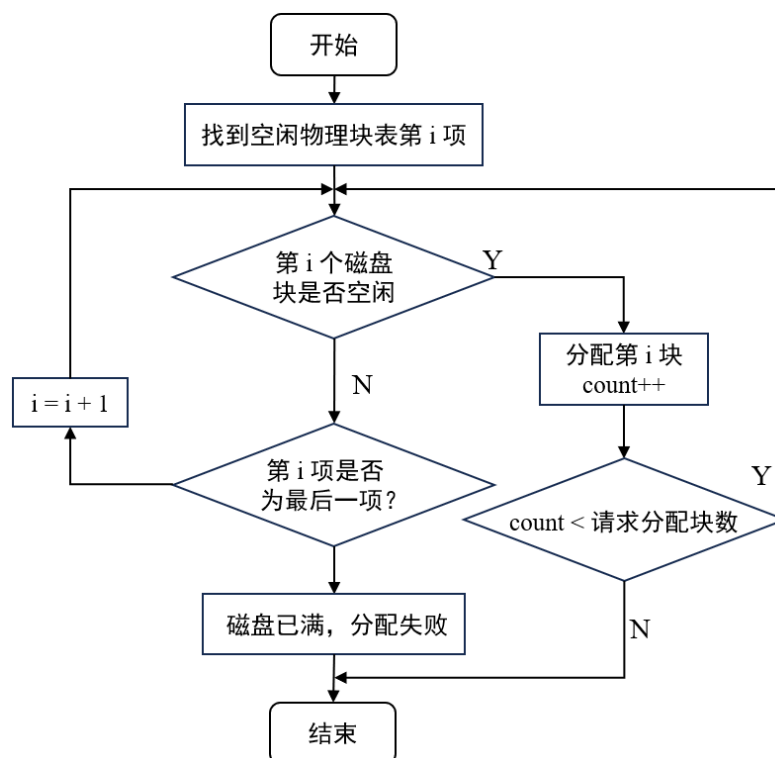


图 3.2 磁盘块分配流程图

3.2.2 磁盘块回收算法

当用户删除某一文件时，系统需要对该文件所占用的磁盘块进行回收，从而有效地利用磁盘空间。回收算法的流程相对分配算法更为简单，系统按照文件占用的磁盘块顺序进行顺序遍历，逐一将这些磁盘块的状态设置为空闲。为确保磁盘块的重复利用，系统同时进行数据清空操作，将回收的磁盘块内容清零。这一流程有效地维护了磁盘空间的有序性，并确保回收后的磁盘块可以安全地被其他文件再次利用。磁盘块回收流程如图 3.3 所示。

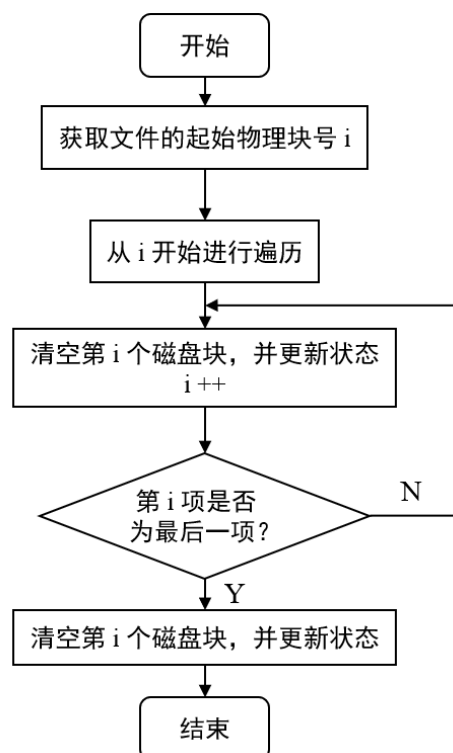


图 3.3 磁盘块回收流程图

3.2.3 读文件

在该系统中，进行读文件操作时，仅需提供文件名和读取长度，例如使用指令 `cat <文件名> <读取长度>`。由于系统采用流式文件结构，读取长度以字节为单位表示。

读文件的主要步骤包括以下几个方面：首先，系统会检查文件表中是否存在所指定的文件，若文件不存在，则读取操作无法执行。其次，系统会验证文件的访问权限，确保文件的权限设置为只读或可读可写，如果权限不符合要求，则阻止读取操作。最后，系统遍历文件所占用的物理块，逐一读取其内容。在读取的过程中，系统会判断是否遇到文件结束符，若遇到，则终止读操作，确保仅读取文件的实际内容。读文件操作流程如图 3.4 所示。

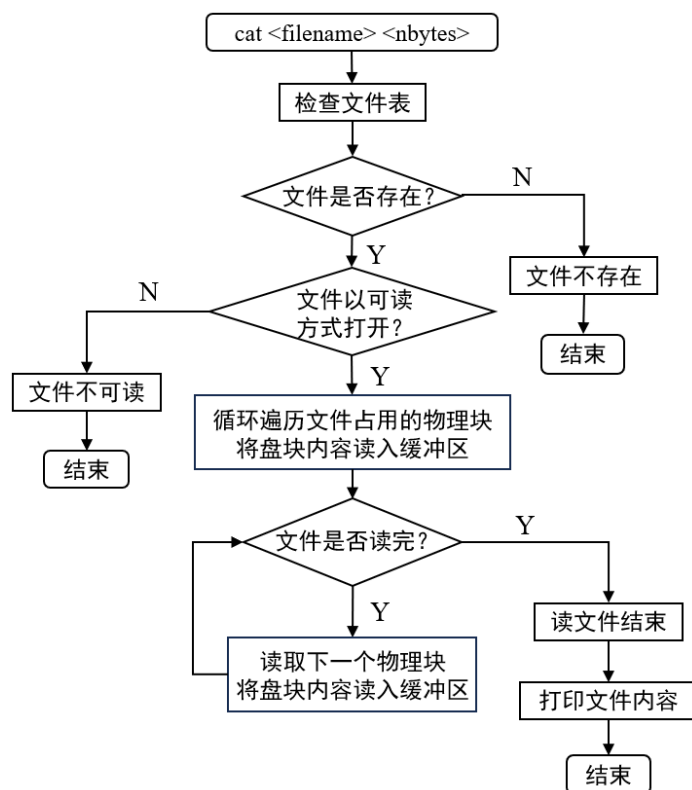


图 3.4 读文件操作流程

3.2.4 写文件

在该文件系统中，用户要求进行文件信息存取时，可通过调用文件系统的“写文件”操作来实现。用户执行写文件操作时，只需提供文件名、存放准备写入磁盘信息的缓冲区以及写的长度，例如使用指令 `write <filename> <-m> <nbytes>`。系统的文件结构采用流式结构，写长度以字节为单位表示。

写文件的主要步骤涵盖以下几个方面：首先，系统会检查文件表中是否存在用户指定的文件，若文件不存在，则写入操作无法执行。其次，系统会验证文件的访问权限，确保文件的权限设置为只写或可读可写，如果权限不符合要求，则会阻止写入操作。最后，系统从已打开文件表中读取写指针的位置，然后将缓冲区中的数据写入文件。整个写文件操作流程如图 3.5 所示。

这一设计确保了写文件操作的有效性和可控性。用户通过提供必要的信息，系统能够判断文件是否存在、验证权限是否合规，并在满足条件的情况下将数据写入文件。这种操作流程不仅保障了文件系统的安全性，也为用户提供了便捷的文件写入方式。

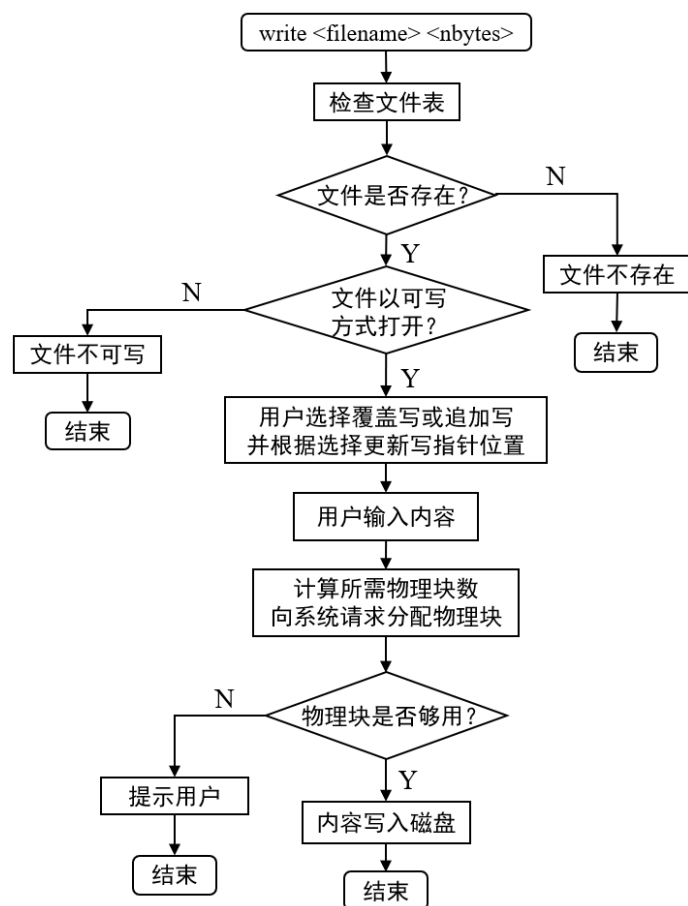


图 3.5 写文件操作流程

写文件操作在系统中存在两种情况：一是覆盖写，即新写入的内容会覆盖文件已有的内容；二是追加写，即新写入的内容会追加到文件已有内容的末尾。这两种情况的主要区别在于写指针的位置。

在覆盖写的情况下，写指针定位在文件的起始位置，而在追加写的情况下，写指针则定位在文件已有内容的末尾。系统在执行写文件操作时根据用户需求选择相应的写入方式，从而实现了文件内容的更新或追加。这灵活的设计允许用户根据具体需求选择适当的写入模式，提高了系统的适用性。

3.2.5 创建文件

用户欲将新文件存储到介质上时，首先执行文件系统的“创建”操作。创建文件所需的参数相对较少，仅提供文件名和文件权限，即使用指令 `create <filename> <mode>`。

创建文件的主要步骤涵盖以下几个方面：首先，系统会检查是否存在同名文件，若存在，则提示用户该文件已存在，创建文件失败；若无同名文件，则为新文件分配一个磁盘块；最后，系统会更新用户文件目录和打开文件表的信息。创建文件的具体流程如图 3.6 所示。

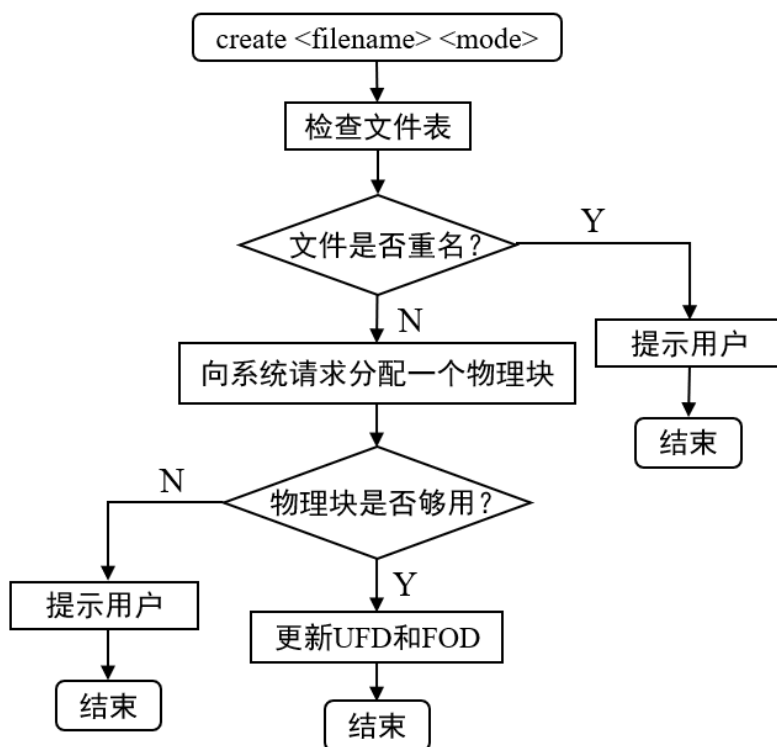


图 3.6 创建文件操作流程图

3.2.6 删除文件

用户认为文件没有必要保存时需要调用文件系统的“删除文件”操作。在该系统中，删除文件时参数只要文件名：**delete <文件名>**。

删除文件的主要步骤包括以下几个方面：首先，系统会检查文件是否存在，若不存在，则删除操作失败；若文件存在，则继续下一步。然后，系统会删除文件目录项并释放文件所占用的磁盘空间。删除文件的具体流程如图 3.7 所示。

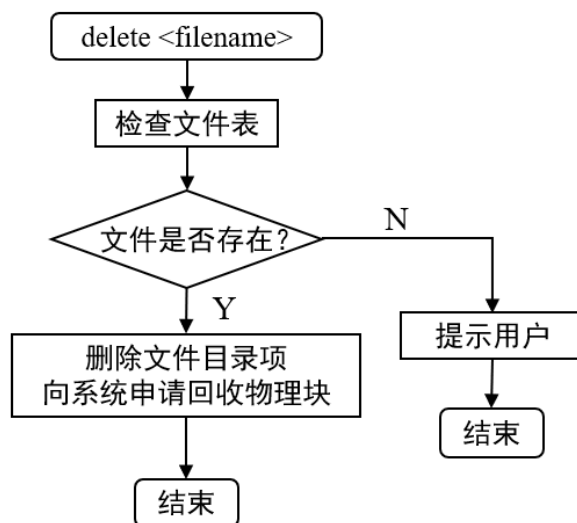


图 3.7 删除文件操作流程图

3.2.7 文件拷贝

在该系统中，拷贝文件时参数需要源文件名和目标文件名：`cp <srcfile> <descfile>`。

拷贝文件的主要步骤包括以下几个方面：首先，系统会检查源文件是否存在，若不存在，则拷贝操作失败；若文件存在，则继续下一步。然后，系统会检查目标文件是否存在，若不存在，则会创建一个目标文件，反之继续。接下来，获取源文件内容置缓冲区。最后将缓冲区内容写入目标文件。拷贝文件的具体流程如图 3.8 所示。

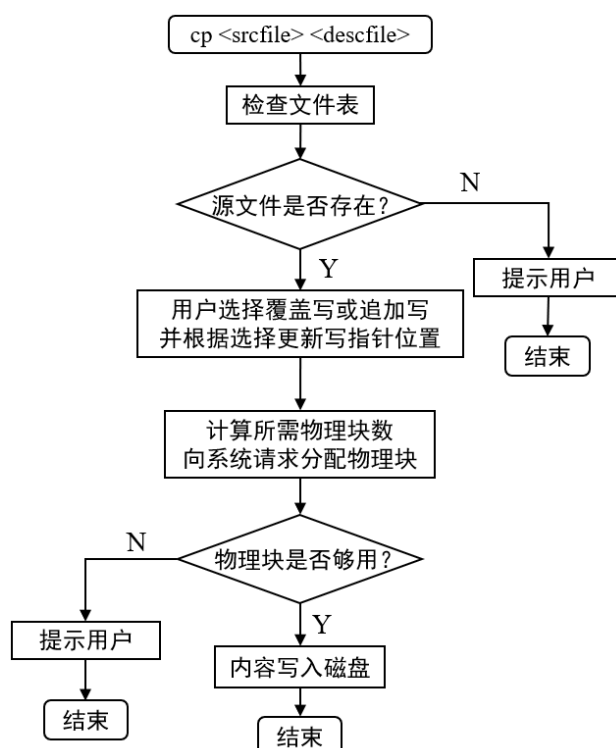


图 3.8 拷贝文件操作流程

拷贝文件有两种情况：一种情况是覆盖拷贝，即拷贝的内容会覆盖文件已有的内容。另一种情况是追加拷贝，即拷贝的内容会追加到文件已有内容的末尾。两者的区别在于写指针的位置不同。

3.2.8 属性修改

本系统实现了用户目录属性和文件属性的修改功能。用户目录属性的操作范围包括用户注册（`register`）、用户登录（`login`）和修改用户密码（`passwd`），而文件属性的修改则涵盖文件名（`rename`）和文件权限（`chmod`）的调整。这些操作共享相似的基本原理，即首先获取原始文件或目录的信息，随后采用新的信息进行替换。在以下说明中，我们以修改文件名为例，详细解释属性修改操作的步骤：

首先，系统会检查源文件是否存在，若不存在，则修改操作失败；若文件存在，则继续下一步。然后，系统会检查目标文件是否存在，若存在，则提示

用户文件重名，无法修改，反之继续。最后将新文件名替换旧文件名，并同步到磁盘中。修改文件名的具体流程如图 3.9 所示。

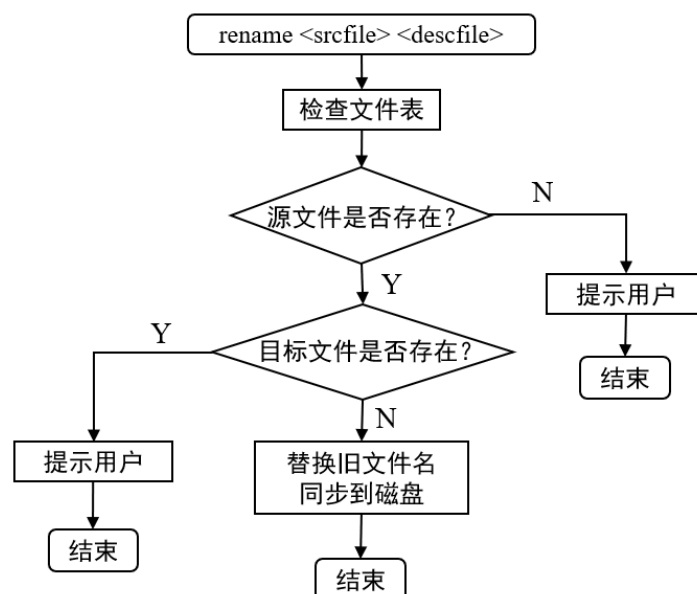


图 3.9 文件名修改操作流程

3.2.9 其他操作

为提升用户使用体验和确保系统功能完整性，我们的系统已经实现了一系列便捷操作。具体而言，这些操作包括退出系统（exit）、保存（save）、打印目录下文件信息（ls）、清屏（cls）以及登出（logout）。相对于系统的核心操作，这些功能较为简单，因此在这里不会进行过多的详细说明。

3.3 算法思路

3.3.1 实现方法

- (1) cat 函数：读取文件内容，并将文件的内容输出到终端。
- (2) copy 函数：用于复制文件。
- (3) copy_check 函数：检查复制操作的有效性。包括检查源文件是否存在、目标文件是否存在等。
- (4) copy_write 函数：将复制源文件内容写入目标文件，有两种复制方法，覆盖和追加。
- (5) create 函数：创建新的文件。
- (6) disk2mem 函数：将磁盘上的数据加载到内存中，以便进一步处理。
- (7) out_file 函数：将输出流导向文件，将缓冲区内容保存到磁盘文件中。
- (8) rd 函数：删除指定文件。
- (9) write 函数：向指定文件中写入数据。
- (10) chage_psw 函数：用于更改用户密码。
- (11) clear 函数：用于清空终端屏幕的内容。

- (12) `cmd_in_parse` 函数：用于解析命令行输入，将用户输入的命令分解成可处理的部分。
- (13) `exit` 函数：退出程序或者终端会话。
- (14) `help` 函数：用于显示帮助信息，列出可用命令和其说明。
- (15) `login` 函数：用于实现用户登录。
- (16) `logout` 函数：用于实现用户登出。
- (17) `ls` 函数：用于列出当前目录下的文件或子目录。
- (18) `Register` 函数：用于实现用户注册。
- (19) `rename` 函数：用于重命名文件。
- (20) `save` 函数：保存当前状态或数据至磁盘。

在本文件系统中，主要的函数调用关系如图 3.10 所示。

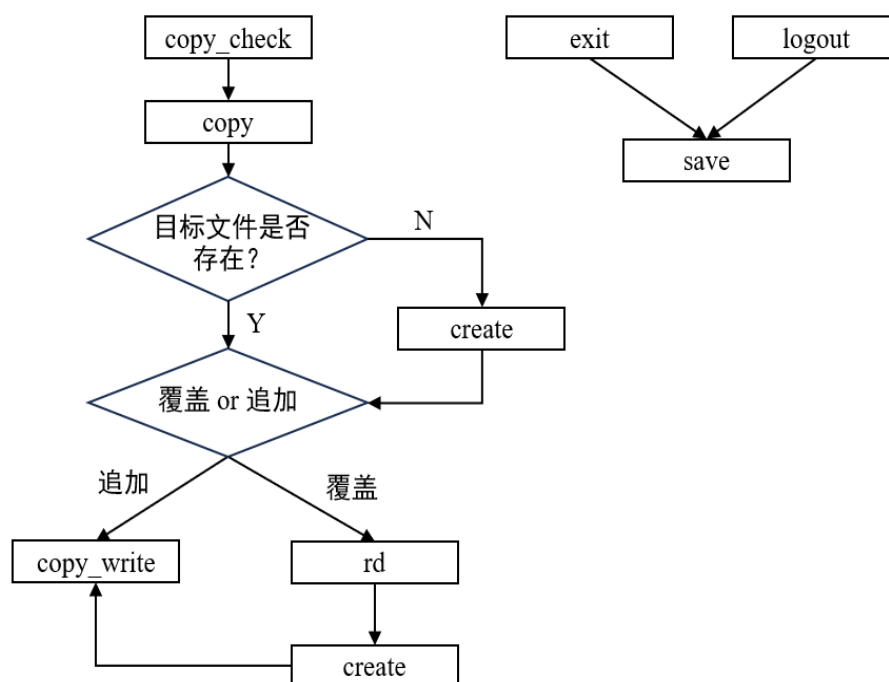


图 3.10 函数关系调用图

当用户输入 `copy` 命令时，系统首先调用 `copy_check` 函数，用于检查目标文件是否存在。如果目标文件不存在，系统会调用 `creat` 函数创建新文件，否则继续执行。用户接着选择覆盖拷贝或追加拷贝，若选择追加拷贝，则系统调用 `copy_write` 函数直接进行拷贝操作。如果用户选择覆盖拷贝，系统先调用 `rd` 函数删除目标文件，然后调用 `create` 函数创建新的目标文件，最后进行拷贝操作。

另一方面，当用户执行 `exit` 或 `logout` 命令时，系统首先调用 `save` 函数，将当前系统状态或数据保存至磁盘，确保数据的持久性。随后，系统执行 `exit` 或 `logout` 函数，完成用户退出操作。这样的设计保证了在退出系统之前先保存了可能的修改，以确保数据的完整性和可恢复性。

3.3.2 设计思想

(1) 定义全局变量

```
//定义数据结构，表示用户信息、文件信息、文件状态、磁盘块
vector<MFD> UsrInfo;           // 主文件目录项向量
vector<vector<UFD>>> FileInfo; // 用户文件目录项二维向量
vector<vector<OFD>>> FileState; // 用户打开文件项二维向量
vector<Block> FileCluster;     // 磁盘物理块向量
// 定义单个数据项的输入结构体
MFD UsrInput;                 // 用户输入的主文件目录项
UFD FileInput;                // 用户输入的用户文件目录项
OFD StateInput;               // 用户输入的用户打开文件项
Block ClusterInput;           // 用户输入的磁盘物理块
//定义用户变量，方便编程
int num;                      // 选项选择变量
int Headnum;                  // 头节点编号
int curID;                    // 当前用户的编号
// 命令处理函数列表
Hand_to handlerlist[] = {
    {"chmod", chmod}, {"rename", rename}, {"cp", copy_check}, {"rd", rd},
    {"cat", cat}, {"passwd", chage_psw}, {"login", login}, {"help", help},
    {"logout", logout}, {"create", create}, {"write", write}, {"ls", ls},
    {"save", save}, {"register", Register}, {"exit", exit}, {"cls", clear},
    {NULL, NULL}
};
```

(2) 主函数模块

```
// 主程序入口
int main(){
    system("color F0");
    loginWelcome();           // 登陆欢迎信息
    disk2mem();               // 初始化文件系统到内存
    // 主循环
    while (1){
        // 默认以 root 用户登陆
        cout << "\\\" << UsrInfo[curID].username << "> ";
        cmd_in_parse();       // 解析用户输入的命令
    }
    return 0;
}
```

主函数模块主要包括全局变量的声明、指令解析以及从磁盘加载数据至内存，实现全局变量的初始化。全局变量的声明和初始化有助于各功能模块的有效实现。用户输入指令后，通过指令解析调用相应的功能模块，功能模块处理完毕后将结果存储至内存变量，并将变量数据保存至磁盘，以确保下一次程序运行时能够维持正确的状态。主函数模块结构图如图 3.11 所示。

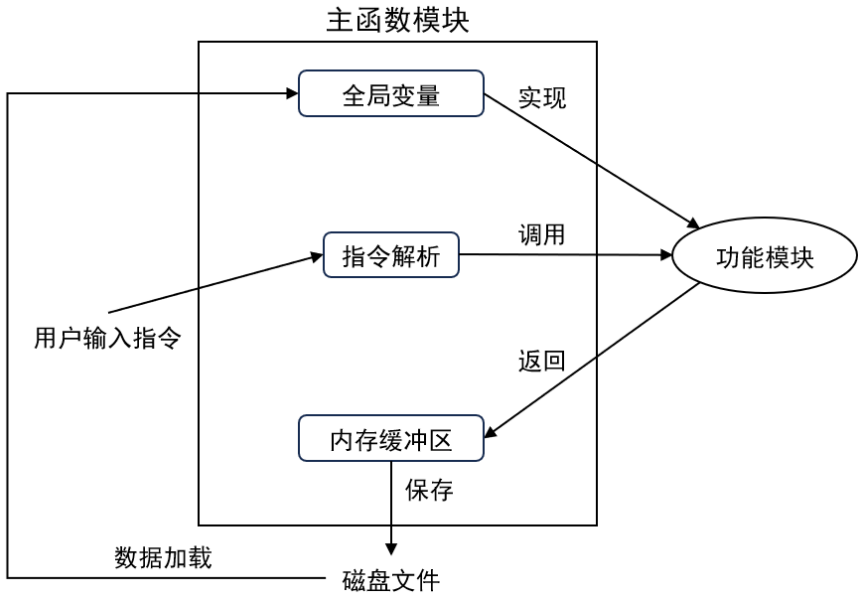


图 3.11 主函数模块结构图

（3） 其他模块说明

- 1) 用户注册和登录功能：该模块允许新用户通过注册功能创建账户，提供个性化服务入口，同时通过登录功能验证用户身份，确保只有合法用户可以访问其文件和个人信息。
- 2) 读写文件：允许用户查看和编辑文件内容，提供直接访问文件数据的能力，支持用户对文本或二进制文件的处理。
- 3) 拷贝文件：允许用户在不同目录之间复制文件，便于文件的备份、移动和共享，提高了文件管理的灵活性。
- 4) 创建和删除文件：提供创建新文件和删除不再需要的文件的功能，使用户能够自由地组织和清理文件资源。
- 5) 其他操作功能：包括退出系统、保存、打印目录下文件信息、清屏以及登出等，提升了系统的整体用户体验。退出系统功能允许用户安全地关闭整个应用程序，保存操作确保了用户的修改和更新得以保存。打印目录下文件信息功能提供了查看文件列表的便捷方式，清屏操作提供了更好的用户界面清理。登出操作允许用户在使用完毕后安全退出当前账户，增强了用户的隐私和安全性。
- 6) 文件和用户属性修改功能：用户可以通过该模块修改密码，提高账户的安全性。同时，修改文件名和文件权限功能增强了用户对文件的控制权，使文件管理更为灵活。

4 测试数据及程序运行情况

4.1 系统功能测试

4.1.1 用户注册与登录

(1) 用户注册功能

测试用例：register user1 123

使用 register 命令注册一个新用户，其中用户名是 user1，密码是 123。键入命令后，输入回车，系统给出创建成功的提示。注册功能测试图如图 4.1。



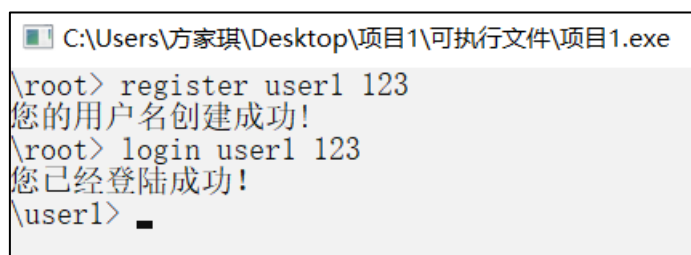
```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\root> register user1 123
您的用户名创建成功!
\root> _
```

图 4.1 注册功能测试图

(2) 用户登录功能

测试用例：login user1 123

在注册完成之后，使用 login 命令登录系统，其中用户名是 user1，密码是 123。键入命令后，输入回车，系统给出登录成功的提示。登录功能测试图如图 4.2。



```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\root> register user1 123
您的用户名创建成功!
\root> login user1 123
您已经登陆成功!
\user1> _
```

图 4.2 登录功能测试图

4.1.2 创建和删除文件

(1) 创建文件功能

测试样例：create 1.txt 2

在登录系统之后，使用 create 命令可以创建文件，其中 1.txt 是文件名，2 是文件的权限，即可读可写。键入命令后，输入回车，系统给出文件创建成功的提示。文件创建功能测试图如图 4.3。

```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\root> register user1 123
您的用户名创建成功!
\root> login user1 123
您已经登陆成功!
\user1> create 1.txt 2
文件创建成功
\user1> █
```

图 4.3 文件创建功能测试图

(2) 删除文件功能

测试样例：rd 1.txt

由于上面已经创建过文件 1.txt, 因此想要删除文件时, 需要使用 rd 命令。其中 1.txt 是文件名。键入命令后, 输入回车, 系统给出文件删除成功的提示。文件删除功能测试图如图 4.4。

```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\root> register user1 123
您的用户名创建成功!
\root> login user1 123
您已经登陆成功!
\user1> create 1.txt 2
文件创建成功
\user1> rd 1.txt
删除成功!
\user1> █
```

图 4.4 文件删除功能测试图

4.1.3 读写文件

(1) 写入文件功能

测试用例：create 1.txt 2

write 1.txt -m 500

1

The sun dipped below the horizon, casting a warm glow across the tranquil ocean. Gentle waves whispered against the shore, creating a soothing melody. Seagulls soared in the pastel-colored sky, their calls blending with the soft rustle of palm leaves. As twilight embraced the landscape, the serenity of the moment enveloped everything, creating a magical atmosphere that seemed to suspend time itself.

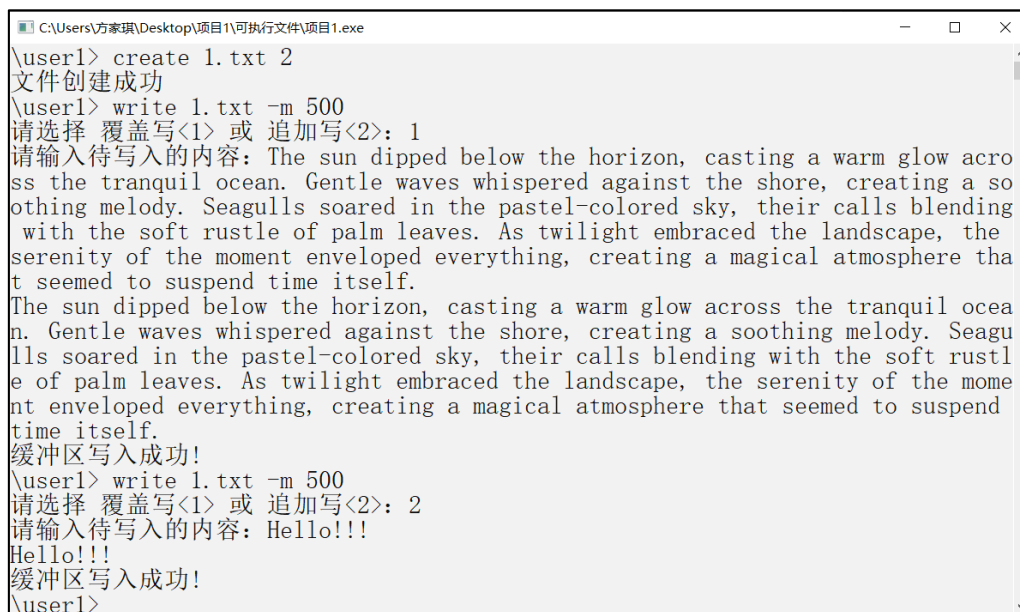
write 1.txt -m 500

2

Hello!!!

由于上面对 1.txt 进行了删除, 所以想要实现写文件功能, 需要先创建文件。首先使用 create 命令创建文件 1.txt。然后使用 write 命令向 1.txt 文件写入

内容，其中 1.txt 是文件名，-m 是缓冲区占位符，500 是待写入的字节长度。键入命令后，输入回车，系统给出写入成功的提示。文件写入功能测试图如图 4.5。



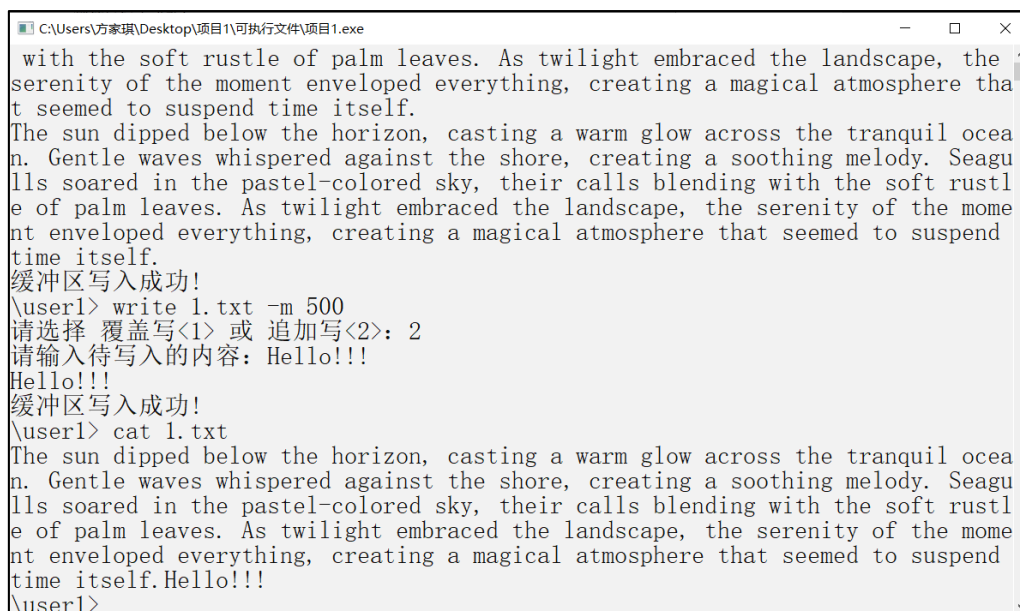
```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\user1> create 1.txt 2
文件创建成功
\user1> write 1.txt -m 500
请选择 覆盖写<1> 或 追加写<2>: 1
请输入待写入的内容: The sun dipped below the horizon, casting a warm glow across the tranquil ocean. Gentle waves whispered against the shore, creating a soothing melody. Seagulls soared in the pastel-colored sky, their calls blending with the soft rustle of palm leaves. As twilight embraced the landscape, the serenity of the moment enveloped everything, creating a magical atmosphere that seemed to suspend time itself.
The sun dipped below the horizon, casting a warm glow across the tranquil ocean. Gentle waves whispered against the shore, creating a soothing melody. Seagulls soared in the pastel-colored sky, their calls blending with the soft rustle of palm leaves. As twilight embraced the landscape, the serenity of the moment enveloped everything, creating a magical atmosphere that seemed to suspend time itself.
缓冲区写入成功!
\user1> write 1.txt -m 500
请选择 覆盖写<1> 或 追加写<2>: 2
请输入待写入的内容: Hello!!!
Hello!!!
缓冲区写入成功!
\user1>
```

图 4.5 文件写入功能测试图

(2) 读取文件功能

测试用例：cat 1.txt

在写入文件之后，使用 cat 命令可以读取文件内容，其中 1.txt 是文件名。键入命令后，输入回车，系统将会打印文件的内容。文件读取功能测试图如图 4.6。



```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
with the soft rustle of palm leaves. As twilight embraced the landscape, the serenity of the moment enveloped everything, creating a magical atmosphere that seemed to suspend time itself.
The sun dipped below the horizon, casting a warm glow across the tranquil ocean. Gentle waves whispered against the shore, creating a soothing melody. Seagulls soared in the pastel-colored sky, their calls blending with the soft rustle of palm leaves. As twilight embraced the landscape, the serenity of the moment enveloped everything, creating a magical atmosphere that seemed to suspend time itself.
缓冲区写入成功!
\user1> write 1.txt -m 500
请选择 覆盖写<1> 或 追加写<2>: 2
请输入待写入的内容: Hello!!!
Hello!!!
缓冲区写入成功!
\user1> cat 1.txt
The sun dipped below the horizon, casting a warm glow across the tranquil ocean. Gentle waves whispered against the shore, creating a soothing melody. Seagulls soared in the pastel-colored sky, their calls blending with the soft rustle of palm leaves. As twilight embraced the landscape, the serenity of the moment enveloped everything, creating a magical atmosphere that seemed to suspend time itself.Hello!!!
\user1>
```

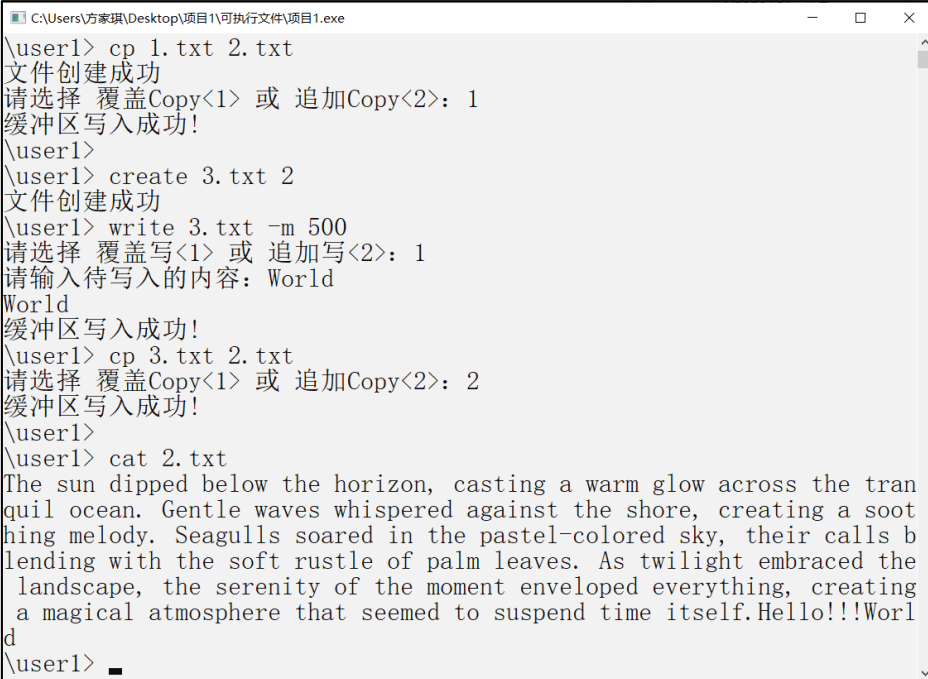
图 4.6 文件读取功能测试图

4.1.4 文件拷贝

测试用例：cp 1.txt 2.txt

```
1
create 3.txt 2
write 3.txt -m 500
1
World
cp 3.txt 2.txt
2
cat 2.txt
```

在上述操作的基础上，可以使用 `cp` 命令对文件进行拷贝操作。其中 1.txt 是源文件，2.txt 是目标文件。键入命令并按下回车后，系统会提示用户选择拷贝类型——覆盖拷贝和追加拷贝。前者拷贝的内容会覆盖文件已有的内容。后者拷贝的内容会追加到文件已有内容的末尾。对两种功能都进行测试，并使用 `cat` 命令打印文件内容。文件拷贝功能测试图如图 4.7。



```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\user1> cp 1.txt 2.txt
文件创建成功
请选择 覆盖Copy<1> 或 追加Copy<2>: 1
缓冲区写入成功!
\user1>
\user1> create 3.txt 2
文件创建成功
\user1> write 3.txt -m 500
请选择 覆盖写<1> 或 追加写<2>: 1
请输入待写入的内容: World
World
缓冲区写入成功!
\user1> cp 3.txt 2.txt
请选择 覆盖Copy<1> 或 追加Copy<2>: 2
缓冲区写入成功!
\user1>
\user1> cat 2.txt
The sun dipped below the horizon, casting a warm glow across the tran
quil ocean. Gentle waves whispered against the shore, creating a soot
hing melody. Seagulls soared in the pastel-colored sky, their calls b
lending with the soft rustle of palm leaves. As twilight embraced the
landscape, the serenity of the moment enveloped everything, creating
a magical atmosphere that seemed to suspend time itself.Hello!!!Worl
d
\user1> █
```

图 4.7 文件拷贝功能测试图

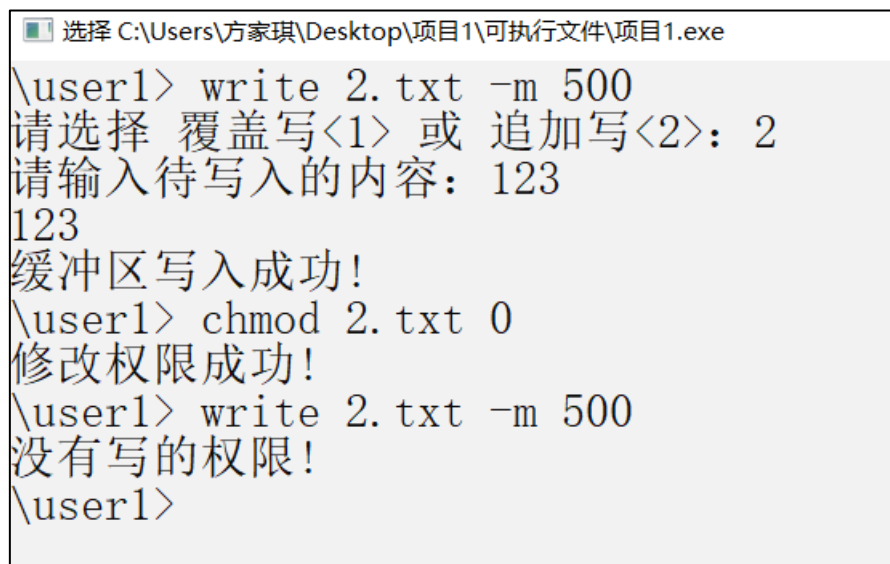
4.1.5 属性修改

(1) 修改文件权限

```
测试用例: write 2.txt -m 500
2
123
chmod 2.txt 0
write 2.txt -m 500
```

以 2.txt 为例来测试文件权限修改功能。首先，使用 `write` 命令对 2.txt 进行写入操作，系统成功写入文件，说明此时 2.txt 是可写的。然后使用 `chmod`

命令修改 2.txt 的权限为只读。最后再次使用 write 命令对 2.txt 进行写入操作，此时系统提示用户对该文件没有写的权限，说明成功地修改了文件的权限。文件权限修改功能测试图如图 4.8。



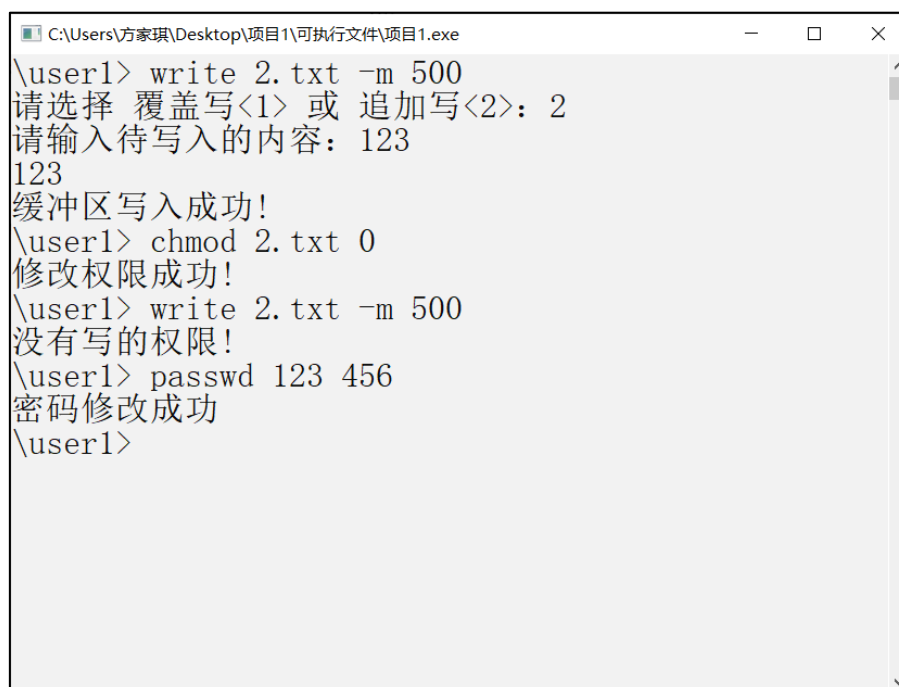
```
\user1> write 2.txt -m 500
请选择 覆盖写<1> 或 追加写<2>: 2
请输入待写入的内容: 123
123
缓冲区写入成功!
\user1> chmod 2.txt 0
修改权限成功!
\user1> write 2.txt -m 500
没有写的权限!
\user1>
```

图 4.8 文件权限修改功能测试图

(2) 修改用户密码

测试用例: passwd 123 456

可以使用 passwd 命令对用户密码进行修改，其中 123 是旧密码，456 是新密码。用户密码修改功能测试图如图 4.9。通过图 4.9，可以清晰地了解密码修改功能过程，包括输入旧密码和新密码的流程，以及任何相关的反馈或提示信息。



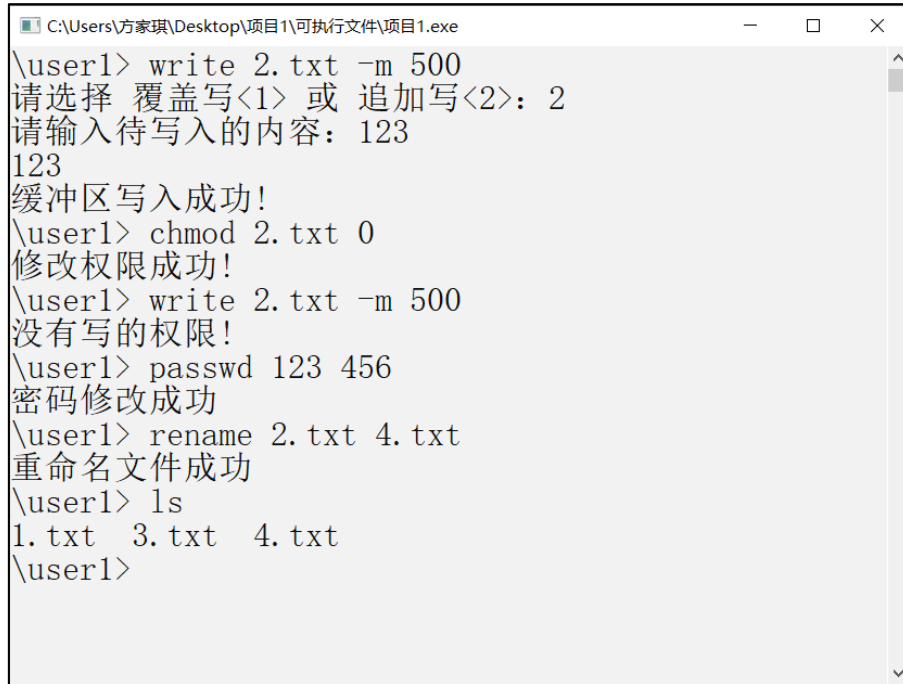
```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\user1> write 2.txt -m 500
请选择 覆盖写<1> 或 追加写<2>: 2
请输入待写入的内容: 123
123
缓冲区写入成功!
\user1> chmod 2.txt 0
修改权限成功!
\user1> write 2.txt -m 500
没有写的权限!
\user1> passwd 123 456
密码修改成功
\user1>
```

图 4.9 用户密码修改功能测试图

(3) 修改文件名

测试用例：rename 2.txt 4.txt

可以使用 rename 命令对文件名进行修改，其中 2.txt 是旧文件名，4.txt 是新文件名。文件名修改功能测试图如图 4.10。通过图 4.10，可以看到文件名修改功能过程，包括输入旧文件名和新文件名的流程，以及任何相关的反馈或提示信息。



```
\user1> write 2.txt -m 500
请选择 覆盖写<1> 或 追加写<2>: 2
请输入待写入的内容: 123
123
缓冲区写入成功!
\user1> chmod 2.txt 0
修改权限成功!
\user1> write 2.txt -m 500
没有写的权限!
\user1> passwd 123 456
密码修改成功
\user1> rename 2.txt 4.txt
重命名文件成功
\user1> ls
1.txt 3.txt 4.txt
\user1>
```

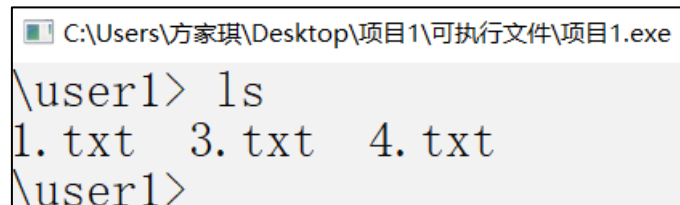
图 4.10 文件名修改功能测试图

4.1.6 其他操作

(1) 打印文件信息

测试用例：ls

使用 ls 命令，可以打印目录下的所有文件信息。文件信息打印功能测试图如图 4.11。通过图 4.11，可以清晰地文件的打印信息，包括该目录下所有的文件名。



```
\user1> ls
1.txt 3.txt 4.txt
\user1>
```

图 4.11 文件信息打印功能测试图

(2) 文件保存

测试用例：save

使用 save 命令，可以将内存缓存区内容同步到磁盘。文件保存功能测试图如图 4.12。

```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\user1> ls
1.txt  3.txt  4.txt
\user1> save
写入到磁盘成功!
\user1>
```

图 4.12 文件保存功能侧视图

(3) 系统退出

测试用例: exit

使用 exit 命令,可以退出终端。系统退出功能测试图如图 4.13。值得注意的是,在退出终端的过程中,系统实施了一种安全机制。在执行 exit 命令之前,系统会首先执行 save 命令,将内存缓存区的内容同步到磁盘。这一步骤的目的是确保用户的所有操作和数据得以妥善保存,以防止任何潜在的数据丢失或不完整。终端退出的流程可以概括为以下两个步骤:首先,通过执行 save 命令,系统将确保所有未保存的更改都被记录到磁盘;接着,执行 exit 命令,用户可以安全地退出终端系统。系统退出功能测试图如图 4.13。

```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\user1> ls
1.txt  3.txt  4.txt
\user1> save
写入到磁盘成功!
\user1> exit
写入到磁盘成功!
成功退出!

-----
Process exited after 480.8 seconds with return value 0
请按任意键继续. . .
```

图 4.13 系统退出功能测试图

4.2 输入验证测试

4.2.1 指令解析

测试用例: help

writtt

用户在系统操作中,首先可以利用 help 命令获取详细的帮助菜单,其中包括系统支持的各种命令及其用法。这样的帮助信息有助于用户更好地理解系统功能和操作流程。进一步,在系统的指令解析测试中,我们进行了一项测

试用例，其中用户输入了一个拼写错误的命令 `writtt`。系统在接收到无效指令后，迅速作出响应，发现该命令无法正确解析和执行。用户将会收到一条明确的错误提示，指出输入的命令无效。指令解析测试图如图 4.14。

```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
-----两级文件目录系统 (@方家琪)-----
输入help获得帮助菜单
\root> help
*****
*      exit          退出程序          *
*      save          同步到磁盘        *
*      cls           清屏              *
*      logout        登出              *
*      ls            列出所有文件      *
*      rd filename   删除文件          *
*      cat filename  显示文件内容      *
*      login username pwd  登录        *
*      register username pwd 注册      *
*      passwd oldPwd newPwd 修改密码    *
*      create filename mode 创建文件    *
*      chmod filename mode  改变文件属性 *
*      rename srcFile desFile 改变文件名 *
*      cp srcFile desFile  文件拷贝    *
*      write filename -m nbytes 写文件   *
*****
\root> writtt
命令无效!!!
\root>
```

图 4.14 指令解析测试图

4.2.2 非法输入

(1) rename 命令测试

测试用例：login user1 456

```
ls
rename 4.txt 3.txt
rename 2.txt 3.txt
rename 2.txt 2.txt
rename 2.txt 5.txt
```

使用 `login` 命令登录系统，在执行文件操作前使用 `ls` 命令查看当前目录下的文件列表。随后，尝试通过 `rename` 命令重命名文件。

首先，试图将 `4.txt` 重命名为 `3.txt`，但由于新文件名与已有文件名重复，系统提示重新设置。接着，两次尝试将 `2.txt` 重命名为 `3.txt` 和 `2.txt` 均失败，分别因为新文件名已存在和旧文件名与新文件名相同。最后，用户尝试将不存在的文件 `2.txt` 重命名为 `5.txt`，系统提示旧文件名不存在。这一系列命令信息展示了系统对文件重命名操作的有效性检查和错误反馈机制。`rename` 输入测试图如图 4.15。

```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\root> login user1 456
您已经登陆成功!
\user1> ls
1.txt  3.txt  4.txt
\user1> rename 4.txt 3.txt
新文件名与现有文件名重复, 请重新设置!!!
\user1> rename 2.txt 3.txt
新文件名与现有文件名重复, 请重新设置!!!
\user1> rename 2.txt 2.txt
旧文件名应于新文件名不同!!!
\user1> rename 2.txt 5.txt
旧文件名不存在!!!
\user1> █
```

图 4.15 rename 输入测试图

(1) 其他命令测试

测试用例: ls

cp 2.txt 3.txt

write 2.txt -m 500

login user100 123

cat 2.txt

通过执行 ls 命令查看当前目录下的文件列表, 发现存在三个文件: 1.txt、3.txt、4.txt。

接着, 尝试使用 cp 命令复制文件, 但因为源文件 2.txt 不存在, 系统提示源文件不存在, 要求重新输入。随后, 使用 write 命令编辑文件 2.txt, 然而同样由于文件名不存在而导致操作失败, 系统提示重新输入命令。进一步, 用户尝试以用户名 user100 和密码 123 登录, 但系统反馈用户名不存在, 要求用户重新登录。最后, 执行 cat 命令查看文件 2.txt 的内容, 但由于文件名不存在, 系统显示文件名不存在的错误提示。整个过程突显了系统对于文件操作中不同情况的错误检测 and 用户友好的反馈机制。各种命令输入测试图如图 4.16。

```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
\user1> ls
1.txt  3.txt  4.txt
\user1> cp 2.txt 3.txt
源文件不存在!!!
\user1> write 2.txt -m 500
文件名不存在, 请重新输入命令!!!
\user1> login user100 123
您的用户名不存在, 请重新登陆!
\user1> cat 2.txt
文件名不存在!!!
\user1>
```

图 4.16 各种命令输入测试图

4.3 鲁棒性测试

4.3.1 用户注册数量边界测试

测试用例：register user2 123

register user3 123

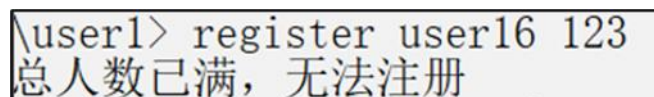
register user4 123

register user5 123

...

register user16 123

由于系统空间最多容纳 16 个用户，有上面的测试可知，系统当前存在 root 和 user1 用户。如果再注册 15 个新用户，则系统会提示总人数已满，无法注册。边界测试图如图 4.17 所示。该测试说明了系统对于用户注册数量上限的边缘情况能够有效处理，确保了系统达到最大用户数时能够提供准确的反馈。



```
\user1> register user16 123
总人数已满，无法注册
```

图 4.17 边界测试图

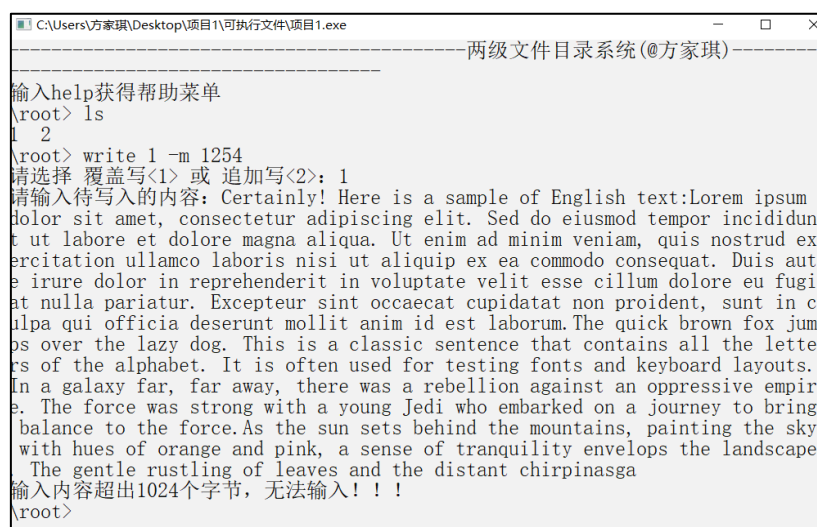
4.3.2 大规模数据测试

测试用例：write 1.txt -m 124245

2

<大量文本数据>

由于系统空间开辟的缓冲区大小有限，所以必须对输入的文本数据大小进行限制，此处限制文本输入内容不能超过 1024 字节，当输入文本内容超过 1024 字节时，系统会提示无法写入。大规模数据测试图如图 4.18 所示。



```
C:\Users\方家琪\Desktop\项目1\可执行文件\项目1.exe
-----两级文件目录系统(@方家琪)-----
输入help获得帮助菜单
\root> ls
1 2
\root> write 1 -m 1254
请选择 覆盖写<1> 或 追加写<2>: 1
请输入待写入的内容: Certainly! Here is a sample of English text:Lorem ipsum
dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididun
t ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud ex
ercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aut
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugi
at nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in c
ulpa qui officia deserunt mollit anim id est laborum.The quick brown fox jum
ps over the lazy dog. This is a classic sentence that contains all the lette
rs of the alphabet. It is often used for testing fonts and keyboard layouts.
In a galaxy far, far away, there was a rebellion against an oppressive empir
e. The force was strong with a young Jedi who embarked on a journey to bring
balance to the force.As the sun sets behind the mountains, painting the sky
with hues of orange and pink, a sense of tranquility envelops the landscape
The gentle rustling of leaves and the distant chirpinasga
输入内容超出1024个字节，无法输入!!!
\root>
```

图 4.18 大规模数据测试图

5 设计过程中出现的问题及解决方法

5.1 技术问题

5.1.1 指令解析

(1) 所遇问题

在众多对输入命令的函数处理中，使用传统的 `if..else..` 或者 `switch..case..` 的方法存在着代码的冗余和可读性的缺失。这种方式要求为每个命令都要单独处理，导致代码量庞大且难以维护。这不符合代码规范中简洁高效的原则，也使得代码的结构显得不够清晰。

(2) 解决办法

为了解决这个问题，我采用了函数指针的方法。具体而言，我定义了一个结构体，专门用于存储命令的字符串和相应的函数处理名称（函数指针）。这样，我只需要在结构体中定义一次命令与处理函数的映射关系，然后通过一次 `for` 循环遍历结构体就可以高效地处理这些命令。这种设计既减少了冗余代码，又提高了代码的简洁性和可读性。通过函数指针的灵活运用，我不仅符合了代码规范中简洁高效的原则，还加深了对 C++ 语法中函数指针应用的理解。这样的代码结构使得命令的添加和维护更加方便，提升了整体代码的可维护性和可扩展性。

5.1.2 指针定位

(1) 问题描述

在对文件内容进行分块存储时，使用了 OFD（打开文件目录）记录了文件内容的起始物理块。对于写指针来说，这种记录方式足以定位当前位置，因为只需记录最后一个物理块的偏移量即可，而在追加写入时，可以直接迭代到最后一个物理块进行写入。然而，对于读指针来说，仅记录最后一个物理块的偏移量是不够的，因为上一次读的位置不一定位于文件末尾，这可能导致没有数据记录当前读的物理块的块号。

(2) 解决办法

为了解决读指针的问题，作者采用了一种巧妙的解决方法。在读指针方面，不仅记录了最后一个物理块的偏移量，而且记录了当前字节的总数。这样，通过读指针除以每个物理块的字节数（256 字节）可以得出当前的物理块的块号，而读指针对每个物理块字节数取余可以得出当前读的物理块的偏移量。这种记录方式有效地解决了读指针在定位和计算物理块信息时可能遇到的问题，确保了对文件内容的准确读取和定位。这种巧妙的设计在文件的分块存储中为读操作提供了高效而可靠的方法。

5.2 细节问题

5.2.1 文件打开失败

(1) 所遇问题

在 C++ 中，处理文件输入输出流时常见的一个 bug 是在打开文件时未检查文件是否成功打开，可能导致后续的文件操作无法正常执行，特别是在尝试读取或写入文件时可能引发未定义的行为或崩溃。

(2) 解决办法

为了避免这个问题，我学到了一些良好的实践和解决办法，即在打开文件之后立即检查文件是否成功打开。这可以通过检查文件流对象的状态来实现，例如，使用 `if (!fileStream)` 来验证文件是否成功打开。核心代码如下：

```
#include <iostream>
#include <fstream>

int main() {
    std::ifstream inputFile("input.txt");
    if (!inputFile.is_open()) {
        std::cerr << "Error opening input file!" << std::endl;
        return 1;
    }
    // 其他文件操作...
    inputFile.close();
    return 0;
}
```

通过这种方式，可以确保在文件操作之前进行了合适的错误检查，以防止由于文件打开失败而导致的问题。这一做法提高了程序的健壮性，避免了潜在的运行时错误。

5.2.2 未进行异常处理

(1) 所遇问题

文件操作可能引发异常，如文件不存在、权限问题等。如果未正确处理这些异常，程序可能会在运行时崩溃。

(2) 解决办法

使用异常处理机制，捕获可能发生的异常，并进行适当的错误处理。核心代码如下：

```
try {
    std::ifstream inputFile("nonexistent_file.txt");
    // 文件不存在，但没有异常处理
}
```



```

    } catch (const std::exception& e) {
        std::cerr << "Exception: " << e.what() << std::endl;
    }

```

在上述代码中，try 块包含可能引发异常的代码。如果在 try 块中的代码导致异常（例如，尝试打开一个不存在的文件），则程序会立即跳转到匹配的 catch 块。在这里，我们捕获了 std::exception 类型的异常，并通过 e.what() 获取异常的描述信息，然后将错误消息输出到标准错误流（std::cerr）。

这样做的好处是，即使程序发生异常，它也能够通过异常处理机制优雅地退出，并提供有用的错误信息，而不至于导致程序崩溃。这有助于调试和排除问题，提高了程序的稳定性和可维护性。

5.2.3 fgets 函数

（1）所遇问题

在从磁盘中读取文件信息至内存的过程中，出现了乱码现象。在调试过程中发现，问题的根本原因是因为 fgets 函数的参数设置不当。

（2）解决办法

问题的核心在于 fgets 函数的参数设置。原始设置为 fgets(buffer, sizeof(buffer), file) 表明从文件中读取 sizeof(buffer) 个字节的内容到缓冲区 buffer 中。然而，在 C++ 中，字符串是以 \0 结尾的，因此应确保缓冲区足够容纳字符串和结尾的空字符。

改进后的解决办法是调整 fgets 函数的参数，将缓冲区的大小设置为 sizeof(Tempbuf)，即 fgets(Tempbuf, sizeof(Tempbuf), streamInit)。这样确保了 Tempbuf 缓冲区有足够的空间来容纳读取的 256 个字符及其结尾的空字符。这样的调整有效解决了由于截断字符串而导致的乱码问题，保证了读取的内容能够正确被处理并存储在缓冲区中。这种改进是在读取字符串时始终考虑字符串结尾的空字符，以防止潜在的缓冲区溢出和乱码问题的一种良好实践。核心代码如下：

```

char Tempbuf[258]; //定义 Tempbuff 变量
fgets(Tempbuf, 257, streamInit); //读取 256 个字符，但应设置参数为 257

```

6 扩展功能与算法对比

6.1 扩展功能实现

课程设计的基本要求包含了四个基本的文件系统指令，分别是 `ls`、`cat`、`cp` 和 `rd`，涵盖了列出目录内容、查看文件内容、复制文件和删除文件等功能。为了提升文件系统的完整性和用户友好性，本文件系统额外扩展了 11 个指令，分别是：

- (1) `exit`：退出当前系统。
- (2) `save`：将当前文件数据同步到磁盘。
- (3) `cls`：用户认为当前屏幕内容过多可以使用该指令。
- (4) `register <username> <password>`：用户注册。
- (5) `login <username> <password>`：用户登录系统。
- (6) `logout`：退出当前用户目录至根目录。
- (7) `passwd <oldPwd> <newPwd>`：用户修改账户密码。
- (8) `create <filename> <mode>`：在当前目录下创建文件，需用户输入文件名和文件的权限。
- (9) `chmod <filename> <mode>`：修改用户指定文件的权限，需用户输入文件名和修改后的文件权限。
- (10) `rename <srcFilename> <desFilename>`：修改用户指定的文件名。
- (11) `write <filename> <-m> <nbytes>`：向指定文件写入指定字节长度的内容。

这些额外扩展的指令使得文件系统更具实用性和灵活性，用户可以进行更多操作，包括注册、登录、密码修改、文件创建、权限管理、文件重命名、文件写入等。每个指令都对应一些具体的文件系统操作，使得用户能够更方便地管理和操作文件系统。这样的设计不仅满足基本的文件管理需求，还考虑了用户交互和系统的安全性。

6.2 算法对比

6.2.1 连续分配与链接分配

课程设计的基本要求是文件采用连续分配方式，而且不考虑文件系统的分区。然而，本文件系统的设计中选择了链接分配，并且充分考虑了系统区和用户区的划分。这个设计决策基于以下几个主要原因，旨在提高文件系统的效率和适应性。

首先，链接分配相对于连续分配具有更多的优点。连续分配可能导致外部碎片，文件大小受限，而且删除和增长文件的操作可能引起文件的移动，带来

额外的管理复杂性。相比之下，链接分配避免了外部碎片问题，支持变长文件，且操作相对简单。这使得文件系统更加灵活，能够适应不同大小的文件和频繁变化的文件大小需求。

其次，引入系统区和用户区的划分是为了更好地管理和保护文件系统的核心信息。系统区用于存储文件系统的元数据，而用户区则用于存放用户数据。这种划分提高了文件系统的组织性和安全性，有助于防止用户误操作对文件系统造成的不可逆损害。

综合考虑文件系统的设计要求和实际需求，链接分配和系统区与用户区的划分更好地符合文件系统的开发目标。链接分配为文件的管理提供了更大的灵活性，而系统区和用户区的划分则有助于提高文件系统的稳定性和安全性。这样的设计决策旨在创建一个更加健壮和适应性强的文件系统。

6.2.2 单级目录与两级目录

课程设计的基本要求不需要子目录机制，只需实现单级目录即可，但本文件系统设计了二级目录机制，这使得文件系统的组织性和管理效率得到了提高，且在不同用户目录下文件可实现重名机制。

首先，采用两级目录结构可以有效地改善文件系统的查找效率。单级目录在文件数量较少时可能表现良好，但随着文件数量的增加，查找时间可能会显著增加。通过引入两级目录，文件的文件得以按照更有层次的结构进行组织，提高了查找效率，尤其是在大规模文件系统中。

其次，两级目录提供了更好的组织性和可管理性。文件按逻辑关系被放置在不同的目录中，使得文件系统的结构更清晰，更容易管理。用户能够更轻松地找到所需的文件，而不必浏览整个文件系统。

引入两级目录结构后使得文件系统更具扩展性，适应了更大规模、更复杂文件组织的需求。这样的设计决策考虑到了文件系统的长期发展和应对更为复杂应用场景的可能性，为文件系统的实现增加了灵活性和可扩展性。

7 自我评析与总结

在课程设计中，我成功实现了一个具备基本文件系统指令的两级文件目录系统，包括 `ls`、`cat`、`cp` 和 `rd` 等四个基本功能，满足了文件系统的基本要求。此外，为了提升系统的完整性和用户友好性，我额外扩展了 11 个指令，包括 `exit`、`save`、`cls`、`register`、`login`、`logout`、`passwd`、`create`、`chmod`、`rename` 和 `write` 等功能，使得用户能够更方便地管理和操作文件系统。

设计上的亮点包括引入链接分配以提高系统的灵活性，划分系统区和用户区以增强系统的组织性和安全性，以及引入两级目录结构以提高文件系统的查找效率和可管理性。这些设计决策使得系统更加全面、灵活，并且有助于适应不同的文件管理需求。

虽然扩展功能在丰富系统的同时为用户提供了更多的操作选项，但在实现这些额外功能的过程中，需要特别关注用户交互体验。重点在于确保系统能够提供清晰、友好的错误处理机制，以及详尽而易懂的提示信息，以提高用户体验。这是一个值得进一步改进的地方。

自评成绩：95

参考文献

- [1] 杨琼,任晓瑞,张鹏.文件系统通用技术框架设计[J].航空计算技术,2023,53(06):80-83+87.
- [2] 张强强. 分布式文件系统在鲲鹏 ARM 架构下的性能与优化研究[D].兰州大学,2023.DOI:10.27204/d.cnki.glzhu.2023.001346.
- [3] 李翔. Ceph 分布式文件系统的研究及性能测试[D].西安电子科技大学,2014.
- [4] 石珊. 云平台下基于 FastDFS 的文件管理系统的研究与实现[D].电子科技大学,2020.
- [5] 苗佳欣. 基于 CRUSH 算法的分布式网络文件系统异构存储的研究与实现[D].华东师范大学,2022.DOI:10.27149/d.cnki.ghdsu.2022.001993.