

华南理工大学

《操作系统》课程实验报告

实验题目：_____ 计算机操作系统实验报告 _____

姓名：_____ 陈明奕 _____ 学号：_____ 201930640018 _____

班级：_____ 19 计科 2 班 _____ 组别：_____

合作者：_____

指导教师：_____ 刘发贵 _____

实验概述

【实验目的及要求】

实验目的：

本设计的目的是实现操作系统和相关系统软件的设计，其中涉及进程编程、I/O 操作、存储管理、文件系统等操作系统概念。

实验要求：

(1) 对进行认真分析，列出实验具体步骤，写出符合题目要求的程序清单，准备出调试程序使用的数据。

(2) 以完整的作业包的形式提交原始代码、设计文档和可运行程序。提交的光盘应当包括：设计题目，程序清单，运行结果分析，所选取的算法及其优缺点，以及通过上机取得了哪些经验。程序清单要求格式规范，注意加注释（包含关键字、方法、变量等），在每个模块前加注释，注释不得少于 20%。课程设计同时上交打印文档，设计报告包括设计题目，算法分析，关键代码及其数据结构说明，运行结果分析以及上机实践的经验总结。

设计任务：

设计一：

设计任务：模拟 Linux 文件系统

在任一 OS 下，建立一个文件，把它假象成一张盘，在其中实现一个简单的模拟 Linux 文件系统。

1. 在现有机器硬盘上开辟 100M 的硬盘空间，作为设定的硬盘空间。
2. 编写一管理程序 simdisk 对此空间进行管理，以模拟 Linux 文件系统，要求：
 - (1) 盘块大小 1k
 - (2) 空闲盘块的管理：Linux 位图法
 - (3) 结构：超级块，i 结点区，根目录区
3. 该 simdisk 管理程序的功能要求如下：
 - (1) info: 显示整个系统信息(参考 Linux 文件系统的系统信息)，文件可以根

据用户进行读写保护。目录名和文件名支持全路径名和相对路径名，路径名各分量间用“/”隔开。

(2) cd ...: 改变目录: 改变当前工作目录，目录不存在时给出出错信息。

(3) dir ...: 显示目录: 显示指定目录下或当前目录下的信息，包括文件名、物理地址、保护码、文件长度、子目录等（带/s参数的dir命令，显示所有子目录）。

(4) md ...: 创建目录: 在指定路径或当前路径下创建指定目录。重名时给出出错信息。

(5) rd ...: 删除目录: 删除指定目录下所有文件和子目录。要删目录不空时，要给出提示是否要删除。

(6) newfile ...: 建立文件。

格式: newfile 文件名.扩展名 保护类型 文件内容

(7) cat ...: 打开文件。//显示文件内容

(8) copy ...: 拷贝文件，除支持模拟Linux文件系统内部的文件拷贝外，还支持host文件系统与模拟Linux文件系统间的文件拷贝，host文件系统的文件命名为<host>...，如: 将windows下D: 盘的文件\data\sample\test.txt文件拷贝到模拟Linux文件系统中的/test/data目录，windows下D: 盘的当前目录为D: \data，则使用命令:

simdisk copy <host>D: \data\sample\test.txt /test/data

或者: simdisk copy <host>D: sample\test.txt /test/data

(9) del ...: 删除文件: 删除指定文件，不存在时给出出错信息。

(10) check: 检测并恢复文件系统: 对文件系统中的数据一致性进行检测，并自动根据文件系统的结构和信息进行数据再整理。 //文件系统一致性

4. 程序的总体流程为:

(1) 初始化文件目录;

(2) 输出提示符，等待接受命令，分析键入的命令;

(3) 对合法的命令，执行相应的处理程序，否则输出错误信息，继续等待新命令，直到键入EXIT退出为止。

设计二:

设计任务: 模拟文件系统的前端操作 shell

实现一个简单的 shell (命令行解释器)。

将设计一的管理程序 simdisk 作为后台进程运行，利用本设计任务的 shell 操作 simdisk。

本设计任务在于学会如何实现在前端的 shell 进程和后端的 simdisk 进程之间利用共享内存进行进程间通信 (IPC)。

设计三:

设计任务: 模拟文件系统的操作管理

实现多个进程同时对模拟文件系统进行操作。设计管理程序 simdisk 的用户访问权限管理。访问模拟文件系统的每个进程都属于某个用户，管理程序 simdisk 根据其访问权限决定其对模拟文件系统的操作。

对模拟文件系统的操作要求做到: 共享读，互斥写。

本设计任务在于学会如何实现信息的安全管理和进程同步。

注：要求从课程设计的整体来考虑设计任务一、二、三，并分阶段实现。

【实验环境】

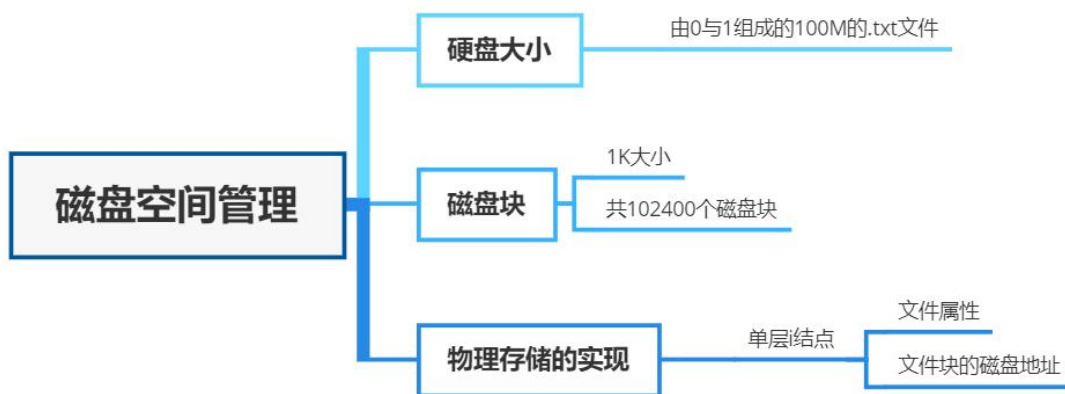
操作系统：Windows10

实验内容

一、 操作系统的架构

1.1 任务一：内核

1.1.1 磁盘空间管理

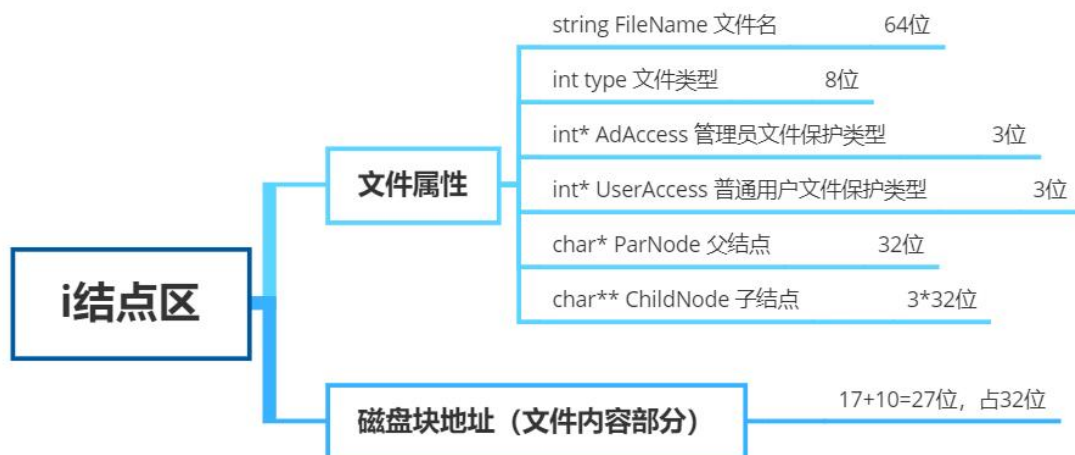


磁盘大小。采用仅包含 0 与 1 的.txt 文本文件作为磁盘。每一个 0 或 1 看作 1 比特，则 100M 的磁盘需 $100 \times 1024 \times 1024 = 104857600$ 比特。以含 104857600 个 0 的“HardDisk.txt”文件作为待初始化的磁盘。操作系统与硬盘之间的交互视作与该文本文件的交互。

磁盘块。磁盘块大小为 1K，由于磁盘的大小为 100M，则该硬盘包含 102400 个磁盘块，需要用 17 位二进制数表示。将磁盘块从 0 开始编址，则磁盘块的地址为 0~102399。将块内地址从 0 开始编制，块内地址表示为 0~1023，用 10 位二进制数表示。空闲的磁盘块采用位图进行管理。由于磁盘块有 102400 个，则位图中应该有 102400 位，占 100 个磁盘块。磁盘块地址+块内地址共去 27 位，故采用 **32 位地址**对文件系统进行索引。其中第 0 为表示该磁盘块是否被占用，第 1~4 位无意义，第 5~21 位表示磁盘块地址，第 22~31 位表示块内地址。

物理存储。采用单层的*i*结点对文件进行存储。对于目录文件，*i*结点中仅包含文件属性。对于普通文件，*i*结点中还包含文件内容块对应的地址。*i*结点为单层结构，不进行扩展。

*i*结点的架构如下所示：



1) 文件名。固定长度。采用 ASCII 码进行编码，64 位最多可以表示 8 个

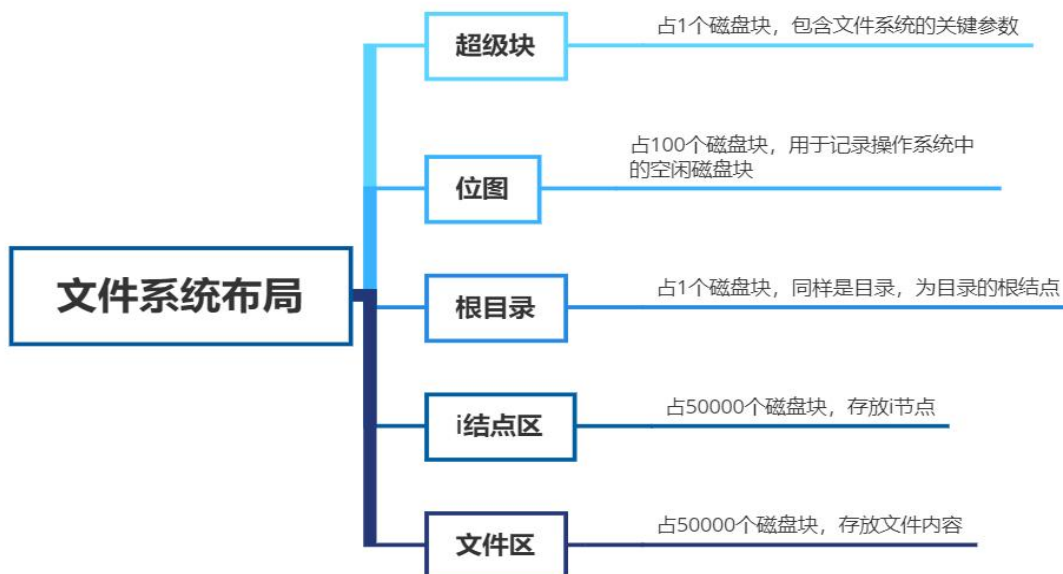
ASCII 字符。对于目录，其文件名长度最多为 8；对于普通文件，其完整文件名为文件名+‘.’+扩展名，其中.长度为 1，扩展名长度固定为 3，故文件名长度为 4。

- 2) 文件类型。分类目录文件与普通文件两种类型，其中 00000000 表示目录文件，00000001 表示普通文件。
- 3) 管理员文件保护类型。3 位分别表示“可读”、“可写”、“可执行”。0 表示“不可”，1 表示“可”。用于表示管理员用户能够对文件进行的操作。其中，对于目录文件，本文件系统不讨论其保护类型，三位均为 0 但不表示任何意义。
- 4) 普通用户文件保护类型。3 位分别表示“可读”、“可写”、“可执行”。0 表示“不可”，1 表示“可”。用于表示普通用户能够对文件进行的操作。其中，对于目录文件，本文件系统不讨论其保护类型，三位均为 0 但不表示任何意义。两种保护类型共采用 8 位进行表示，其中第 0、4 位置 0，无意义；第 1、2、3 位表示管理员文件保护类型；第 5、6、7 位表示普通用户文件保护类型。
- 5) 父结点。指向磁盘块的父结点。根目录的父结点为 00000000000000000000000000000000，但并不表示指向第 0 块磁盘块，而是表示没有父结点。
- 6) 子结点。指向磁盘块的子结点。子结点数目最多为 3。其中 0~31 位、32~63 位、64~93 位分别表示第 1、2、3 个子结点。
- 7) 磁盘块地址（文件内容部分）。指向普通文件的文件内容结点。

以上所有共 240 位，在一个磁盘块中能够放下，因此 i 结点的“文件属性”部分仅需一个磁盘块，而“文件内容”部分固定为一个磁盘块，也即一个文件最多保存 128 个 ASCII 字符。

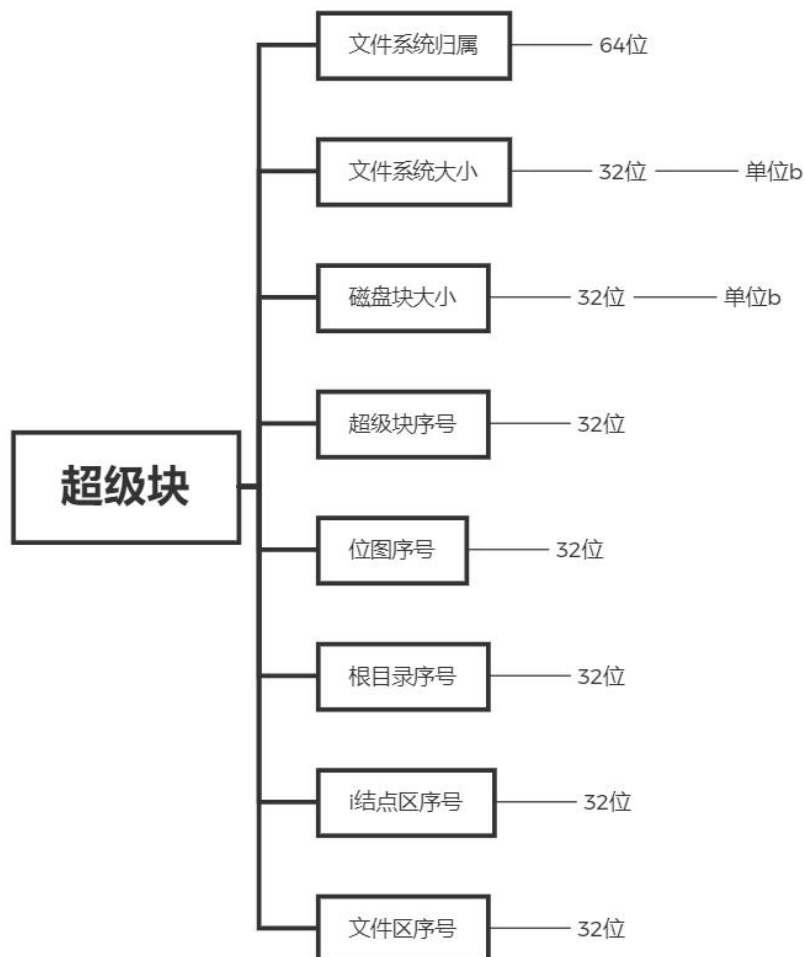
文件系统的索引。文件系统所有的节点以树的方式进行存储，树则通过静态链表的方式进行实现。文件系统的索引方式也即树的索引方式。文件系统索引的地址分为绝对地址和相对地址，其中绝对地址要求地址从根结点/cmy 开始，相对地址则默认仅在当前文件夹下执行操作。无法使用“.”或“..”命令。

1.1.2 磁盘布局



文件系统的布局总共分为超级块、位图、根目录、*i*结点区、文件区五个部分。

超级块。占 1 个磁盘块，磁盘块序号为 0，包含文件系统的关键信息。超级块所包含的信息如下：



文件系统归属，表示该文件系统归谁所有，占 64 位，采用 ASCII 码进行编码，因此文件系统归属最多可以表示 8 个 ASCII 字符。操作系统大小，为 104857600 为，用 32 位表示。超级块序号，为 0，表示超级块所在磁盘块的序号。位图序号，为 1，表示位图所在磁盘块的序号。根目录序号，表示根目录所在磁盘块的序号，为 101。*i*结点区序号，表示根目录所在*i*结点区的序号，为 102。文件区序号，表示文件区所在磁盘块的序号，为 50102。

位图，记录文件系统空闲磁盘块。0 表示磁盘块空闲，1 表示磁盘块被占用。

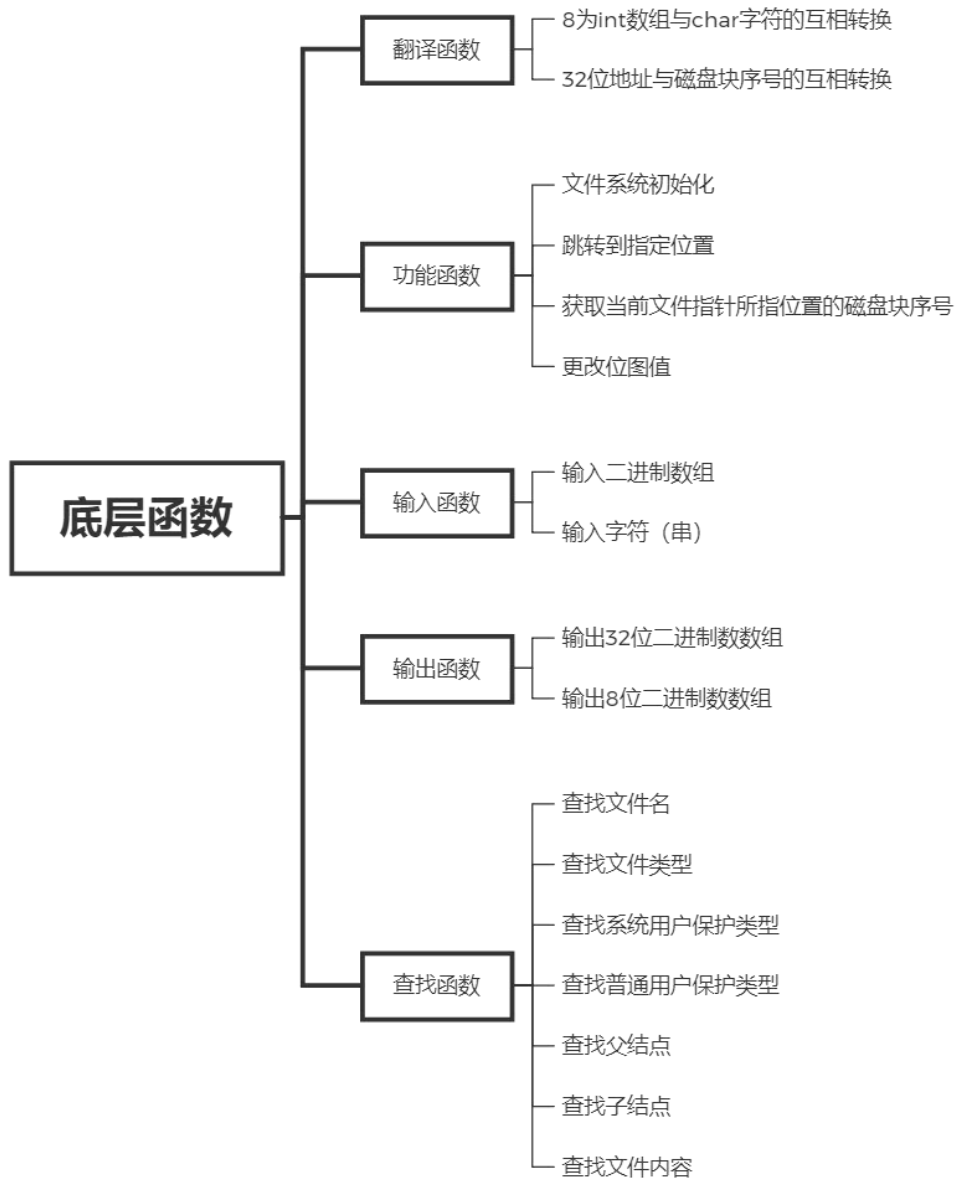
根目录。表示文件系统的起点。

*i*结点区。用于存放*i*结点。

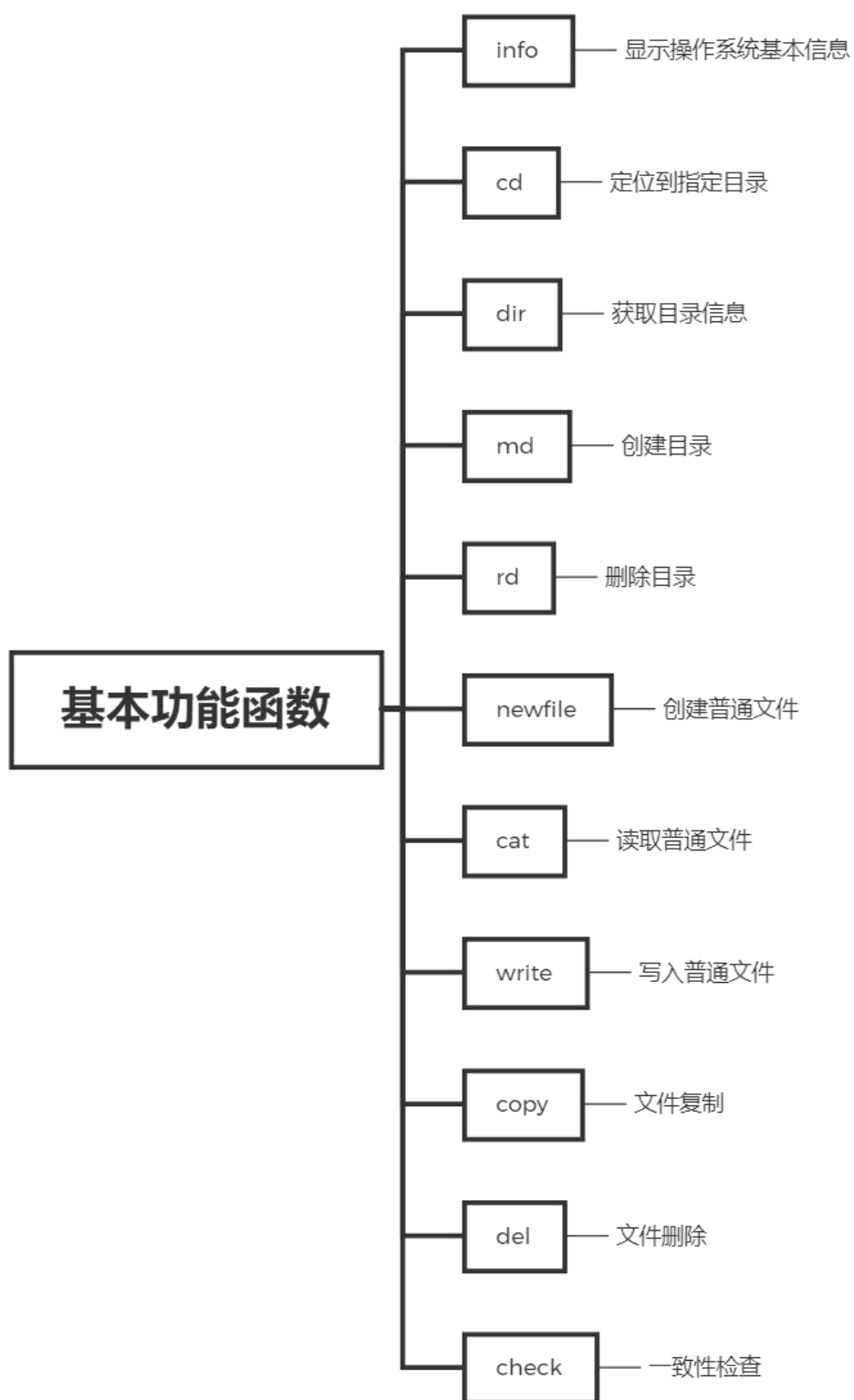
文件区。用于存放普通文件的具体内容。

1.1.3 功能实现

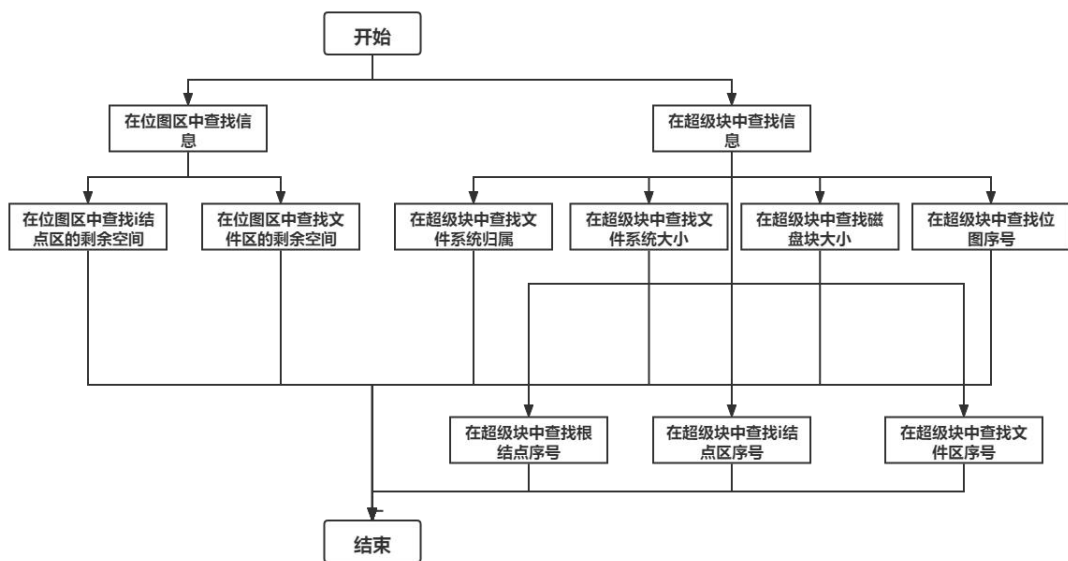
底层函数。底层函数为一个简单的、基础的函数，复杂的功能将由其组合而成。基本的底层函数如下所示：



基本功能。基本功能为文件系统所能够执行的一些基本的操作。基本功能函数如下所示：

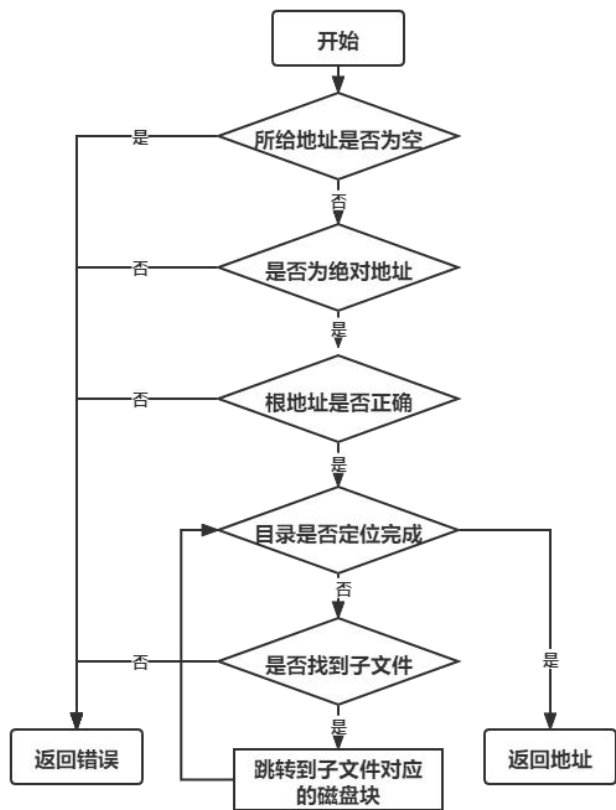


Info。Info 函数的功能流程图如下：



Info 函数需要返回给 Shell 文件系统的相关信息。Info 函数的输入格式为：info。信息分为两部分：关于文件系统中的剩余空间。通过查找位图获取*i*结点区以及文件区的剩余空间。超级块中的信息，通过检索超级块返回“文件系统归属”、“文件系统大小”、“磁盘块大小”、“位图序号”、“根结点序号”、“*i*结点区序号”、“文件区序号”的信息。

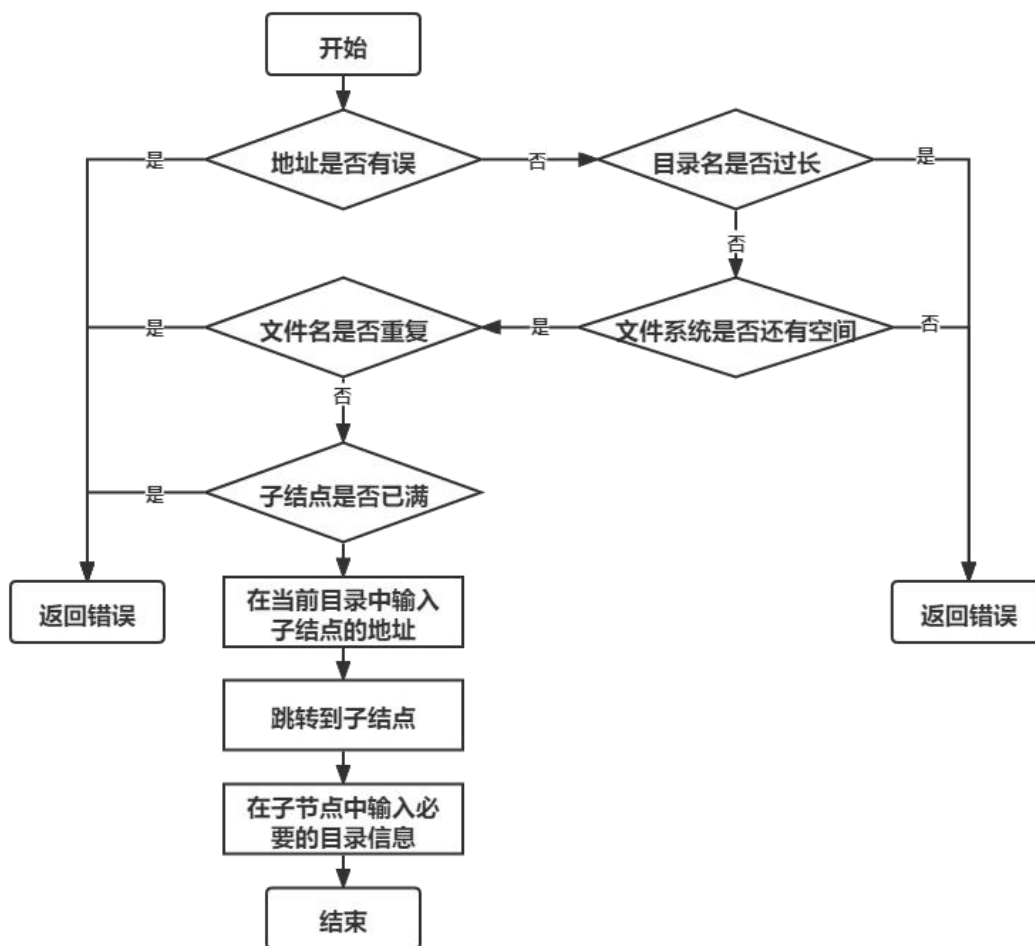
cd。cd 函数的功能流程图如下：



cd 函数的功能是将文件指针跳转到对应的位置，并返回绝对地址的信息给

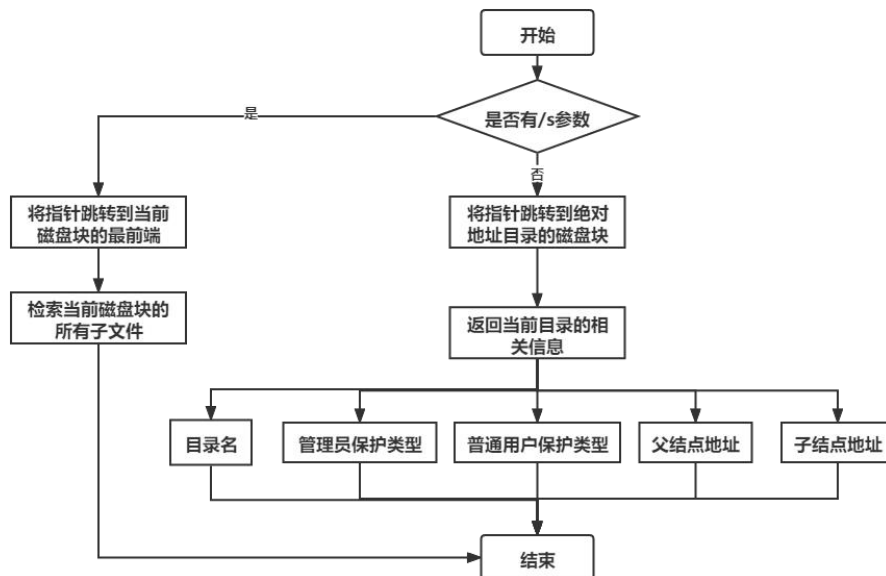
shell。dir 函数的输入格式为：cd+绝对地址。Cd 函数首先会对 shell 发送来的地址进行检查，判断其地址是否为空、是否为绝对地址、是否为根地址。然后根据所给地址在当前目录汇总不断地寻找对应的子文件并跳转到子文件对应的磁盘块。如果没有找到对应的子文件，则返回错误。当目录定位完成后，将绝对地址返回给 Shell，用于提示用户地址改变。

md。md 函数的功能流程图如下：



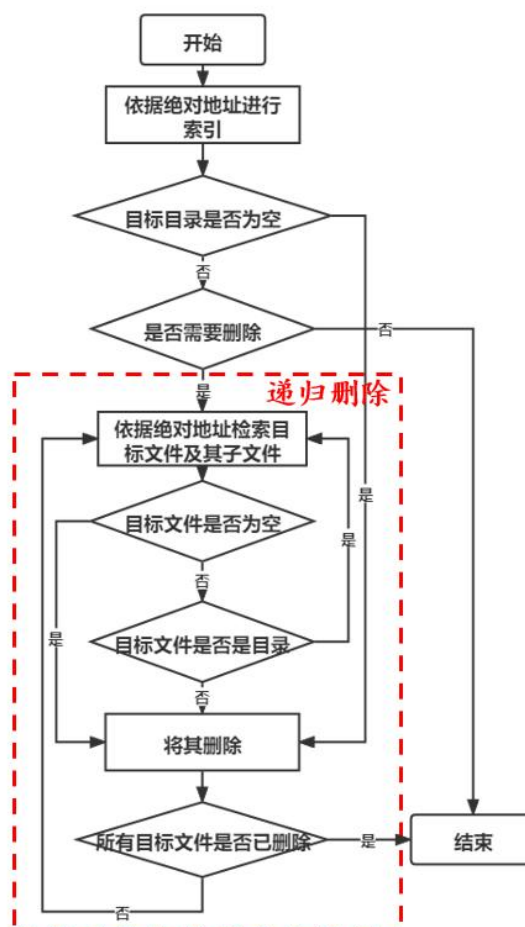
md 函数用于获取目录信息。md 函数的输入格式为：md+绝对地址或相对地址+目录名。在创建目录前，首先会进行一系列的判断，以确认该目录的创建是否满足条件。当判断为满足条件的时候，文件指针将跳转到对应的磁盘块并输入相应的信息，主要为文件名。

dir。dir 函数的功能流程图如下：



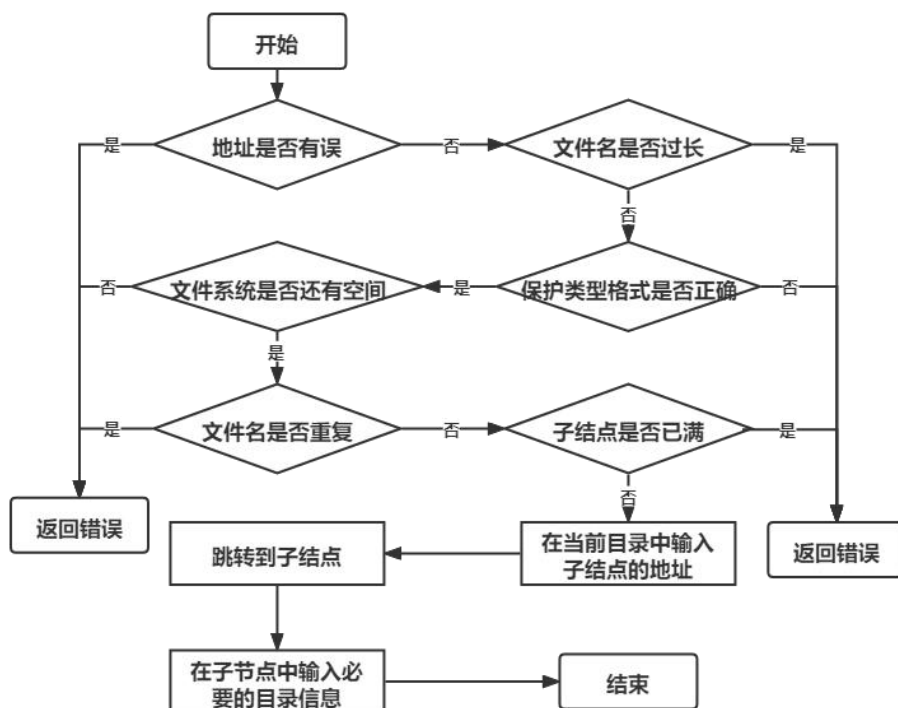
dir 函数用于获取目录信息。**dir** 函数的输入格式为：**dir(+/s)(+绝对地址)**。当带有/s 参数的时候，文件指针将读取当前文件的所有子文件并检索其文件名称用作返回,且/s 参数只能用于获取当前地址的信息。当不带有/s 参数的时候，文件指针将跳转到绝对地址对应的磁盘块并检索其相关信息。当不带绝对地址与/s 参数时，将返回当前目录的相关信息。

rd。**rd** 函数的功能流程图如下：



rd 函数用于删除某一个目录。rd 函数的输入格式为：rd +绝对地址。rd 函数采用类似于树的结点删除的递归删除方法。当一个文件是普通文件的时候，将其删除；当一个文件是目录文件且为空的时候，将其删除；当一个文件是目录文件且不空的时候，检索其子文件并执行上述判断。删除中要注意的地方是当一个磁盘块被清空后，对应的位图区域需要被置 0。

newfile。newfile 函数的功能流程图如下：



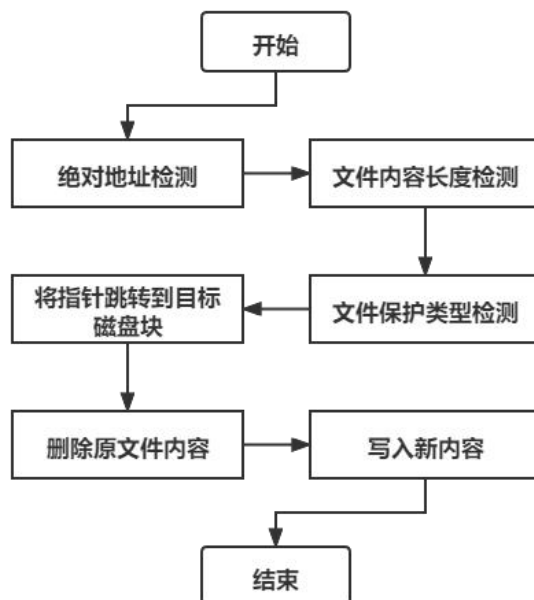
newfile 函数用于创建一个新的普通文件。newfile 函数的输入格式为：newfile +绝对地址或相对地址+文件保护类型+文件内容。其中文件保护类型为六位二进制整数，分别对应 3 位系统用户保护类型与 3 位普通用户保护类型。文件内容的长度不能超过 128。同时注意，文件创建成功后，需要在位图中对应位置置 1。

cat。cat 函数的功能流程图如下：



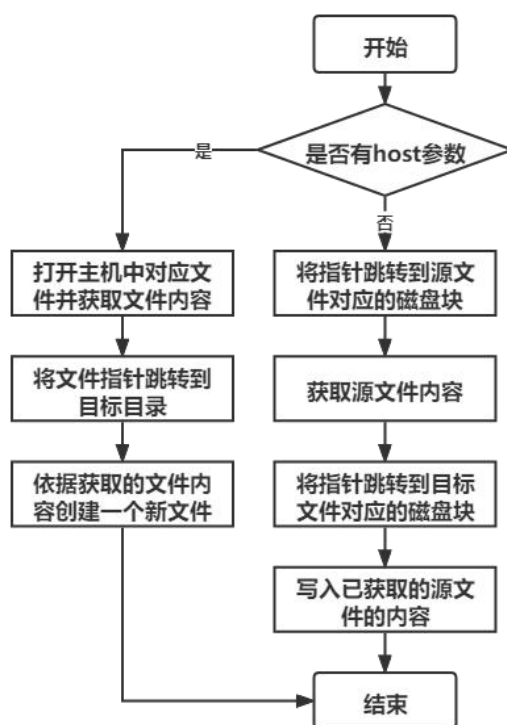
cat 函数用于删除某一个目录。cat 函数的输入格式为：cat +绝对地址。文件指针将定位到对应的磁盘块中并读取其中的二进制数将其翻译为 ASCII 字符，最后返回所读取的文件内容

write。write 函数的功能流程图如下：



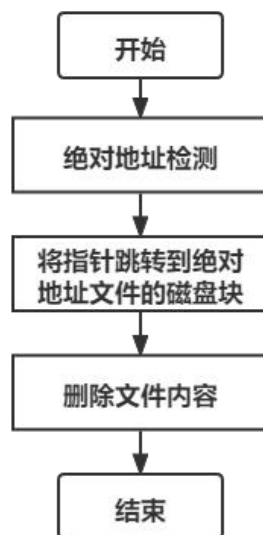
write 函数用于写入某一个文件。write 函数的输入格式为：write+绝对地址+文件内容。在完成地址和内容检测后，文件指针将跳转到对应的磁盘块位置，先删除所有的内容，再写入新的内容。对于保护类型检测，将判断该文件的对应保护类型与用户的身份，并判断该用户能否写入该文件。

copy。copy 函数的功能流程图如下：



copy 函数用于文件的复制。**copy** 函数的输入格式为：**copy** +源绝对地址+目标绝对地址(+host)。当有 **host** 参数的时候，将从主机中提取文件内容，并根据文件内容在指定目录创建一个新文件。这个新文件的访问类型默认为 111111。当没有 **host** 参数的时候，将源文件的内容复制到目标文件。

del。**del** 函数的功能流程图如下：



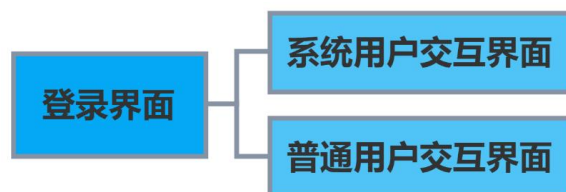
del 函数用于删除某一个目录。**del** 函数的输入格式为：**del** +绝对地址。文件指针首先将定位到对应的磁盘块，并写入 1024 个 0。同时应在位图中将相应的位置置 0

check。**Check** 函数用于检测文件的一致性。文件指针将从根结点开始遍历所有文件。每当指针访问一个文件的时候，都将检测该文件对应的位图位置是否正确，如果不正确，则将其修正。

1.2 任务二：Shell 与进程间通讯

1.2.1 Shell 的架构。

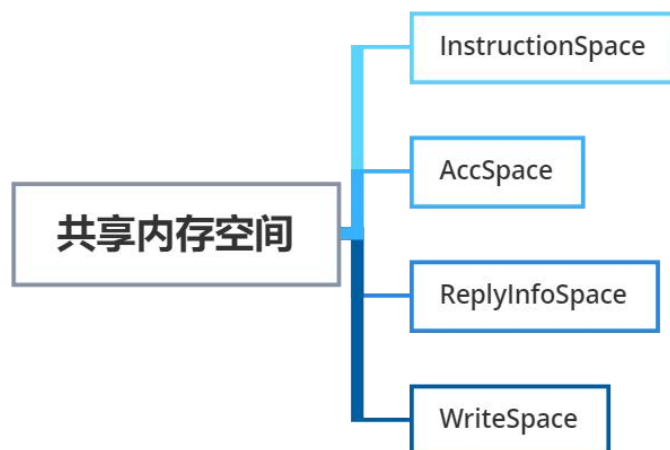
Shell 的基本架构如下：



交互界面将区分为普通用户交互界面与系统用户交互界面。两个界面表面上并无不同，但在进行进程间通讯时，两个界面将发送不同的信息以区分系统用户和普通用户。在登录界面中，用户需要输入账号与密码才可进入交互界面。不同的账号将用作区分系统用户与普通用户。

1.2.2 进程间通讯。

采用共享内存的方式进行进程间通讯。内存空间将由 **Shell** 或者 **Core** 进行创建。两者均可读取或写入内存空间中的内容，以实现进程间通讯。本文件中用到的内存共享空间如下：



InstructionSpace。由 Shell 创建，用于从 Shell 向 Simdisk 发送信息。

AccSpace。由 Shell 创建，用于从 Shell 向 Simdisk 发送用户身份信息以区分用户身份。

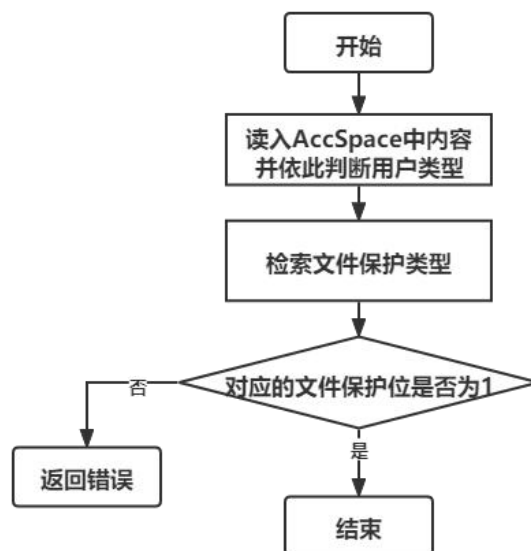
ReplyInfoSpace。由 Simdisk 创建，用于从 Core 向 Shell 回复信息。

WriteSpace。由 Simdisk 创建，用于实现互斥写。

1.3 任务三：安全管理与进程同步

1.3.1 安全管理

文件保护位的架构详见 1.1.1 3)、4)。由于此系统为文件系统，故不讨论程序的“可执行”问题，只讨论文件的读写问题。安全管理的一般流程图如下所示：



当用户请求对某个文件进行读写的时候，Simdisk 会根据提取 *AccSpace* 中内容以区分用户身份。其中内容为 ‘0’ 表示该用户为系统用户，内容为 ‘1’ 表示该用户为普通用户。根据用户身份与对应文件的保护类型，Simdisk 将决定用户是否能够访问目标文件，如果不能将返回错误信息。

1.3.2 进程同步

由于 Simdisk 为单线程，因此采用“宏观上并行，微观上串行”的实现方法模拟多道程序系统。对于文件的读取，不作限制，以此模拟“共享读”。对于文件的写入，在写入文件的时候，Simdisk 会将当前写入的文件的绝对地址放入 *WriteSpace* 中。如果此时存在 Shell 想要访问某一文件，会先判断 *WriteSpace* 中内容与需要写入的文件是否为同一文件，如果是，不可写入，并返回错误信息，以此模拟“互斥写”。

从宏观上看，对于一般的文件操作，Simdisk 都能够比较快地相应并给出返回信息，因此当多个程序同时对 Simdisk 进行访问时，Simdisk 在微观上串行执行，但是在宏观上并行执行。

二、 操作系统的实现

2.1 任务一：内核

2.1.1 底层函数的实现

由于所用函数众多，部分简单的函数将只给出其功能，不给出其具体实现，详细的实现见源代码及注释。

1) 翻译函数

```
int* Char2EightInt(char Char); //将符号转换为 8 位二进制整数数组  
char EightInt2Char(int* EightInt); //将 8 位二进制整数数组转换为符号  
int* BlockSqe2Add(int BlockSqe); //将磁盘块序号转换为 32 位地址  
int Add2BlockSqe(int* Add); //将 32 位地址转换为磁盘块序号
```

2) 功能函数

```
void Init(fstream& Disk_Pointer); //初始化函数，执行根目录、超级块区和位图区初始化
```



```

void Init(fstream& Disk_Pointer)
{
    /*初始化根目录*/
    Disk_Pointer.seekg(1024 * 101, ios::beg);
    char RootDirName[] = "cmy\0\0\0\0\0"; //根目录名
    InputCharArr(Disk_Pointer, RootDirName);
    int FileType[] = { 0, 0, 0, 0, 0, 0, 0, 0 }; //表示文件类型为目录
    InputIntArr(Disk_Pointer, FileType, 8);
    int Access[] = { 0, 0, 0, 0, 0, 0, 0, 0 }; //目录不讨论访问权限问题，全部置为0
    InputIntArr(Disk_Pointer, Access, 8);
    int ParNode[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    //根目录没有父结点
    InputIntArr(Disk_Pointer, ParNode, 32);
    int ChildNode[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    //根目录初始时没有子结点
    InputIntArr(Disk_Pointer, ChildNode, 96);
    //目录没有文件内容
    int DiskAddress[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    InputIntArr(Disk_Pointer, DiskAddress, 32);

    /*初始化超级块*/
    JumpPointer(Disk_Pointer, 0, 0);
    InputCharArr(Disk_Pointer, const_cast<char*>("cmy"));
    JumpPointer(Disk_Pointer, 0, 64);
    InputIntArr(Disk_Pointer, BlockSqe2Add(104357600), 32);
    JumpPointer(Disk_Pointer, 0, 96);
    InputIntArr(Disk_Pointer, BlockSqe2Add(1024), 32);
    JumpPointer(Disk_Pointer, 0, 124);
    InputIntArr(Disk_Pointer, BlockSqe2Add(0), 32);
    JumpPointer(Disk_Pointer, 0, 160);
    InputIntArr(Disk_Pointer, BlockSqe2Add(1), 32);
    JumpPointer(Disk_Pointer, 0, 192);
    InputIntArr(Disk_Pointer, BlockSqe2Add(101), 32);
    JumpPointer(Disk_Pointer, 0, 224);
    InputIntArr(Disk_Pointer, BlockSqe2Add(102), 32);
    JumpPointer(Disk_Pointer, 0, 256);
    InputIntArr(Disk_Pointer, BlockSqe2Add(50102), 32);

    /*初始化位图区*/
    JumpPointer(Disk_Pointer, 1, 0); //跳转到位图区
    for (int i = 0; i < 102; i++) //将前101个磁盘块置为已使用
    {
        Disk_Pointer << 1;
    }
    for (int i = 0; i < 1024 * 100 - 102; i++) //将后面的所有磁盘块置为未使用
    {
        Disk_Pointer << 0;
    }
}

```

int GetBlockSqe(fstream& Disk_Pointer); //获取对应指针的当前磁盘块序号

int MatchDocName(fstream& Disk_Pointer, int BlockSqe, string DocName); //将对应的文件名与特定磁盘块中的所有子结点的文件名进行比较，如果匹配，返回子结点对应的磁盘块序号；如果不匹配，返回 0

```

int MatchDocName(fstream& Disk_Pointer, int BlockSqe, string DocName)
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    for (int i = 0; i < 3; i++) //循环3次判断子结点文件名
    {
        JumpPointer(Disk_Pointer, BlockSqe, 112+32*i); //跳转到子结点部分
        int* BitDocName = Output32Bit(Disk_Pointer); //读入32位子结点地址
        char* CharDocName = FindDocName(Disk_Pointer, Add2BlockSqe(BitDocName)); //寻找子结点文件名
        if (string(CharDocName) == DocName) //判断子结点文件名是不是要找的东西
        {
            JumpPointer(Disk_Pointer, OriSqe, 0);
            return Add2BlockSqe(BitDocName); //如果是，返回子结点对应的磁盘块号
        }
    }
    JumpPointer(Disk_Pointer, OriSqe, 0);
    return 0; //如果不是，返回0
}

```

void JumpPointer(fstream& Disk_Pointer, int BlockSqe, int BlockIn);//将指针跳转到指定位置。其中 BlockSqe 为磁盘块序号, BlockIn 为块内地址

void BitMapChange(fstream& Disk_Pointer, int BlockSqe, int Tag);//将位图区域的某一个值置 1 或者置 0 输入函数

```
void JumpPointer(fstream& Disk_Pointer, int BlockSqe, int BlockIn)
{
    Disk_Pointer.seekg(1024 * BlockSqe + BlockIn, ios::beg);
}
```

3) 输入函数

void InputIntArr(fstream& Disk_Pointer, int* IntArr, int Length);//直接将二进制数输入到硬盘中

void InputChar(fstream& Disk_Pointer, char Char);//将符号转换为二进制数并输入到硬盘中

```
void InputChar(fstream& Disk_Pointer, char Char)
{
    int* InputInt;
    InputInt = new int[8];
    InputInt = Char2EightInt(Char);
    InputIntArr(Disk_Pointer, InputInt, 8);
}
```

void InputCharArr(fstream& Disk_Pointer, char* CharArr);//将符号串转换为二进制数并输入到硬盘中

void InputDocName(fstream& Disk_Pointer, int BlockSqe, string DocName);//将文件名输入到硬盘中, 输入完后, 指针回到原始磁盘块的首位。

```
void InputDocName(fstream& Disk_Pointer, int BlockSqe, string DocName)
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    JumpPointer(Disk_Pointer, BlockSqe, 0);
    int DocNameLength = DocName.length(); //获取文件名的长度
    char* CharDocName;
    CharDocName = new char[DocNameLength]; //申请一个临时数组存放文件名
    strcpy_s(CharDocName, DocNameLength+1, DocName.c_str()); //将文件名汇总内容拷贝到临时数组
    InputCharArr(Disk_Pointer, CharDocName); //输入文件名
    JumpPointer(Disk_Pointer, OriSqe, 0);
}
```

void InputParNode(fstream& Disk_Pointer, int BlockSqe, int* IntArr);//将父指针输入到硬盘中, 输入完后, 指针回到原始磁盘块的首位。

void InputAcc(fstream& Disk_Pointer, int BlockSqe, string Acc);//将保护类型输入到硬盘中

void InputFileContext(fstream& Disk_Pointer, int BlockSqe, int NewDocBlockSqe, string Context);//在对应的磁盘块中输入文件内容磁盘块对应的地址,并在对应的文件磁盘块中输入文件内容

```

void InputFileContext(fstream& Disk_Pointer, int BlockSqe, int NewDocBlockSqe, string Context)
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    JumpPointer(Disk_Pointer, BlockSqe, 208);
    InputIntArr(Disk_Pointer, BlockSqe2Add(NewDocBlockSqe), 32);
    JumpPointer(Disk_Pointer, NewDocBlockSqe, 0);
    InputCharArr(Disk_Pointer, const_cast<char*>(Context.c_str()));
    JumpPointer(Disk_Pointer, OriSqe, 0);
}

```

void InputFileType(fstream& Disk_Pointer, int BlockSqe, int FileType);//将文件类型输入到磁盘中

4) 输出函数

输出函数巴拉巴拉

int* Output32Bit(fstream& Disk_Pointer);//输出 32 位二进制数

int* Output8Bit(fstream& Disk_Pointer);//输出 8 位二进制数

5) 查找函数

int FindFreeAreaInNode(fstream& Disk_Pointer);//用于寻找位图中对应的 i 结点的区域的空闲区域，返回 i 结点的磁盘块序号，查找完后，指针回到原始磁盘块的首位。

int FindFreeAreaInDoc(fstream& Disk_Pointer);//用于寻找位图中对应的文件区域的空闲区域，返回文件块对应的的磁盘块序号，查找完后，指针回到原始磁盘块的首位

int FindFreeChildNode(fstream& Disk_Pointer, int BlockSqe);//判断特定的磁盘块中是否有空闲的子结点，如果有，返回子结点序号 (1/2/3)；如果没有，返回 0 并定位到磁盘块开头

```

int FindFreeChildNode(fstream& Disk_Pointer, int BlockSqe)
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    char IsFree;

    JumpPointer(Disk_Pointer, BlockSqe, 112);//跳转到第一个子结点的位置
    Disk_Pointer >> IsFree;
    if (IsFree == '0')//如果子结点没有被使用
    {
        Disk_Pointer.seekg(-1, ios::cur);//退一位
        JumpPointer(Disk_Pointer, OriSqe, 0);
        return 1;//返回子结点序号
    }

    JumpPointer(Disk_Pointer, BlockSqe, 112 + 32);//跳转到第二个子结点的位置
    Disk_Pointer >> IsFree;
    if (IsFree == '0')//如果子结点没有被使用
    {
        Disk_Pointer.seekg(-1, ios::cur);//退一位
        JumpPointer(Disk_Pointer, OriSqe, 0);
        return 2;//返回子结点序号
    }

    JumpPointer(Disk_Pointer, BlockSqe, 112 + 32 + 32);//跳转到第三个子结点的位置
    Disk_Pointer >> IsFree;
    if (IsFree == '0')//如果子结点没有被使用
    {
        Disk_Pointer.seekg(-1, ios::cur);//退一位
        JumpPointer(Disk_Pointer, OriSqe, 0);
        return 3;//返回子结点序号
    }

    JumpPointer(Disk_Pointer, OriSqe, 0);
    return 0;
}

```

char* FindDocName(fstream& Disk_Pointer, int BlockSqe);//寻找某一 32 位地址对应的文件磁盘块对应的文件名，返回文件名数组，查找完后，指针回到原始磁盘块的首位

```
char* FindDocName(fstream& Disk_Pointer, int BlockSqe)
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    char* DocName; //应该返回的文件名
    DocName = new char[8];
    int* TempIntArr; //用于装载用于翻译的数字
    JumpPointer(Disk_Pointer, BlockSqe, 0); //跳转到对应位置
    for (int i = 0; i < 8; i++)
    {
        TempIntArr = Output8Bit(Disk_Pointer);
        DocName[i] = EightInt2Char(TempIntArr); //将对应的二进制数组转换为符号
    }
    JumpPointer(Disk_Pointer, OriSqe, 0);
    return DocName; //返回文件名
}
```

int FindDocType(fstream& Disk_Pointer, int BlockSqe);//寻找某个磁盘块对应的文件类型

int* FindDocSysAcc(fstream& Disk_Pointer, int BlockSqe);//寻找某个磁盘块对应的系统保护类型

int* FindDocComAcc(fstream& Disk_Pointer, int BlockSqe);//寻找某个磁盘块对应的普通用户保护类型

int* FindDocParNodeAdd(fstream& Disk_Pointer, int BlockSqe);//寻找某个磁盘块对应的父结点地址

int** FindDocChildNodeAdd(fstream& Disk_Pointer, int BlockSqe);//寻找某个磁盘块对应的子结点地址

int* FindDocContextNode(fstream& Disk_Pointer, int BlockSqe);//寻找某个磁盘块对应的内容的地址

2.1.2 基本功能函数的实现

对于基本功能函数，按照架构进行直线，只展示其主函数部分的内容，关于其调用了的其他函数的实现，详细参见源代码。

1) Info


```

else if (InstructionBuf == "info")
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    string ReplyInfo = "";
    string DocSysHost;
    string DocSysSize;
    string DiskBlockSize;
    string SuperBlockSize;
    string BitMapSqe;
    string RootSqe;
    string INodeSqe;
    string DocSqe;
    DocSysHost = FindDocName(Disk_Pointer, 0);
    JumpPointer(Disk_Pointer, 0, 64);
    DocSysSize = to_string(Add2BlockSqe(Output32Bit(Disk_Pointer)));
    JumpPointer(Disk_Pointer, 0, 96);
    DiskBlockSize = to_string(Add2BlockSqe(Output32Bit(Disk_Pointer)));
    JumpPointer(Disk_Pointer, 0, 128);
    SuperBlockSize = to_string(Add2BlockSqe(Output32Bit(Disk_Pointer)));
    JumpPointer(Disk_Pointer, 0, 160);
    BitMapSqe = to_string(Add2BlockSqe(Output32Bit(Disk_Pointer)));
    JumpPointer(Disk_Pointer, 0, 192);
    RootSqe = to_string(Add2BlockSqe(Output32Bit(Disk_Pointer)));
    JumpPointer(Disk_Pointer, 0, 224);
    INodeSqe = to_string(Add2BlockSqe(Output32Bit(Disk_Pointer)));
    JumpPointer(Disk_Pointer, 0, 256);
    DocSqe = to_string(Add2BlockSqe(Output32Bit(Disk_Pointer)));
    ReplyInfo = ReplyInfo + "文件系统归属:" + DocSysHost + "\n";
    ReplyInfo = ReplyInfo + "文件系统大小:" + DocSysSize + "\n";
    ReplyInfo = ReplyInfo + "磁盘块大小:" + DiskBlockSize + "\n";
    ReplyInfo = ReplyInfo + "超级块大小:" + SuperBlockSize + "\n";
    ReplyInfo = ReplyInfo + "位图序号:" + BitMapSqe + "\n";
    ReplyInfo = ReplyInfo + "根结点序号:" + RootSqe + "\n";
    ReplyInfo = ReplyInfo + "I结点区序号:" + INodeSqe + "\n";
    ReplyInfo = ReplyInfo + "I结点区剩余空间:" + to_string(GetINodeRemain(Disk_Pointer)) + "\n";
    ReplyInfo = ReplyInfo + "文件区序号:" + DocSqe + "\n";
    ReplyInfo = ReplyInfo + "文件区剩余空间:" + to_string(GetDocRemain(Disk_Pointer));
    JumpPointer(Disk_Pointer, OriSqe, 0);
    Reply(ReplyInfo, ReplyInfoSend);
    continue;
}

```

首先定义字符串变量用于存储信息，然后将磁盘块定位到超级块中，通过输出函数 Output32bit 与翻译函数 Add2BlockSqe 获取超级块内的信息并存入字符串变量，最后返回所获取的信息。

2) Cd

```

else if (InstructionBuf == "cd")
{
    string cdBuf; //用户存储输入内容中的地址部分
    Buf >> cdBuf;
    string ReplyInfo = cdAddLocation(Disk_Pointer, cdBuf);
    Reply(ReplyInfo, ReplyInfoSend);
}

```

直接调用 cdAddLocation 函数进行文件指针跳转并返回相关信息。

3) Dir

```

else if (InstructionBuf == "dir")
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    string ReplyInfo="";
    string dirBuf;//用户存储输入内容中的地址部分
    Buf >> dirBuf;
    if (dirBuf == "/s")//如果带/s参数
    {
        ReplyInfo += "当前目录下的所有文件和子目录为：";
        ShowAllChildNodeName(Disk_Pointer, GetBlockSqe(Disk_Pointer), ReplyInfo);
        Reply(ReplyInfo, ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
    int CurBlockSqe=dirAddLocation(Disk_Pointer, dirBuf);//文件指针定位
    if (CurBlockSqe == 0)//如果返回错误信息
    {
        Reply("地址有误，创建失败！", ReplyInfoSend);
        continue;
    }
    ReplyInfo += "当前目录下的信息为：";
    ShowAllInfo(Disk_Pointer, CurBlockSqe, ReplyInfo);
    Reply(ReplyInfo, ReplyInfoSend);
    continue;

    JumpPointer(Disk_Pointer, OriSqe, 0);
}

```

在主函数部分，若带/s 参数，调用 ShowAllChildNodeName 函数返回当前结点的所有子结点的名称。若不带/s 参数，调用 ShowAllInfo 函数返回当前结点的所有信息。

4) Md

```
else if (InstructionBuf == "md")//创建新目录
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    string mdBuf;//用户存储输入内容中的地址部分
    Buf >> mdBuf;

    string CurDirName=mdAddLocation(Disk_Pointer, mdBuf);
    if (CurDirName == "")
    {
        Reply("地址有误，创建失败！", ReplyInfoSend);
        continue;
    }

    /*判断新建目录的长度*/
    if (CurDirName.length() > 8)//目录名不能大于8位
    {
        Reply("目录名过长，创建失败！", ReplyInfoSend);
        continue;
    }

    int NewINodeBlockSqe = FindFreeAreaInINode(Disk_Pointer);

    /*判断文件系统是否还有空间*/
    if (!NewINodeBlockSqe)//如果文件系统汇总没有空间，创建失败
    {
        Reply("文件系统已无空闲空间，创建失败！", ReplyInfoSend);
        continue;
    }

    /*判断所创建文件名是否与子结点中文件名重复*/
    if (MatchDocName(Disk_Pointer, GetBlockSqe(Disk_Pointer), CurDirName))
    {
        Reply("文件名重复，创建失败！", ReplyInfoSend);
        continue;
    }

    /*判断子结点是不是满了*/
    int FreeChildSqe = FindFreeChildNode(Disk_Pointer, GetBlockSqe(Disk_Pointer));
    if (FreeChildSqe)//没满，将子结点标记为已经使用，并输入32位地址
    {
        JumpPointer(Disk_Pointer, GetBlockSqe(Disk_Pointer), 112 + 32 * (FreeChildSqe - 1));//跳转到对应的子结点
        int* INode = BlockSqe2Add(NewINodeBlockSqe);
        INode[0] = 1;//将首位置为1，表示该子结点已经被使用
        InputIntArr(Disk_Pointer, INode, 32);//输入32位子结点
        JumpPointer(Disk_Pointer, GetBlockSqe(Disk_Pointer), 0);//跳回开头
    }
    else
    {
        Reply("当前文件的子文件已达上限，创建失败！", ReplyInfoSend);
        continue;
    }

    /*对新文件进行初始化*/
    InputDocName(Disk_Pointer, NewINodeBlockSqe, CurDirName);
    InputParNode(Disk_Pointer, NewINodeBlockSqe, BlockSqe2Add(GetBlockSqe(Disk_Pointer)));
    BitMapChange(Disk_Pointer, NewINodeBlockSqe, 1);
    Reply("目录创建成功！", ReplyInfoSend);
    JumpPointer(Disk_Pointer, OriSqe, 0);
    continue;
}
```

其中 mdAddLocation 函数用过当前的地址返回需要被创建的目录名。

5) Rd

```
else if (InstructionBuf == "rd")
{
    rdOriSqe = GetBlockSqe(Disk_Pointer);
    Buf >> rdBuf;
    rdTarSqe=rdAddLocation(Disk_Pointer, rdBuf);//寻找应该删除的目标磁盘块
    if (rdTarSqe)
    {
        if (!IsDirEmpty(Disk_Pointer, rdTarSqe))//如果非空
        {
            rdIsNotEmpty = 1;
            Reply("目标目录非空，是否需要删除？y/n", ReplyInfoSend);
            continue;
        }
        else//如果是空的
        {
            rdDir(Disk_Pointer, rdTarSqe);//删除对应结点
            JumpPointer(Disk_Pointer, rdOriSqe, 0);
            rdIsNotEmpty = 0;//将所有系数置0
            rdOriSqe = 0;
            rdBuf = "";
            rdTarSqe = 0;
            BitMapChange(Disk_Pointer, rdTarSqe, 0);
            Reply("目标目录删除完成", ReplyInfoSend);
            continue;
        }
    }
    else//如果没有找到目标磁盘块，返回错误信息
    {
        Reply("地址错误！", ReplyInfoSend);
        JumpPointer(Disk_Pointer, rdOriSqe, 0);
        continue;
    }
    JumpPointer(Disk_Pointer, rdOriSqe, 0);
}
```

其中 rdAddLocation 根据输入的地址返回应该删除的目标磁盘块序号。

6) Newfile

```
else if (InstructionBuf == "newfile")
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    string newfileNameBuf://用户存储输入内容中的地址部分
    string newfileAccBuf;
    string newfileContextBuf;
    Buf >> newfileNameBuf;
    Buf >> newfileAccBuf;
    Buf >> newfileContextBuf;

    string CurDirName = mdAddLocation(Disk_Pointer, newfileNameBuf);
    /*文件名格式检测*/
    string ReplyInfo = FileNameTest(CurDirName);
    if (ReplyInfo != "")
    {
        Reply(ReplyInfo, ReplyInfoSend);
        continue;
    }

    /*保护类型格式检测*/
    ReplyInfo = FileAccTest(newfileAccBuf);
    if (ReplyInfo != "")
    {
        Reply(ReplyInfo, ReplyInfoSend);
        continue;
    }

    int NewINodeBlockSqe = FindFreeAreaInINode(Disk_Pointer);
    int NewDocBlockSqe = FindFreeAreaInDoc(Disk_Pointer);

    /*判断文件系统是否还有空间*/
    if (!NewINodeBlockSqe)//如果文件系统汇总没有空间，创建失败
    {
        Reply("文件系统已无空闲空间，创建失败！", ReplyInfoSend);
        continue;
    }
    if (!NewDocBlockSqe)//如果文件系统汇总没有空间，创建失败
    {
        Reply("文件系统已无空闲空间，创建失败！", ReplyInfoSend);
        continue;
    }

    /*判断所创建文件名是否与子结点中文件名重复*/
    if (MatchDocName(Disk_Pointer, GetBlockSqe(Disk_Pointer), CurDirName))
    {
        Reply("文件名重复，创建失败！", ReplyInfoSend);
        continue;
    }

    /*判断子结点是不是满了*/
    int FreeChildSqe = FindFreeChildNode(Disk_Pointer, GetBlockSqe(Disk_Pointer));
    if (FreeChildSqe)//没满，将子结点标记为已经使用，并输入32位地址
    {
        JumpPointer(Disk_Pointer, GetBlockSqe(Disk_Pointer), 112 + 32 * (FreeChildSqe - 1));//跳转到对应的子结点
        int* INode = BlockSqe2Add(NewINodeBlockSqe);
        INode[0] = 1;//将首位置为1，表示该子结点已经被使用
        InputIntArr(Disk_Pointer, INode, 32);//输入32位子结点
        JumpPointer(Disk_Pointer, GetBlockSqe(Disk_Pointer), 0);//跳回开头
    }
    else
    {
        Reply("当前文件的子文件已达上限，创建失败！", ReplyInfoSend);
        continue;
    }

    /*对新文件进行初始化*/
    InputDocName(Disk_Pointer, NewINodeBlockSqe, CurDirName);//输入文件名
    InputParNode(Disk_Pointer, NewINodeBlockSqe, BlockSqe2Add(GetBlockSqe(Disk_Pointer)));//输入父结点地址
    InputAcc(Disk_Pointer, NewINodeBlockSqe, newfileAccBuf);//输入文件保护类型
    InputFileContext(Disk_Pointer, NewINodeBlockSqe, NewDocBlockSqe, newfileContextBuf);//输入文件内容
    InputFileType(Disk_Pointer, NewINodeBlockSqe, 1);//输入文件类型
    BitMapChange(Disk_Pointer, NewINodeBlockSqe, 1);
    BitMapChange(Disk_Pointer, NewDocBlockSqe, 1);
    Reply("文件创建成功！", ReplyInfoSend);
    JumpPointer(Disk_Pointer, OriSqe, 0);
    continue;
}
```

7) Cat

```
else if (InstructionBuf == "cat")
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    string mdBuf;//用户存储输入内容中的地址部分
    Buf >> mdBuf;
    int TarSqe=catAddLocation(Disk_Pointer, mdBuf);
    if (TarSqe)
    {
        int* ComAcc = FindDocComAcc(Disk_Pointer, TarSqe);//查询普通用户访问权限
        int* SysAcc = FindDocSysAcc(Disk_Pointer, TarSqe);//查询系统用户访问权限
        if (!((ComAcc[0] == 1 && UserAccBuf[0] == '1') || (SysAcc[0] == 1 && UserAccBuf[0] == '0')))//判断有无权限写入该文件
        {
            Reply("无权限读取该文件!", ReplyInfoSend);
            JumpPointer(Disk_Pointer, OriSqe, 0);
            WriteLock = 0;//开锁
            continue;
        }
        Reply(GetFileContext(Disk_Pointer, Add2BlockSqe(FindDocContextNode(Disk_Pointer, TarSqe))), ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
    else
    {
        Reply("地址错误!", ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
    JumpPointer(Disk_Pointer, OriSqe, 0);
}
```

8) Write

```
else if (InstructionBuf == "write")
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    string writeBuf;//用户存储输入内容中的地址部分
    string InputInfo;//待输入的文件内容
    Buf >> writeBuf;
    Buf >> InputInfo;
    int TarSqe=catAddLocation(Disk_Pointer, writeBuf);
    if (TarSqe)
    {
        int* ComAcc = FindDocComAcc(Disk_Pointer, TarSqe);//查询普通用户访问权限
        int* SysAcc = FindDocSysAcc(Disk_Pointer, TarSqe);//查询系统用户访问权限
        if (InputInfo.length() > 128)
        {
            Reply("文件内容过多, 无法写入!", ReplyInfoSend);
            JumpPointer(Disk_Pointer, OriSqe, 0);
            continue;
        }
        if (!((ComAcc[1] == 1 && UserAccBuf[0] == '1') || (SysAcc[1] == 1 && UserAccBuf[0] == '0')))//判断有无权限写入该文件
        {
            Reply("无权限写入该文件!", ReplyInfoSend);
            JumpPointer(Disk_Pointer, OriSqe, 0);
            continue;
        }
        /*清空源文件*/
        JumpPointer(Disk_Pointer, Add2BlockSqe(FindDocContextNode(Disk_Pointer, TarSqe)), 0);
        for (int i = 0; i < 1024; i++)
        {
            Disk_Pointer << 0;
        }
        JumpPointer(Disk_Pointer, Add2BlockSqe(FindDocContextNode(Disk_Pointer, TarSqe)), 0);

        /*写时延迟*/
        strcpy_s((char*)WriteSend, writeBuf.length() + 1, const_cast<char*>(writeBuf.c_str()));
        ShareMemoryClear(WriteSend);

        /*写入*/
        InputCharArr(Disk_Pointer, const_cast<char*>(InputInfo.c_str()));
        Reply("写入成功!", ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
    else
    {
        Reply("地址错误!", ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
    JumpPointer(Disk_Pointer, OriSqe, 0);
}
```

9) Copy

```

else if (InstructionBuf == "copy")
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    string OriAdd;
    string TarAdd;
    string Parameter;

    Buf >> OriAdd;
    Buf >> TarAdd;
    Buf >> Parameter;

    if (Parameter == "")//如果没有参数
    {
        /*寻找源文件内容*/
        int OriAddSqe = catAddLocation(Disk_Pointer, OriAdd);
        string OriContext;
        if (OriAddSqe)
        {
            OriContext = GetFileContext(Disk_Pointer, Add2BlockSqe(FindDocContextNode(Disk_Pointer, OriAddSqe)));
        }
        else
        {
            Reply("源地址错误!", ReplyInfoSend);
            JumpPointer(Disk_Pointer, OriSqe, 0);
            continue;
        }

        int TarAddSqe = catAddLocation(Disk_Pointer, TarAdd);
        if (TarAddSqe)
        {
            int* TarAddDocAdd = FindDocContextNode(Disk_Pointer, TarAddSqe);
            JumpPointer(Disk_Pointer, Add2BlockSqe(TarAddDocAdd), 0);
            for (int i = 0; i < 1024; i++)
            {
                Disk_Pointer << 0;
            }
            JumpPointer(Disk_Pointer, Add2BlockSqe(TarAddDocAdd), 0);
            InputCharArr(Disk_Pointer, const_cast<char*>(OriContext.c_str()));
            Reply("复制成功!", ReplyInfoSend);
            JumpPointer(Disk_Pointer, OriSqe, 0);
            continue;
        }
        else
        {
            Reply("目标地址错误!", ReplyInfoSend);
            JumpPointer(Disk_Pointer, OriSqe, 0);
            continue;
        }
    }

    else if (Parameter == "host")
    {
        string TempOriContext;
        string OriContext;
        string OriFileName = GetWindowsFileName(OriAdd);
        ifstream Disk_Pointer2(OriAdd, ios::in || ios::out);

        /*寻找源文件内容*/
        if (Disk_Pointer2)
        {
            while (!Disk_Pointer2.eof())
            {
                Disk_Pointer2 >> TempOriContext;
                OriContext += TempOriContext;
            }
            //GetWindowsFileName(OriAdd, OriFileName)
        }
        else
        {
            Reply("源地址错误!", ReplyInfoSend);
            JumpPointer(Disk_Pointer, OriSqe, 0);
            continue;
        }
        Disk_Pointer2.close();

        /*拷贝到目标文件夹*/
        string InputInstruction = "";
        InputInstruction += OriFileName;
        InputInstruction += " 111111 ";
        InputInstruction += OriContext;
        istringstream Inputsstr(InputInstruction);
        Reply(CopyNewFile(Disk_Pointer, Inputsstr), ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
    else
    {
        Reply("参数错误!", ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
}

JumpPointer(Disk_Pointer, OriSqe, 0);
}

```

10) Del

```
else if (InstructionBuf == "del")
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    string delBuf; //用户存储输入内容中的地址部分
    Buf >> delBuf;
    int TarSqe = catAddLocation(Disk_Pointer, delBuf);
    if (TarSqe)
    {
        delDoc(Disk_Pointer, TarSqe);
        Reply("删除成功!", ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
    else
    {
        Reply("地址错误!", ReplyInfoSend);
        JumpPointer(Disk_Pointer, OriSqe, 0);
        continue;
    }
}
```

11) Check

```
else if (InstructionBuf == "check")
{
    int OriSqe = GetBlockSqe(Disk_Pointer);
    ConsistencyCheck(Disk_Pointer, 101);
    JumpPointer(Disk_Pointer, OriSqe, 0);
    Reply("文件一致性检查完成!", ReplyInfoSend);
    continue;
}
```

2.2 任务二：Shell 与进程间通讯

2.2.1 Shell 的实现

Shell 通过 easyX 图形库进行实现，下面对一些基本组件的实现进行举例分析：

1) 文本框。

文本框的实现采用矩形边框+特定位置文件输出的方式实现。首先选定一个位置并设置一个矩形边框：

```
rectangle(15, 40, 535, 270); //信息框
```

其中四个参数分别表示矩形左、上、右、下四条边的坐标。

在文本框内显示信息实现文本输出：

```
outtextxy(16, 41 + 18 * i, InputStr.c_str());
```

其中第一、二个参数表示输出内容的左上角角点的 (x,y) 坐标。

2) 按钮。

按钮的实现采用矩形边框+鼠标识别的方式进行实现。矩形边框的实现与上同理。在边框中加入文字表示按钮的含义：


```
RECT r1 = { 465, 280, 535, 300 };
settextcolor(BLACK);
drawtext(_T("输入"), &r1, DT_CENTER | DT_VCENTER | DT_SINGLELINE); //输入指令
```

通过鼠标识别的方法，当鼠标点击到按钮所对应的矩形边框内的时候，实现某一些功能：

```
else if (m.x >= 465 && m.y >= 310 && m.x <= 535 && m.y <= 330) { //确认
    if (m.message == WM_LBUTTONDOWN) { //左键弹起时
```

3) 输入框

输入框采用输入+文本显示的方式实现。当点击某一个按钮的时候，会弹出输入框，并提示输入：

```
InputBox(instruction, BUF_SIZE, "请输入指令");
```

将输入的内容保存到一个变量中并在之后输出到某一特定的文本框中：

```
outtextxy(17, 312, instruction);
```

2.2.2 进程间通讯的实现

进程间通讯通过共享内存的方式实现。程序可以通过建立对于共享内存的映射从而实现对于共享内存空间的访问。举*InstructionSpace*为例说明共享内存方式进程间通讯的实现，其他共享内存空间的实现类同。

InstructionSpace。

创建：

```
/*指令空间，用于客户端向服务器传输指令*/
//创建共享文件句柄
HANDLE InstructionSpace = CreateFileMapping(
    INVALID_HANDLE_VALUE, // 物理文件句柄
    NULL, // 默认安全级别
    PAGE_READWRITE, // 可读可写
    0, // 高位文件大小
    BUF_SIZE, // 地位文件大小
    "InstructionSpace" // 共享内存名称
);

//映射缓存区视图，得到指向共享内存的指针
LPVOID InstructionSend = MapViewOfFile(
    InstructionSpace, // 共享内存的句柄
    FILE_MAP_ALL_ACCESS, // 可读写许可
    0,
    0,
    BUF_SIZE
);
```

接收：

```
// 打开共享的文件对象
HANDLE InstructionSpace = OpenFileMapping(FILE_MAP_ALL_ACCESS, NULL, "InstructionSpace");
LPVOID InstructionRec = MapViewOfFile(InstructionSpace, FILE_MAP_ALL_ACCESS, 0, 0, 0);

char InstructionRecBuffer[BUF_SIZE] = { 0 };
MemoryInfoCpy(InstructionRecBuffer, InstructionRec); // 将共享内存数据拷贝出来
```

其中MemoryInfoCpy函数用于将共享内存中的内容从赋值到缓存变量数组中，用作后续的处理。

2.3 任务三：安全管理与进程同步

2.3.1 安全管理

在登录界面，会通过账号来区分用户与管理员，其中有两个预设账号：

管理员：

账号：123

密码：123

普通用户：

账号：321

密码：321

根据账号的不同会进入到不同的 Shell 中：

```
if (account[0] == '1' && account[1] == '2' && account[2] == '3')
{
    SysUserInteractiveUI(); //交互界面
    goto begin;
}
else if (account[0] == '3' && account[1] == '2' && account[2] == '1')
{
    ComUserInteractiveUI(); //交互界面
    goto begin;
}
```

在不同的 Shell 中，Shell 会通过 AccSpace 空间向 Simdisk 发送不同的信息以区分用户的类型：

```
strcpy_s((char*)AccSend, strlen("0") + 1, "0");
```

```
strcpy_s((char*)AccSend, strlen("1") + 1, "1");
```

其中 0 表示管理员，1 表示普通用户。

在读和写的过程中，Simdisk 会根据用户类型和对应文件的访问类型判断用户能否读或写对应的文件：

```
if (!(ComAcc[0] == 1 && UserAccBuf[0] == '1') || (SysAcc[0] == 1 && UserAccBuf[0] == '0')) //判断有无权限写入该文件
{
    Reply("无权限读取该文件!", ReplyInfoSend);
    JumpPointer(Disk_Pointer, OriSqe, 0);
    continue;
}

if (!(ComAcc[1] == 1 && UserAccBuf[0] == '1') || (SysAcc[1] == 1 && UserAccBuf[0] == '0')) //判断有无权限写入该文件
{
    Reply("无权限写入该文件!", ReplyInfoSend);
    JumpPointer(Disk_Pointer, OriSqe, 0);
    continue;
}
```

2.3.2 进程同步

不对多进程的读进行限制，从而在宏观上实现“共享读”。

对于“互斥写”，当一个进程在写入文件的过程中，Simdisk 会将该文件的绝对地址写入到 WriteSpace 空间中：

```
strcpy_s((char*)WriteSend, writeBuf.length() + 1, const_cast<char*>(writeBuf.c_str()));
ShareMemoryClear(WriteSend);
```

当其他进程需要再读取文件的时候，会判断当前需要读取的文件与 WriteSpace 空间中的内容是否相同，如果相同，则不可进行写入，从而实现“互斥写”：

```

if (InputFile == WritingFile && InputInstruction == "write")
{
    clearrectangle(90, 15, 200, 30);
    outtextxy(90, 15, ReplyInfoRecBuffer);
    outtextxy(16, 41, "当前文件正在被其他用户写入，请稍后再试！");
    char NULLChar[] = ""; //清空输入
    strcpy_s((char*)InstructionSend, strlen(NULLChar) + 1, NULLChar);
    goto begin2;
}

```

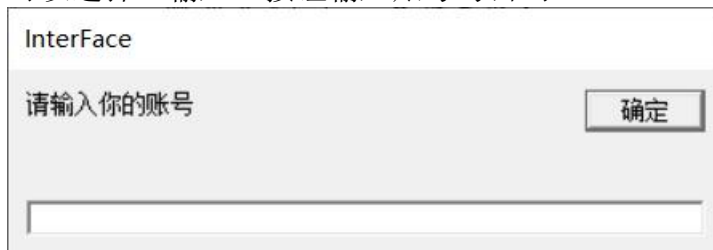
虽然 Simdisk 在微观上串行执行，但是从宏观上看，当发生了同时写入同一个文件的时候会提示错误；当同时写入不同文件的时候，由于写入速度较快，表现出并行的现象，从而实现宏观上的并行。

三、 运行结果分析

3.1 登录界面



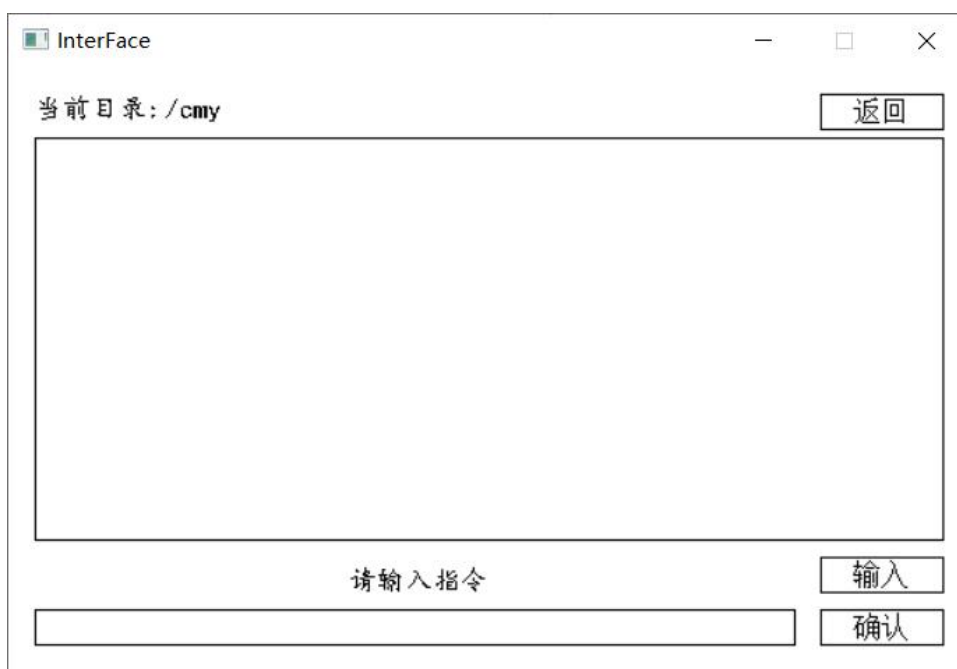
可以选择“输入”按钮输入账号与密码：



如果密码正确，进入下一个界面；如果密码错误，提示错误信息：



3.2 交互界面

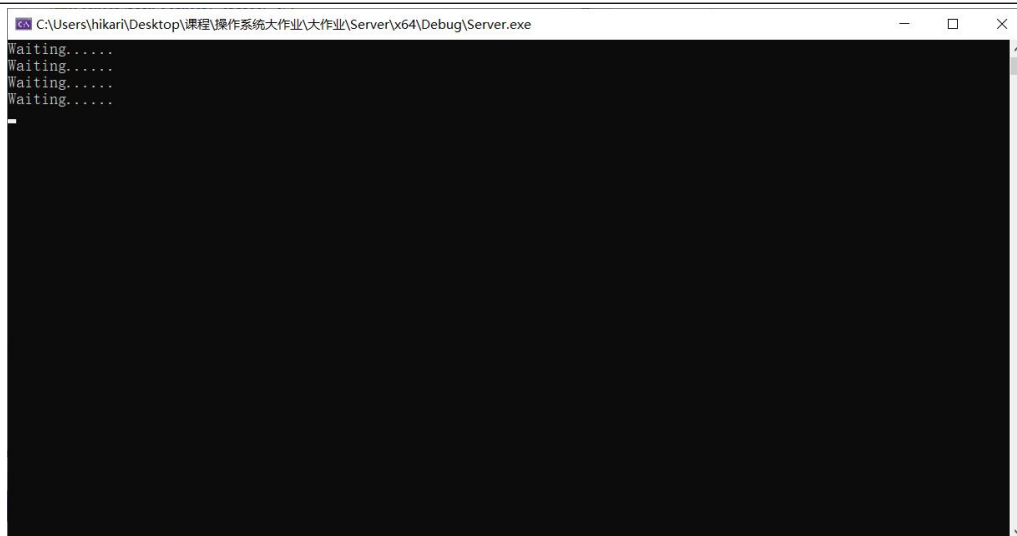


交互界面会显示当前目录，默认为根目录/cmy。点击输入可以输入指令，点击确认发送指令。中间为交互信息框，错误信息或者 Simdisk 返回的文件系统信息会在此显示。

3.3 功能实现

1) Simdisk 内核界面

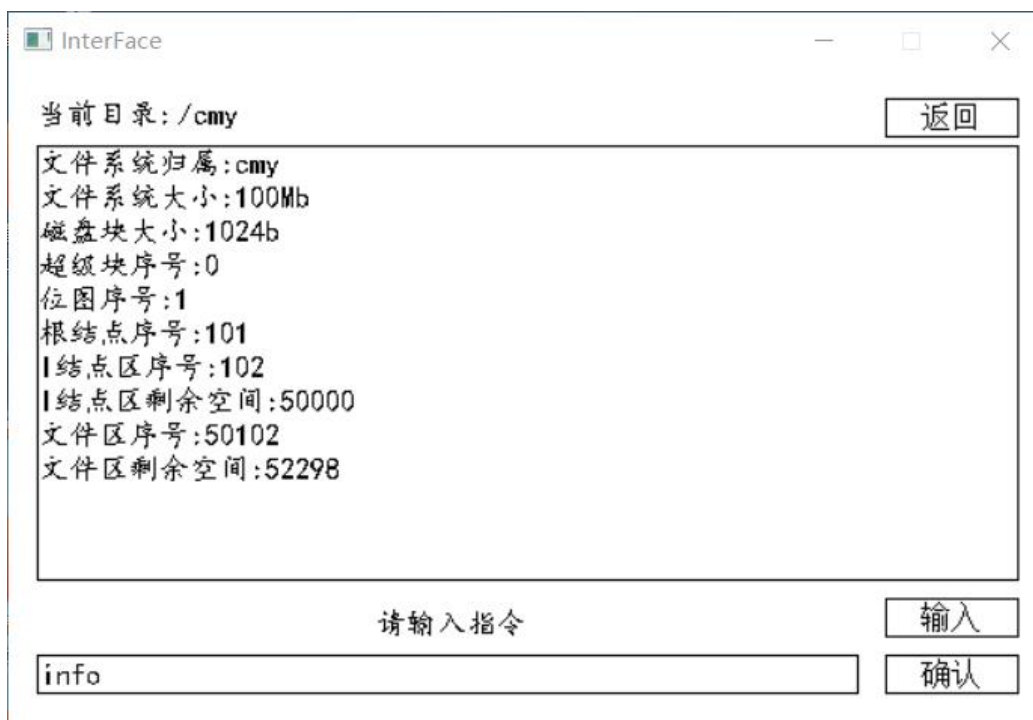
当没有信息输入的时候，界面将显示 waiting...:



当有信息输入或输出的时候，界面将显示相关的信息：

```
Get info:info
Return Info:文件系统归属:cmv
文件系统大小:0
磁盘块大小:1024
超级块大小:0
位图序号:1
根结点序号:101
I结点区序号:102
I结点区剩余空间:50000
文件区序号:50102
文件区剩余空间:52298
```

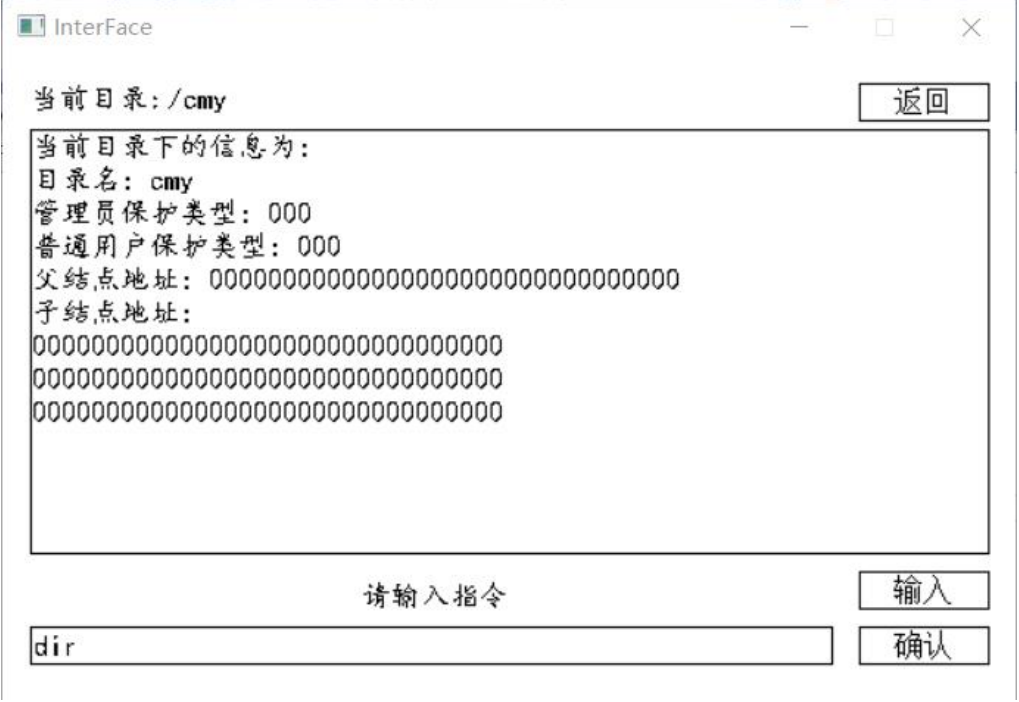
2) Info



显示了文件系统的对应信息。

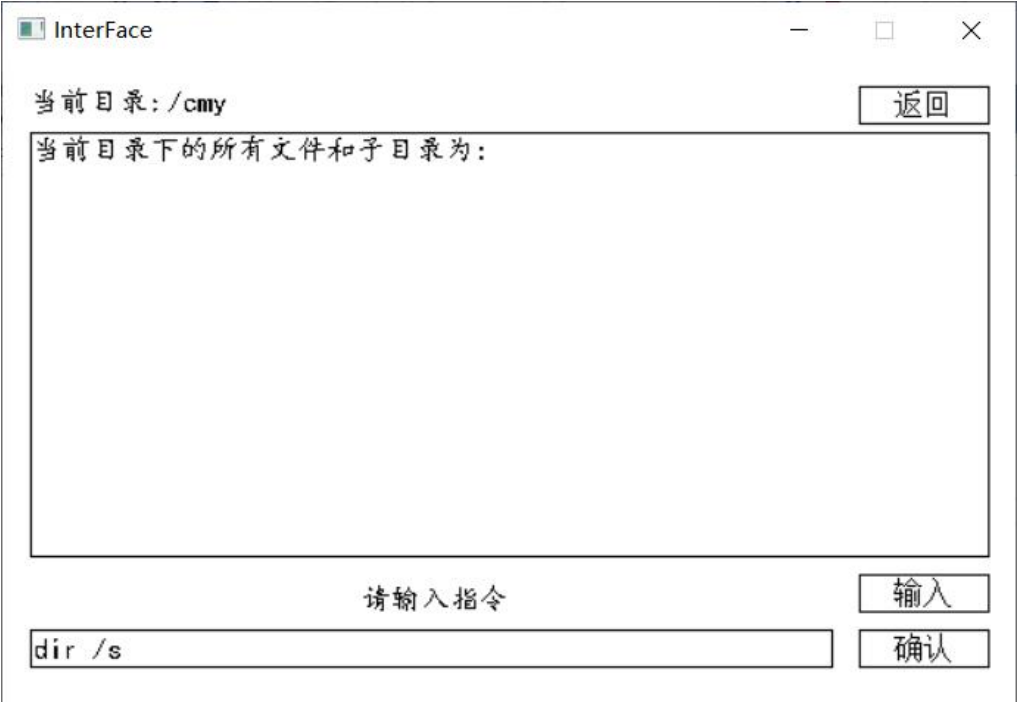
3) Dir

不带 /s 指令，可以返回目录的信息：



The screenshot shows a window titled "InterFace" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, the text "当前目录: /cmy" is displayed at the top left, with a "返回" button to its right. Below this, a large text area contains the following information:
当前目录下的信息为:
目录名: cmy
管理员保护类型: 000
普通用户保护类型: 000
父结点地址: 00000000000000000000000000000000
子结点地址:
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
Below the text area, the text "请输入指令" is centered. To its right are two buttons: "输入" and "确认". At the bottom left, there is a text input field containing the command "dir". To its right is another "确认" button.

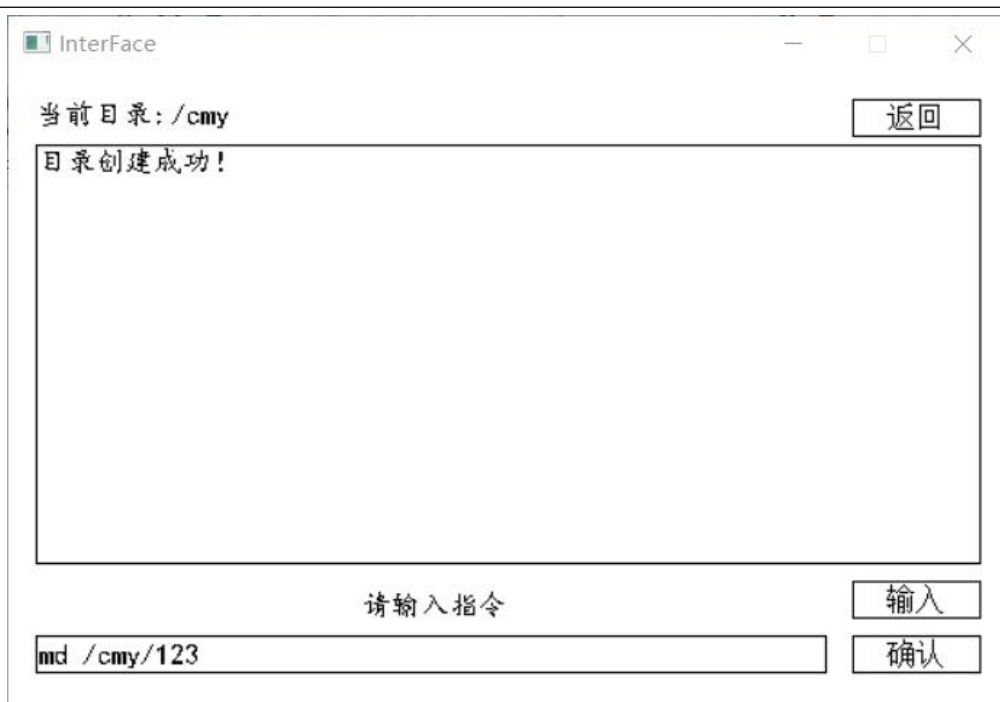
带/s 参数可以显示该目录下的所有子文件的名称：（测试时该目录下还没有子文件）



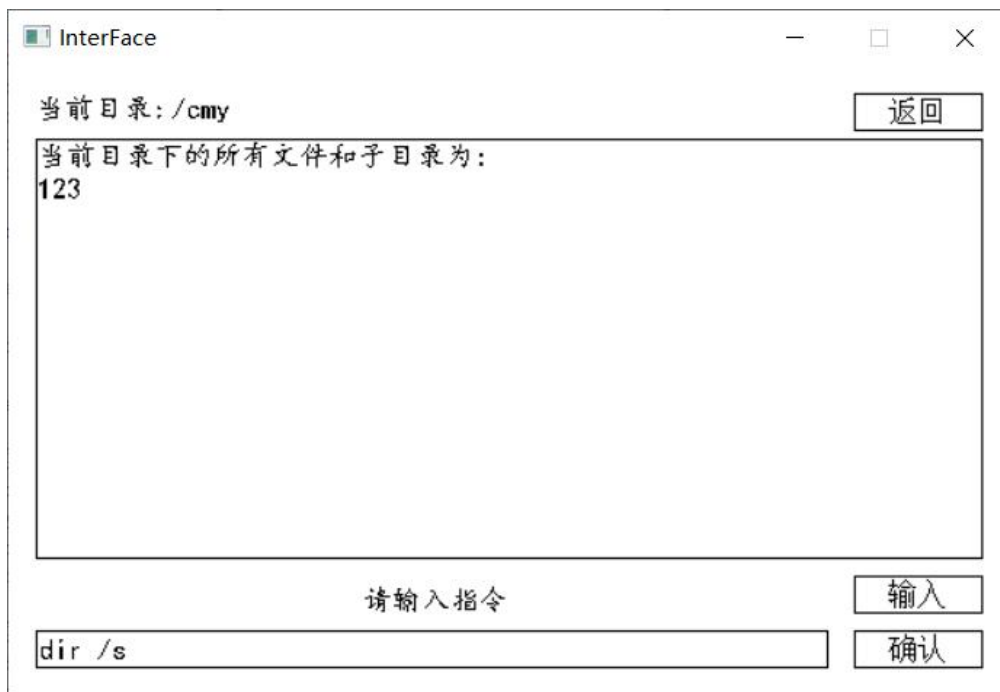
The screenshot shows the same "InterFace" window. The text "当前目录: /cmy" and the "返回" button are still present. The large text area now contains the text "当前目录下的所有文件和子目录为:". Below this, the text "请输入指令" is centered, with "输入" and "确认" buttons to its right. The text input field at the bottom now contains the command "dir /s".

4) md

在根目录下创建一个子目录 123:



查看根目录信息:



发现根目录中显示了新创建的文件夹的信息。

5) Rd

删除/cmy 目录下的 123 目录:

InterFace

当前目录: /cmy

返回

目标目录删除完成

请输入指令

rd /cmy/123

输入

确认

重新查看目录信息:

InterFace

当前目录: /cmy

返回

当前目录下的所有文件和子目录为:

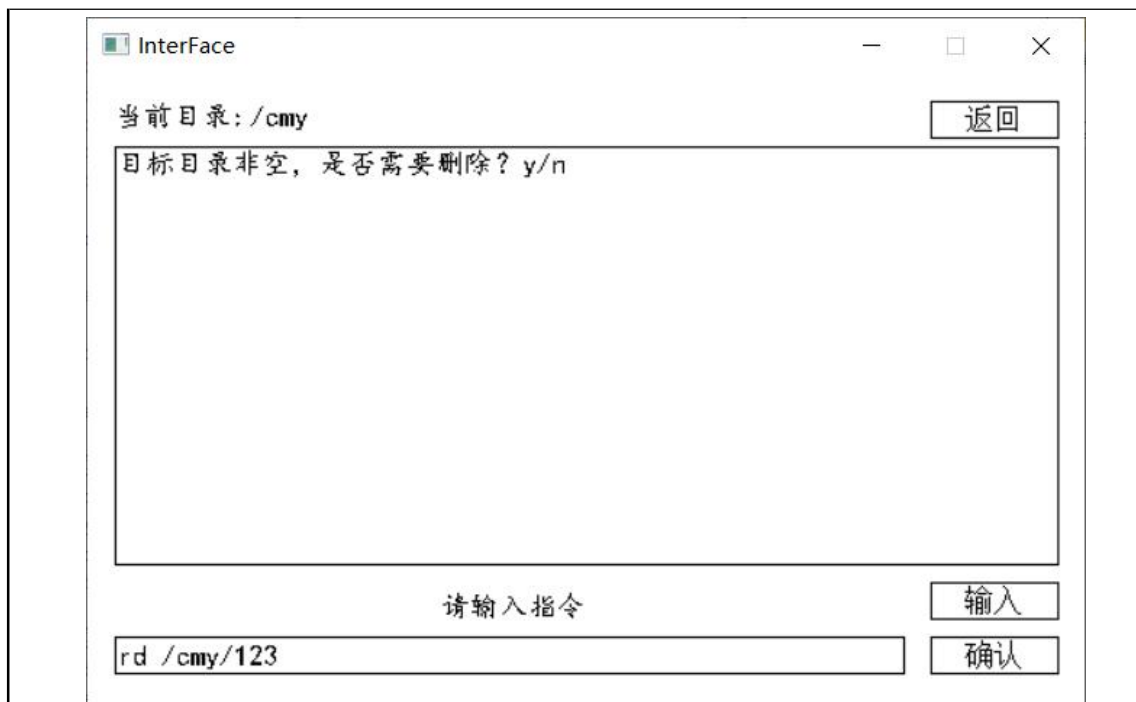
请输入指令

dir /s

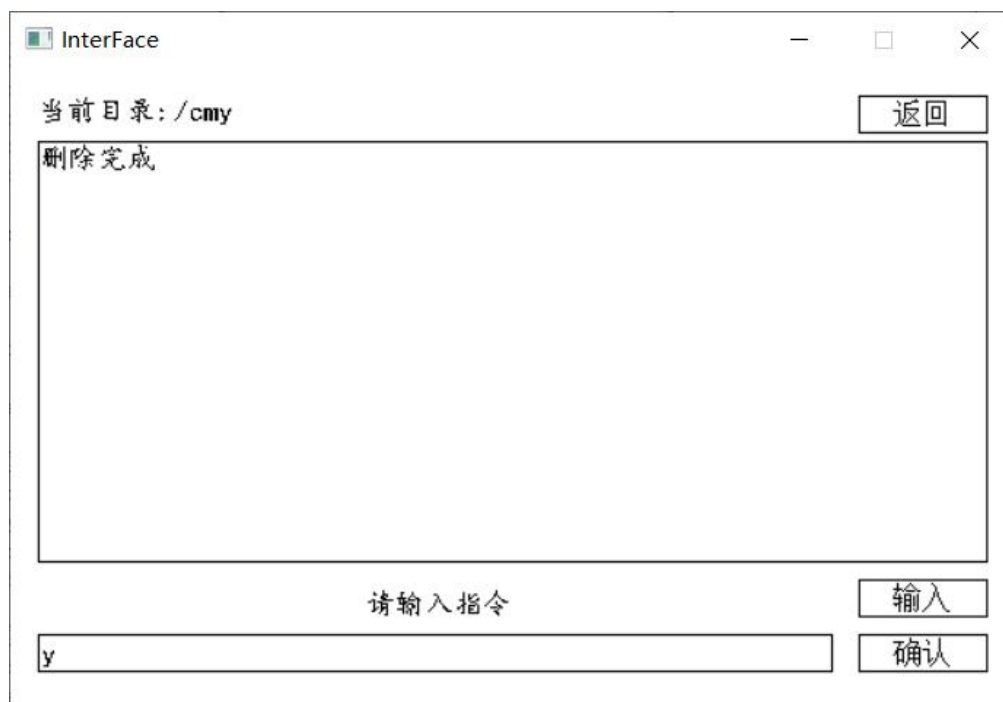
输入

确认

123 目录已经被删除。
如果文件中包含子目录:

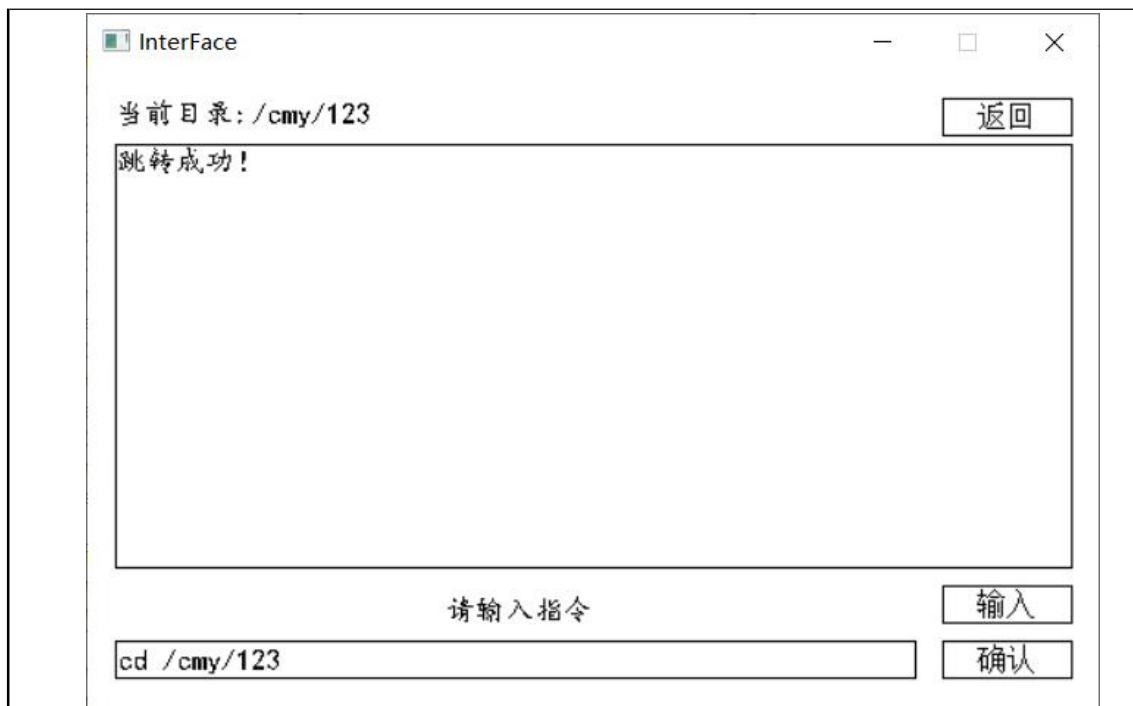


交互界面将提示是否删除。若选择 **n**，则不执行删除操作;若选择 **y**，则执行删除操作:



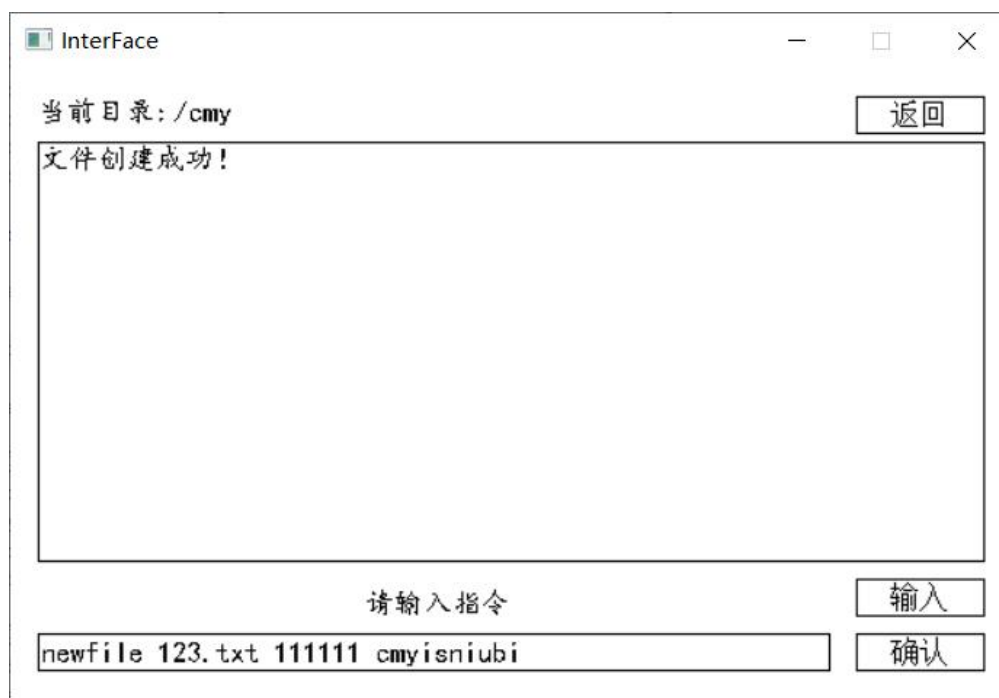
6) Cd

通过跳转指令，可以跳转到对应的目录:

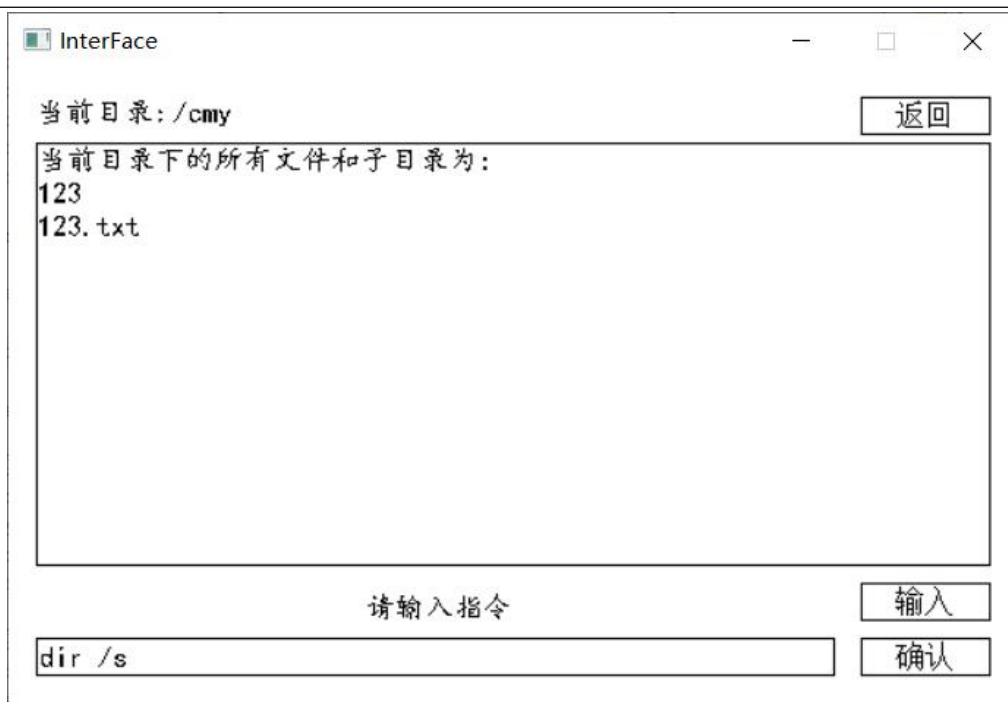


7) Newfile&cat

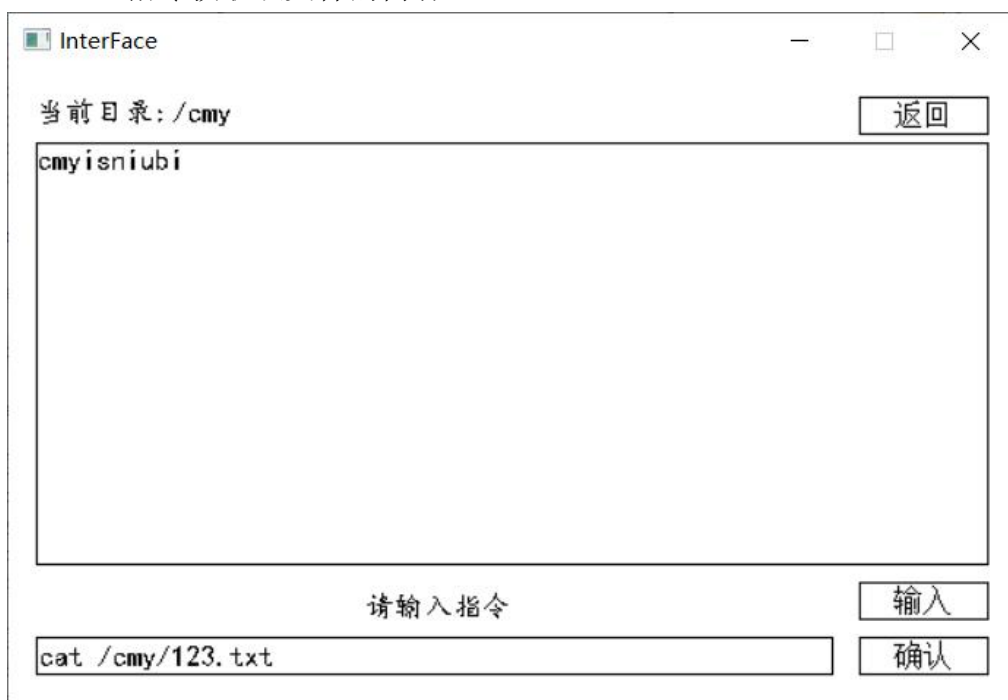
创建一个文件名为 123.txt, 保护类型为 111111, 文件内容为 cmyisniubi 的普通文件:



可见该文件创建成功:



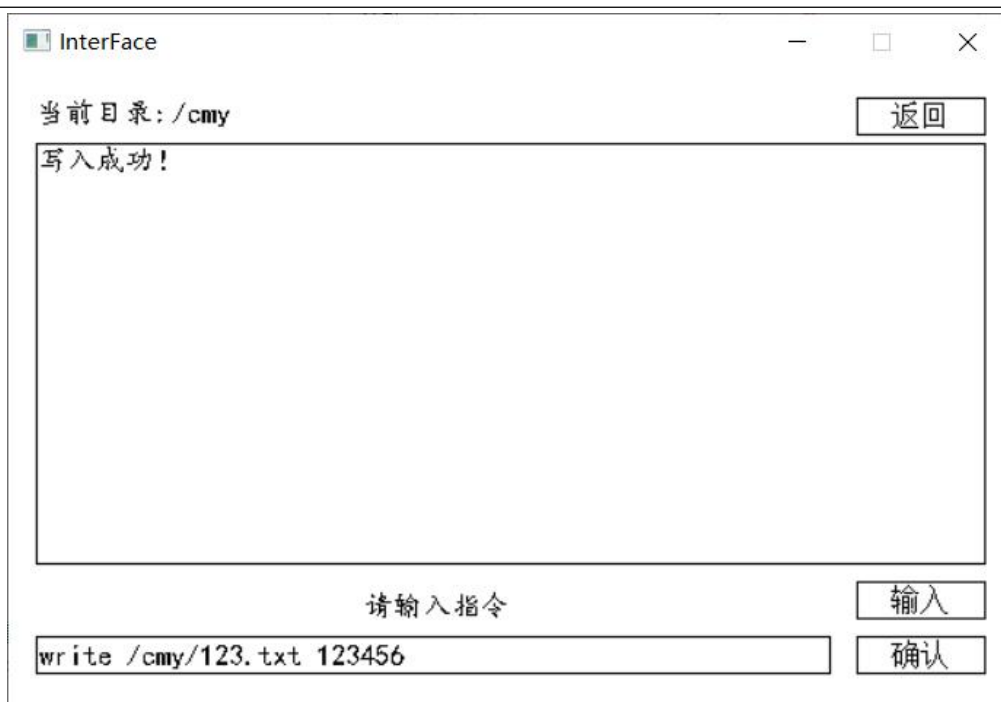
通过 cat 指令获取该文件的内容:



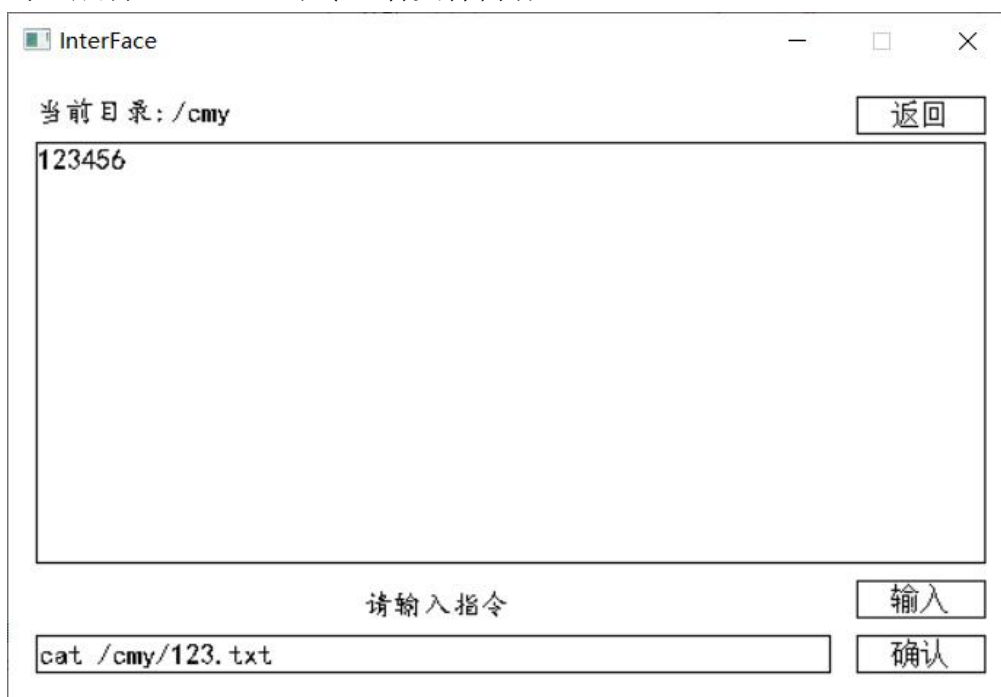
所显示的内容正确

8) Write

通过 write 命令重写文件内容:

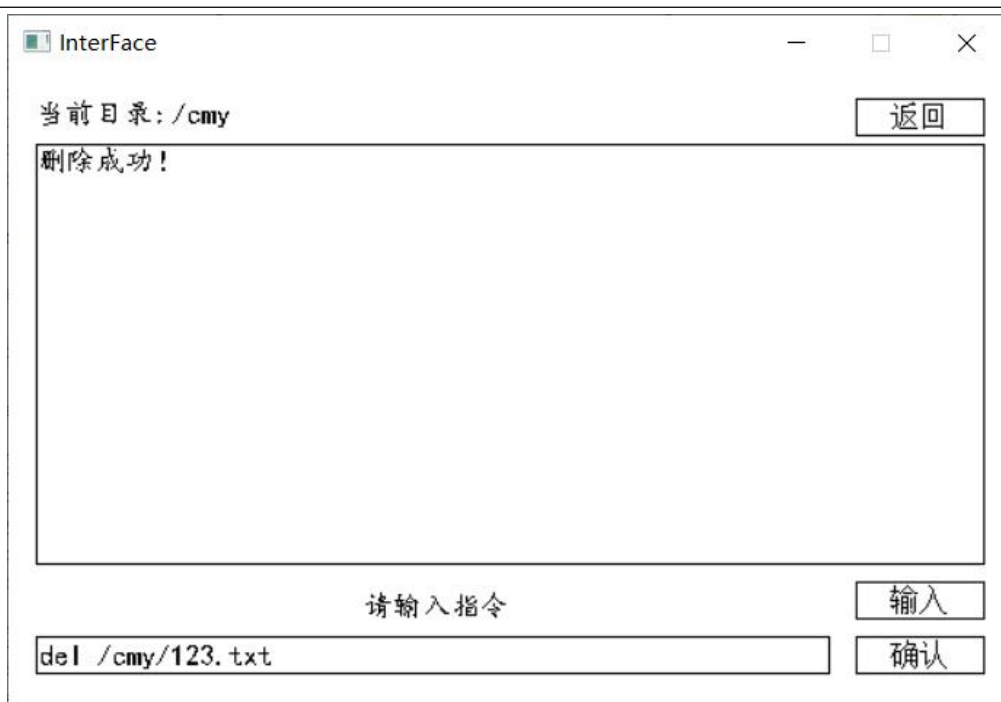


写入成功。通过 cat 命令查看文件内容:

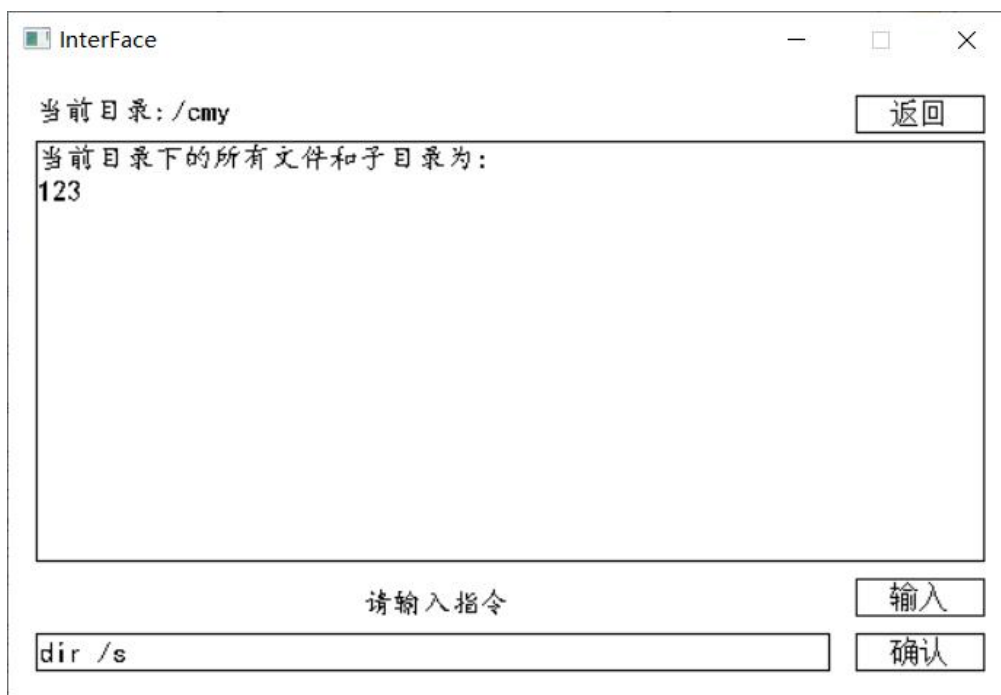


文件内容已经被改变。

9) Del



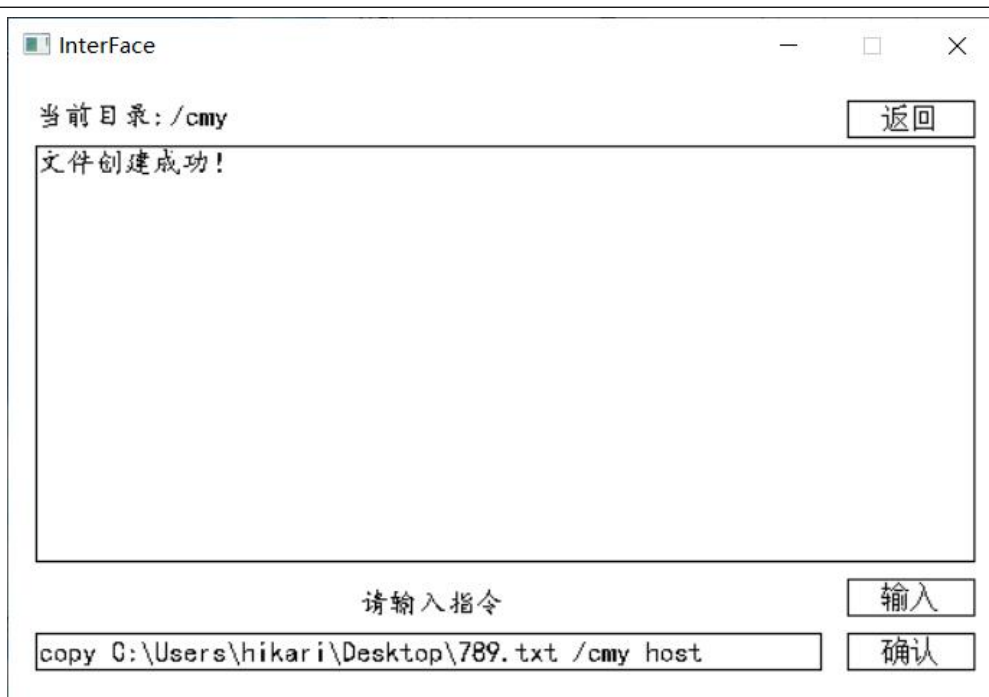
显示删除成功，通过 `dir /s` 查看目录信息：



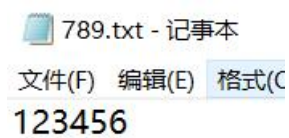
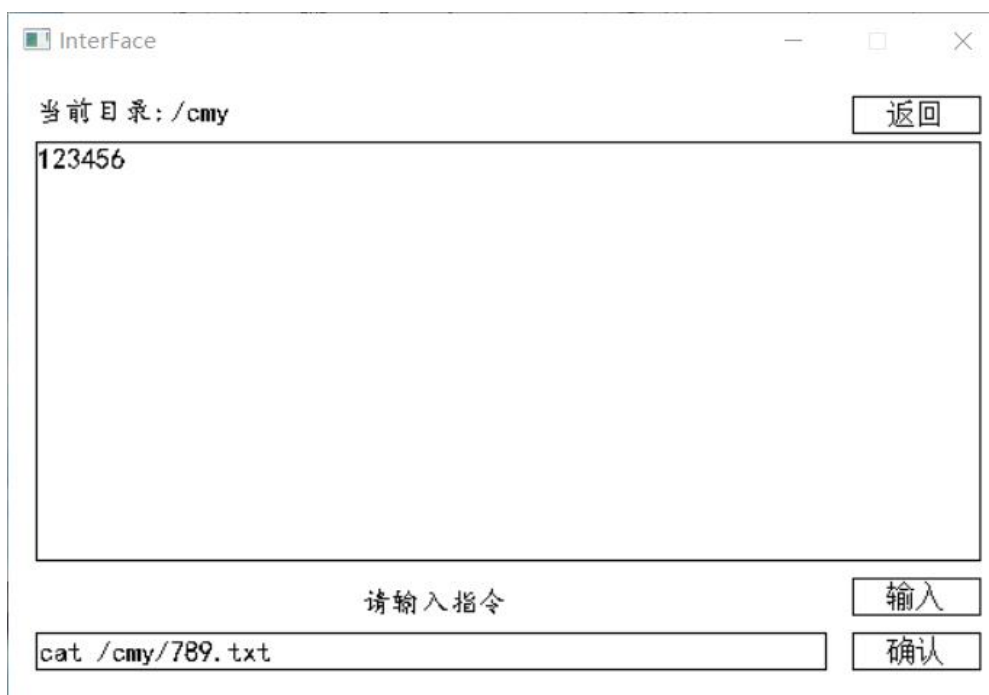
发现已经没有该文件了。

10) Copy

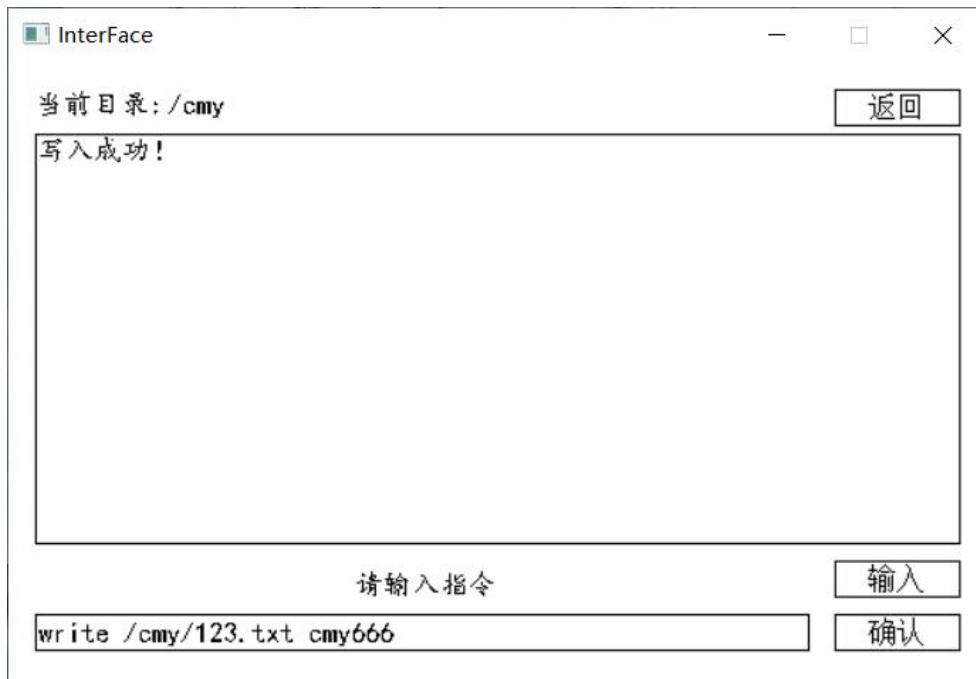
尝试将主机中 789.txt 文件复制到文件系统根目录：



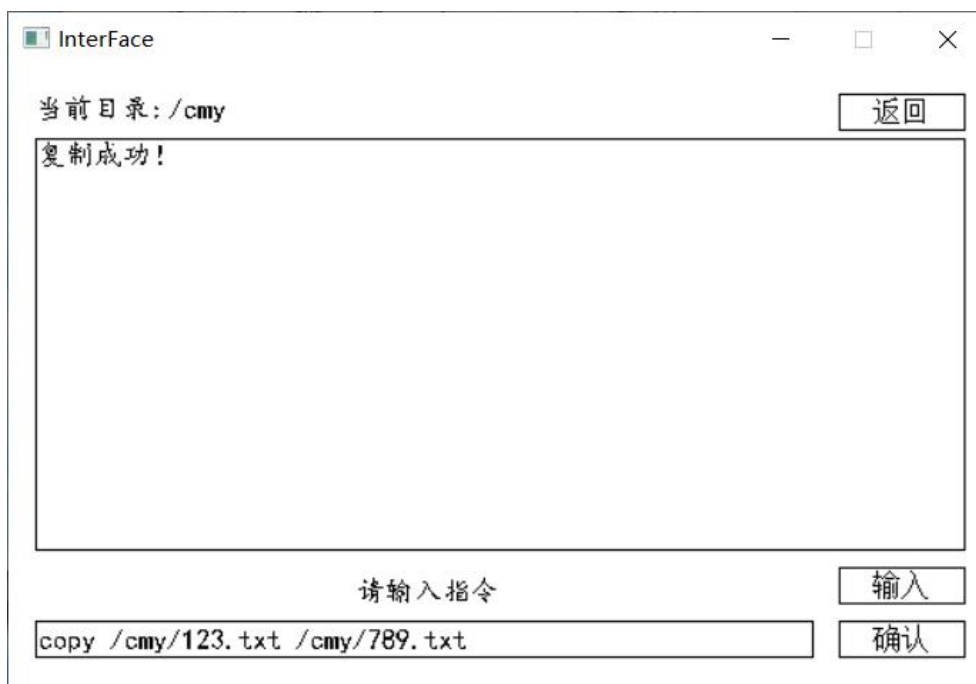
文件创建成功。通过 cat 命令查看文件内容：

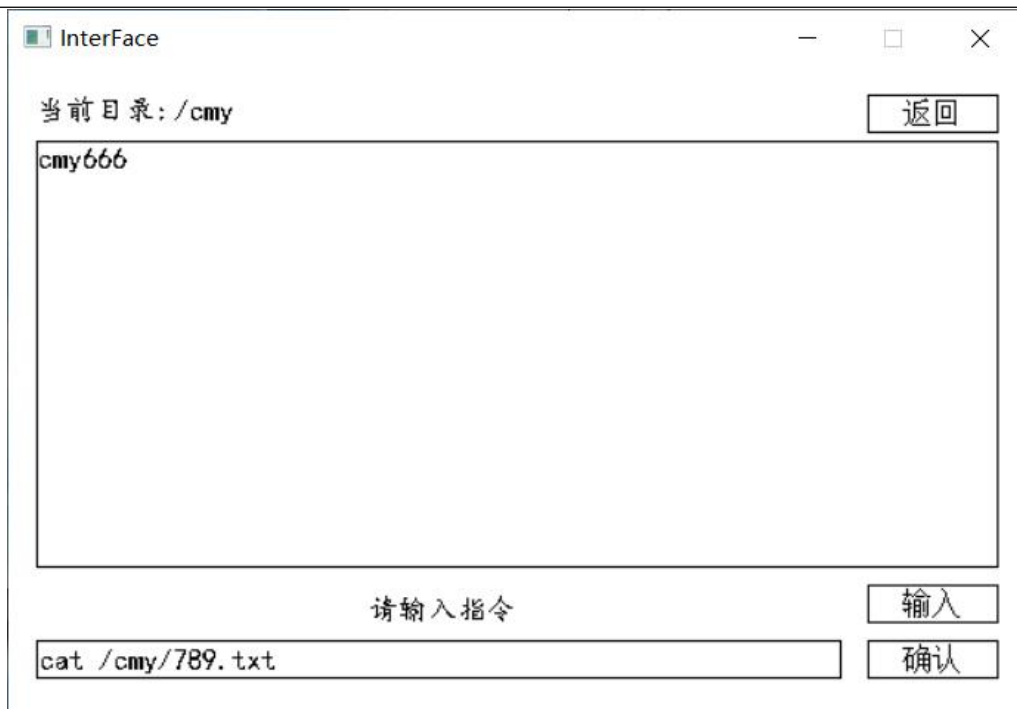


可见内容正确。在文件系统下完成文件的复制操作：首先改写 123.txt 中文件内容：



然后执行复制并查看 789.txt 中文件内容:

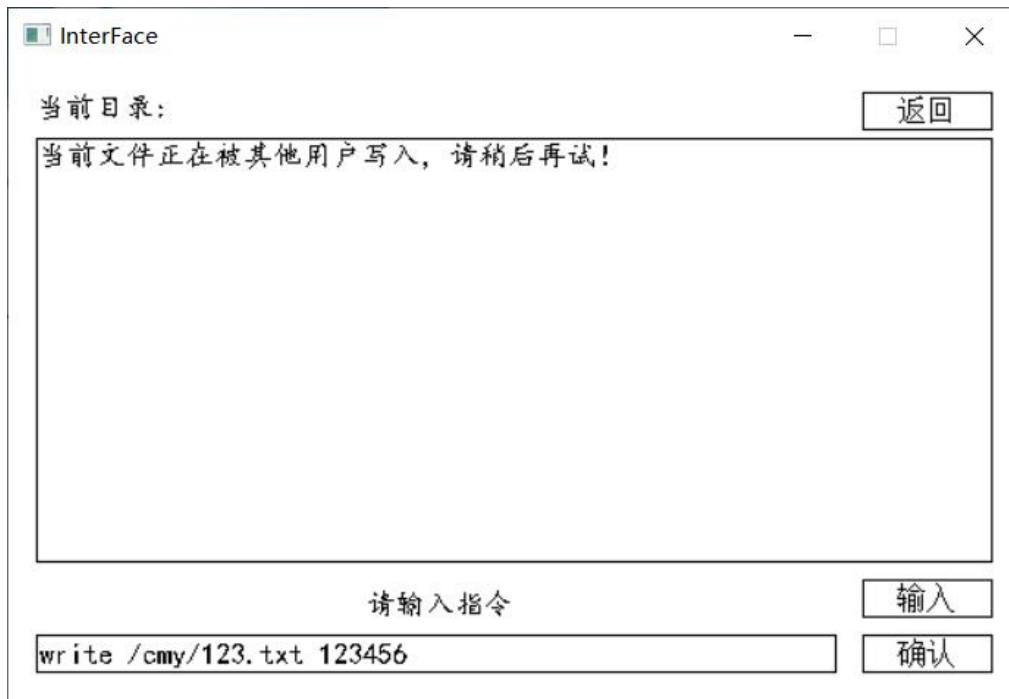




可见复制成功，内容正确。

11) 互斥写

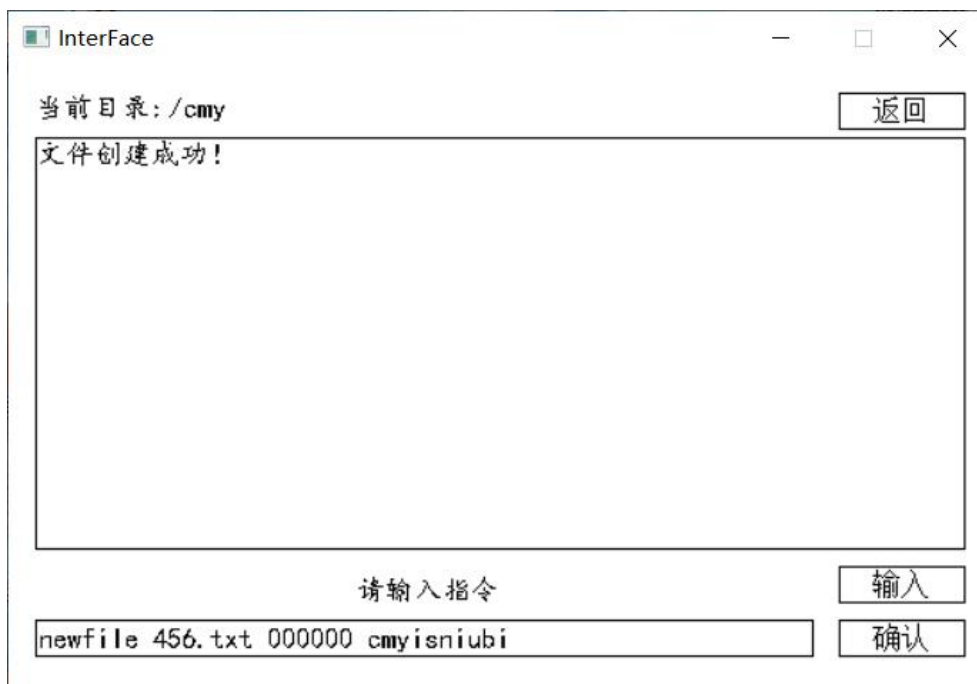
当两个用户同时写入同一个文件的时候：



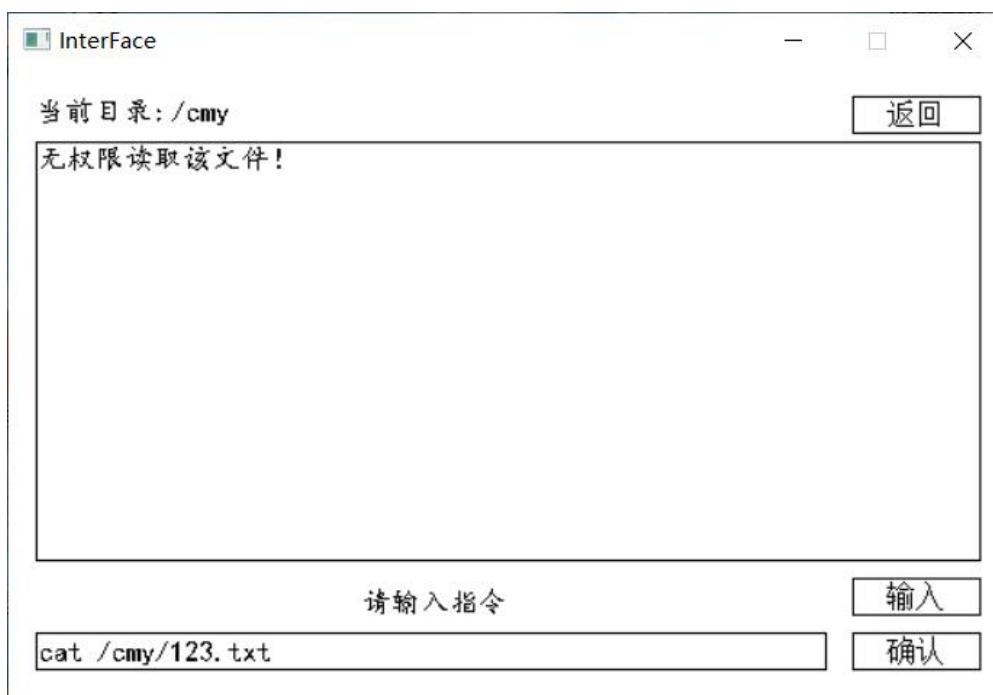
后写入的用户将提示不可写入

12) 文件保护

尝试常见一个文件保护位 000000 的文件：



尝试访问它:



无法访问。

小结

本次实验要求构建一个微型的文件系统，实在是非常的有挑战性。对于我这个计组学得不好的人来说，硬盘上指针映射来映射去简直令人头疼。但通过本次实验，我也确实收获很多：

1.加深了对于文件系统的理解。理论课上对于文件系统的讲解相对抽象，说白了学完理论课我还是不知道文件系统是怎么构建和工作的。当我仔细地翻书，一步一步地对着书上的内容进行文件系统的架构的时候，我才发现原来真的是

可以构造出一个文件系统来的，并且操作系统与硬盘之间的交互变得更为清晰。

2.提高了软件架构的能力。文件系统的架构是一个很麻烦的工作。文件有多大、目录有多大、i 结点有什么内容、文件系统的区域如何分配.....这些都需要根据文件系统的大小、功能等条件进行合理的架构。

3.提高了程序编写的能力，加深了对于编写较大型的程序的理解。一个较为大型的程序往往意味着众多的函数，如何编写这些函数，使他们达到“高内聚、低耦合”的效果并增强其复用性、如何对函数进行命名，做到见名知义的同时还能够进行很好地区分，这些都是需要解决的问题。

指导教师评语及成绩

评语：

成绩： 指导教师签名：

批阅日期：