

**操作系统课程设计**

**题 目： 仿linux文件系统**

**班 级： 2022级计科二班**

**姓 名： 孙 正**

**学 号： 202230442395**

**指导老师： 刘发贵**

**提交时间： 2024.11.24**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

## 课程设计目的

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。

## 课程设计要求

（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%。课程设计要求同时上交打印文档，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，建立一个大文件，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：
3. 盘块大小1k
4. 空闲盘块的管理：Linux位图法
5. 结构：超级块, i结点区, 根目录区
6. 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。
   7. cat …: 打开文件。
   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

* 1. del …: 删除文件：删除指定文件，不存在时给出出错信息。
  2. 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。

1. 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。

**【PARTⅠ】**

模拟Linux文件系统

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模拟Linux文件系统。

【文件系统的结构】

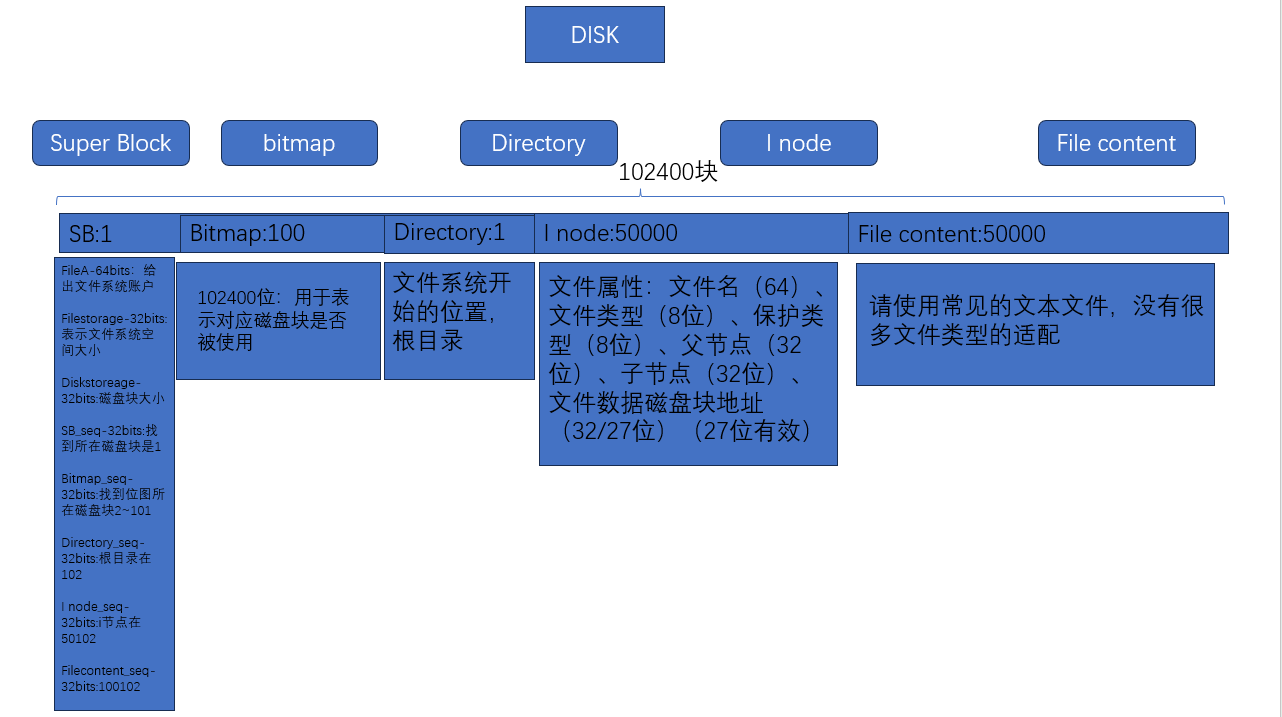


图1

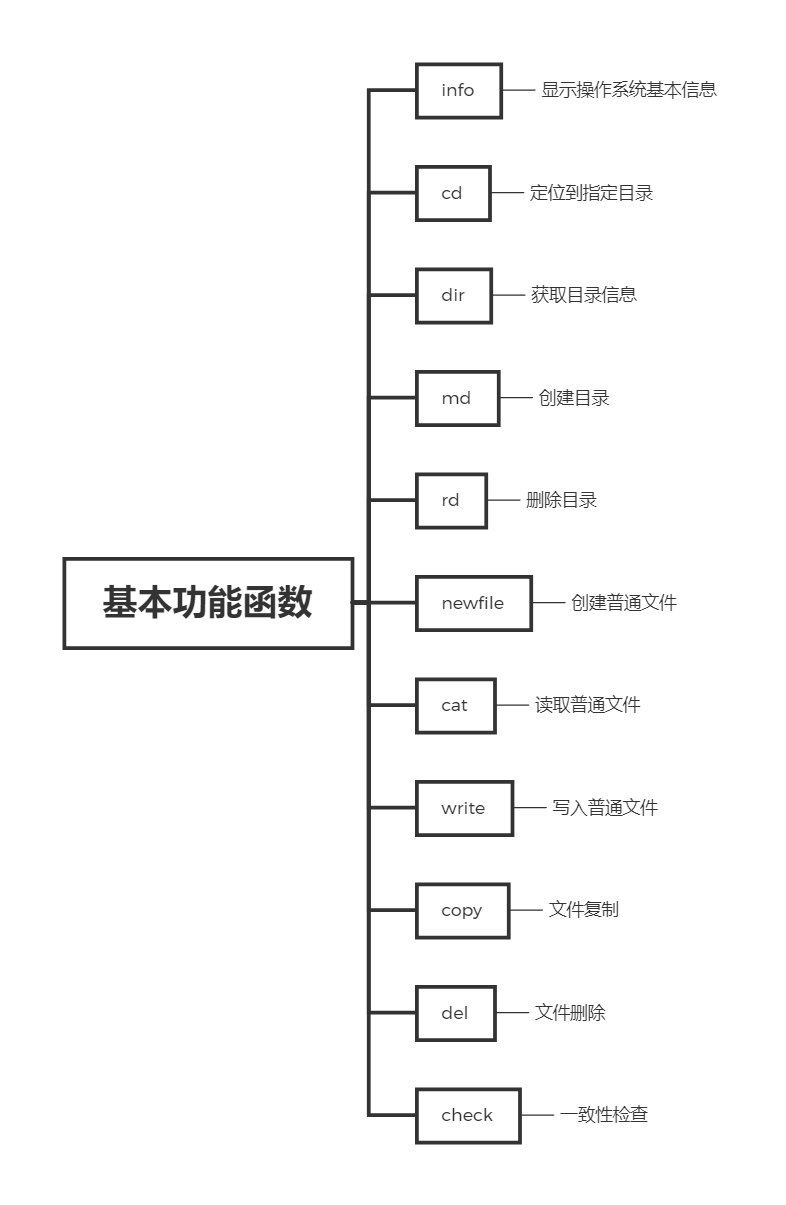
对于磁盘的创建选择手动建立一个100M的diskapp.txt文件作为操作对象，直接对其中所有文本设置为二进制数进行编码和操作，严格意义上104857600bits说明需要这么个数目的0数字，其中字符和二进制数的转换主要依赖ASCII码表。

然后以书本中磁盘的结构作为范例，减去多用户标识和磁盘分区的工作，而是通过保护类型确定用户分类仅为管理员和普通用户，这样更符合一般电脑用户的特性，不对使用者进行账户区分。

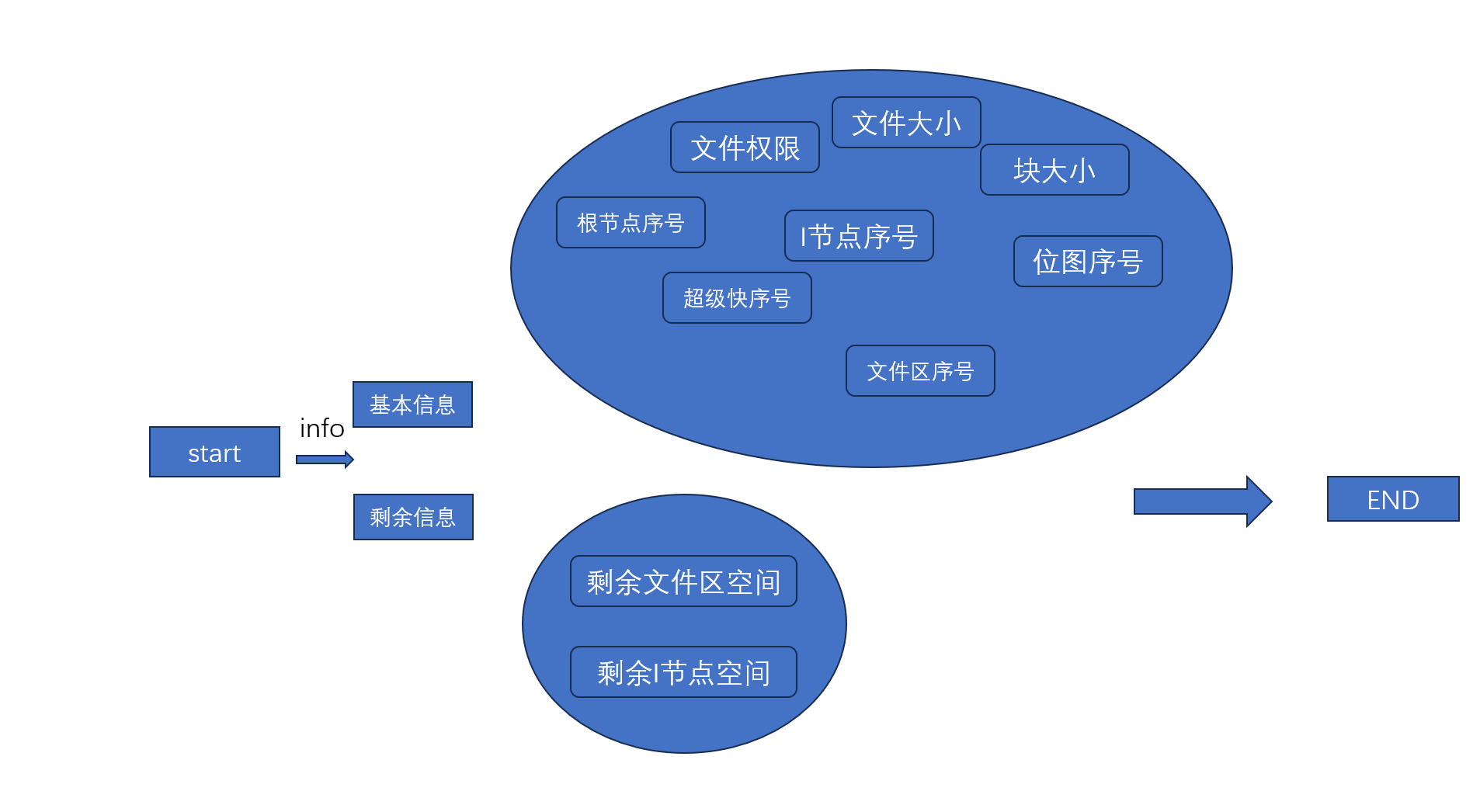
关于具体文件结构如图1，上面会标注具体用到磁盘块数以及盘块各部分进行管理的内容，下面就上面内容做少许补充：

* **超级块**是存储文件系统具体信息的重要块，仅占一块，但是作为首块不易丢失同时维护方便，拥有帮助其他分区备份地址的作用，但是不能直接作为存储数据索引块使用。
* 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。磁盘块大小为1K，由于磁盘的大小为100M，则该硬盘有102400个磁盘块，需要用17位二进制数表示，则块地址0~102399。而块内地址从0开始编制，块内地址表示为0~1023，用10位二进制数表示。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。
* **根目录**，文件开始的地方，顾名思义这里的文件编码是能够为找到其他文件节点的关键节点，其和其他的节点存根目录一样作为树节点的方式被编码为二进制文件存在此处
* **Inode**存储文件信息和文件数据磁盘块存放地址
* **数据块**，就是，数据块啊！！

//simdisk project



基本功能函数速览



Info函数的输入格式为：info，显示整个系统信息，文件可以根据用户进行读写保护。

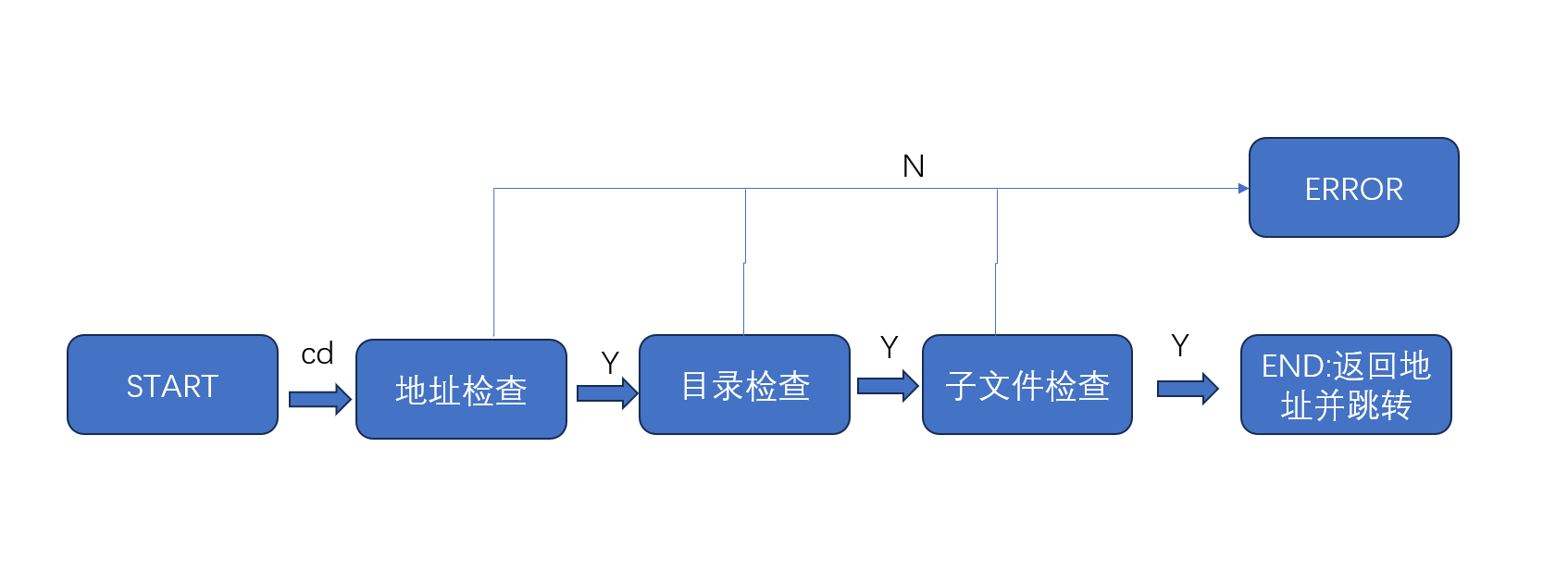


图2

cd指令格式：cd+文件地址，改变当前工作目录，目录不存在时给出出错信息。代码首先从输入流 Buf 中读取用户输入的字符串，并存储在 cdBuf 中。接着，调用 cd\_HELP 函数，传入当前磁盘指针 Disk\_Pointer 和读取的 cdBuf 内容，获取相应的帮助信息，并将其保存到 ReplyInfo 变量中。最后，使用 response 函数将获取的帮助信息 ReplyInfo 发送到 ReplyInfoSend，以便进一步处理或输出。

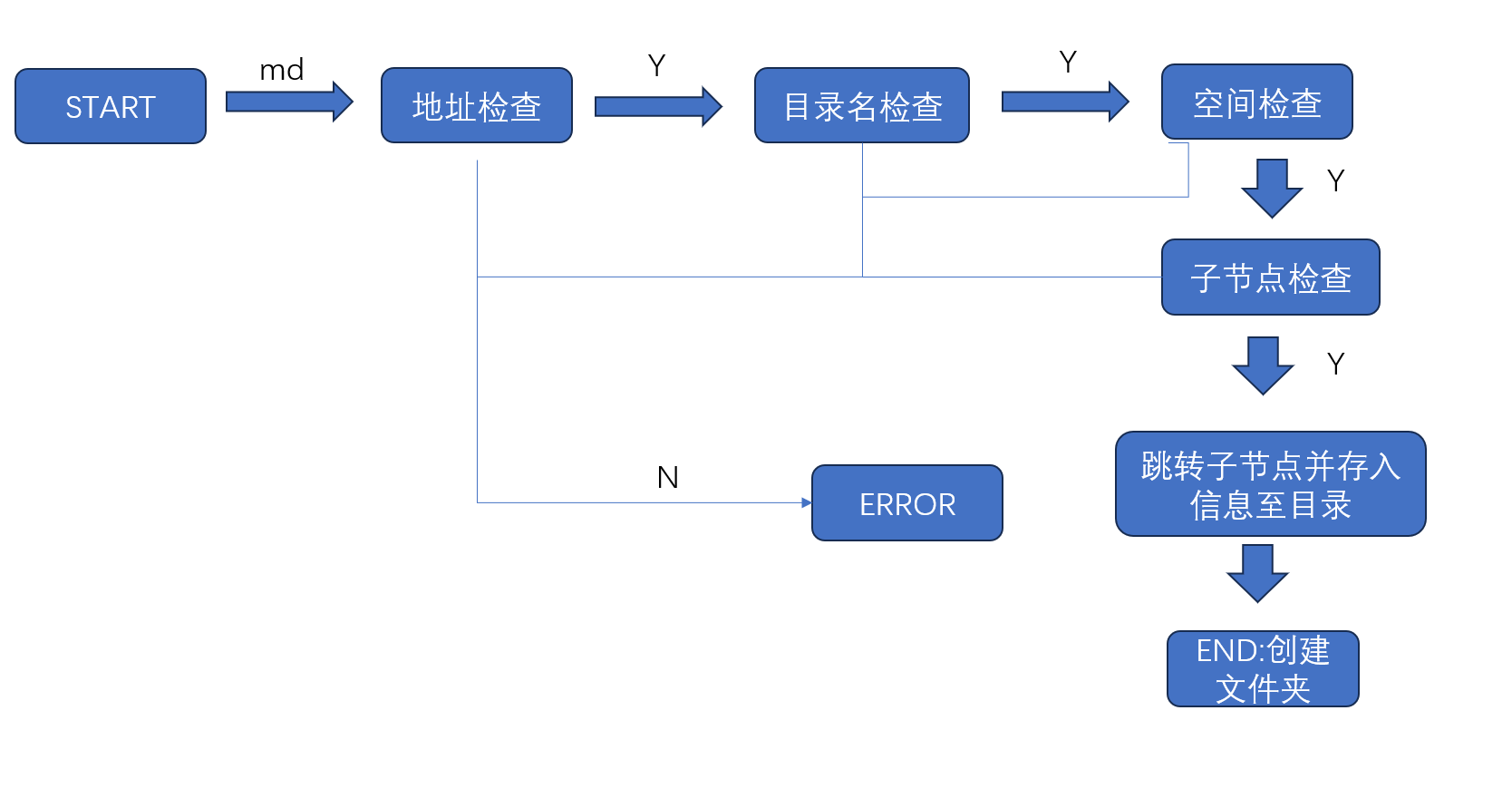


图3

md指令格式为：md+文件地址+目录名。代码实现了一个目录创建的操作。首先，读取用户输入的目录地址，并通过 mdAddLocation 确定该目录的名称。如果目录名称无效或超过8个字符，系统会返回错误提示并停止操作。如果文件系统没有足够的空间创建新目录，或者目录名与现有文件冲突，也会返回错误信息。接着，判断该目录下的子结点是否已满，如果没有满，系统会将新目录信息写入子结点。最后，对新目录进行初始化，包括设置目录名称、输入父节点信息，并更新文件系统的位图，最后返回成功信息。如果过程中有任何问题，系统会适时提供失败的反馈。

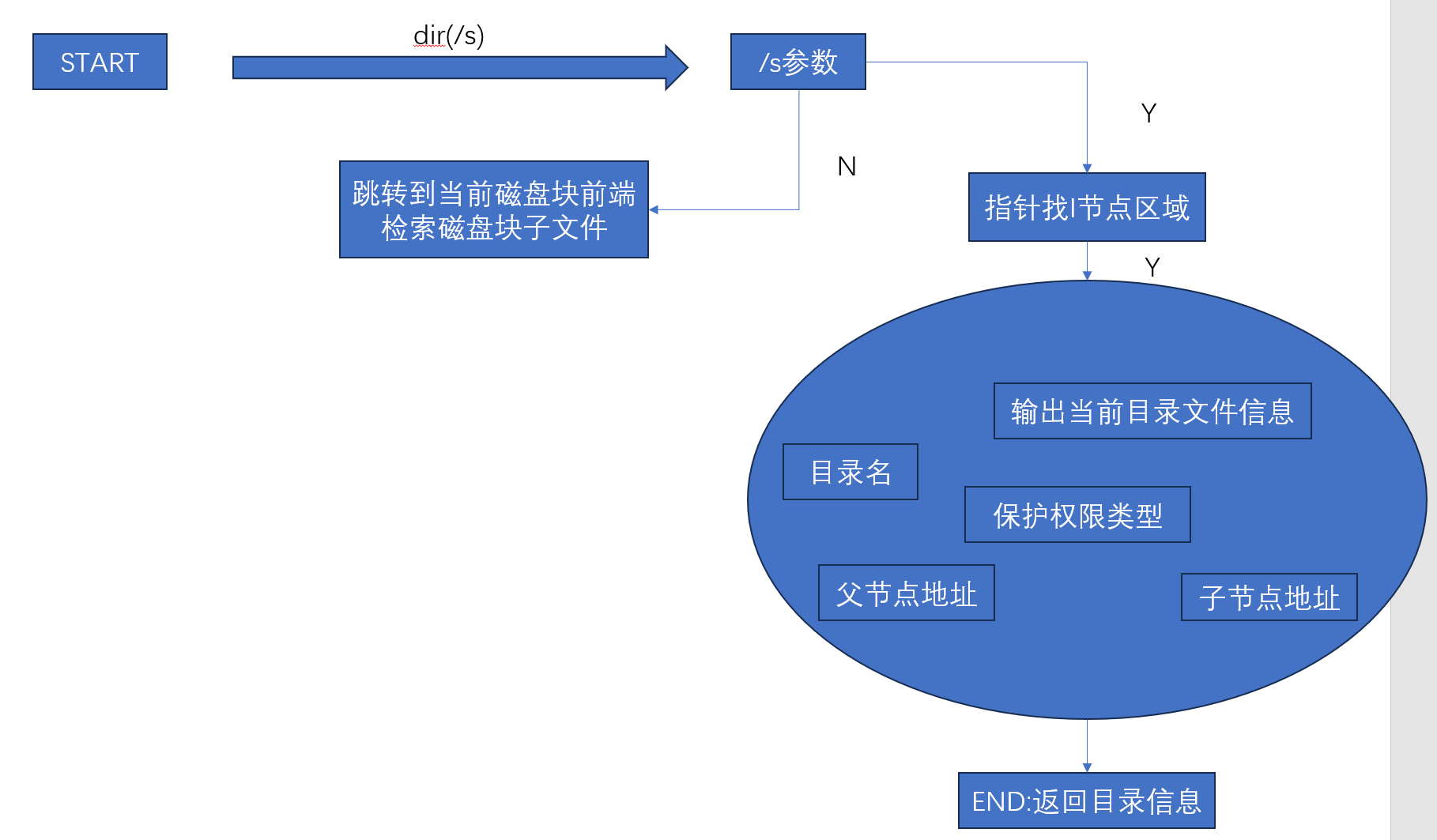


图4

dir指令格式为：dir(+/s)( +绝对地址)，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。代码首先读取用户输入的目录地址，如果输入的是 /s 参数，则通过 ShowAllChildNodeName 列出当前目录下的所有文件和子目录，并将结果返回给用户。如果输入的是其他目录地址，则使用 dir\_HELP 查找该目录并通过 ShowAllInfo 显示其信息。如果目录无效，则返回错误信息。最后，通过 jump2whePTR 恢复磁盘指针到原始位置。

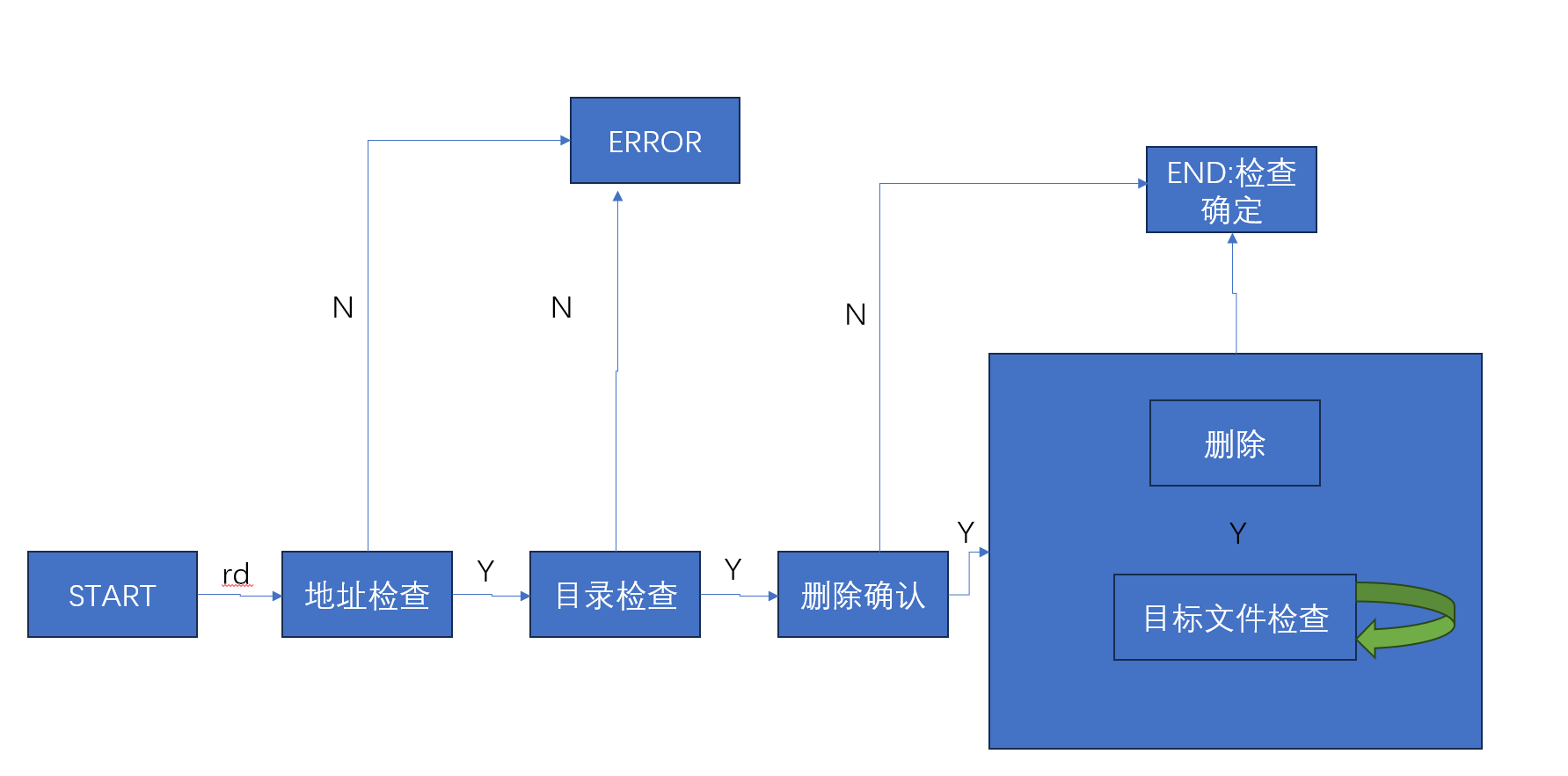


图5

rd指令格式：rd +文件地址，删除指定目录下所有文件和子目录。代码处理了目录删除操作。首先，通过读取用户输入的目标目录地址，获取相应的磁盘块。如果目标是根目录，则返回错误信息，因为根目录不能被删除。如果目标文件是普通文件而非目录，也会返回删除失败的消息。如果目标目录非空，系统会提示用户是否确认删除该目录。如果用户确认，系统将删除目录并更新磁盘位图；如果目录为空，系统直接删除目录并清除相关信息。如果没有找到目标磁盘块，系统会返回错误信息并跳回原始位置。在整个过程中，操作前后都会进行相应的检查和错误提示，确保用户操作的正确性。



图6

newfile指令格式：newfile +绝对地址或相对地址+文件保护类型+文件内容用于新建文件。代码用于处理文件创建操作。首先，程序接收用户输入的文件名、文件保护类型、和文件内容，并对这些输入进行格式检查。若文件名或保护类型不符合规定，程序会返回错误信息并继续执行。接着，检查文件系统是否有足够的空间来创建新文件，包括是否有空闲的inode和数据块。如果文件名与现有文件重复，或者文件系统已满，则会返回错误信息并中止操作。

如果空间充足且没有冲突，程序会继续进行目录结构的更新，将新的文件信息插入到相应的目录中。之后，文件的各种信息（如文件名、父节点地址、保护类型、文件内容等）会被输入到磁盘中，文件的inode和数据块都会标记为已使用。最后，返回文件创建成功的消息，并跳回到原来的操作位置。

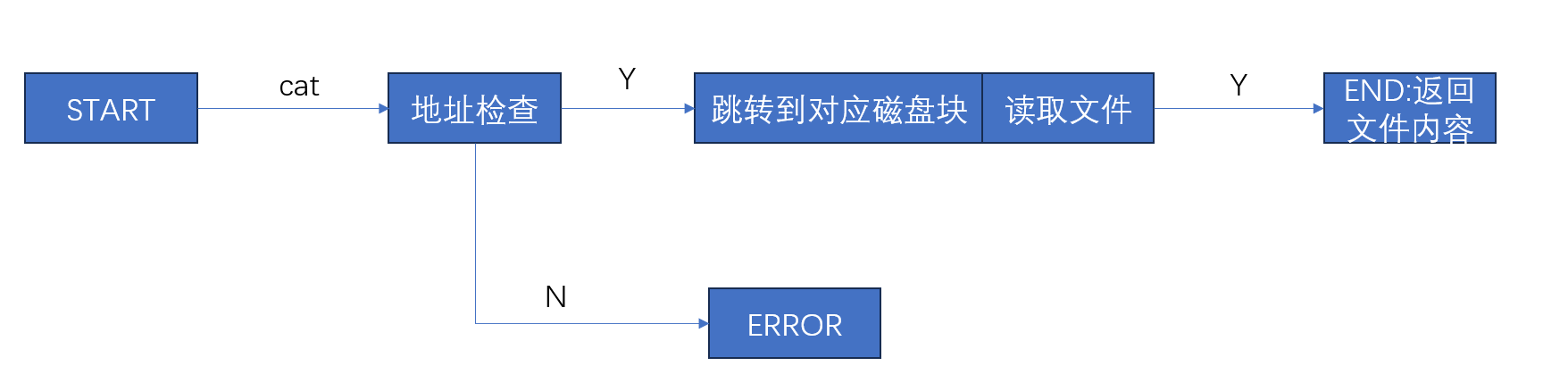


图7

cat指令格式为：cat +绝对地址。处理用户请求读取指定文件内容的操作。它首先获取用户输入的文件地址并检查文件是否为普通文件，如果不是则返回错误信息。如果文件存在，接着验证当前用户是否有权限读取该文件，基于普通用户和系统用户的访问权限进行判断。如果用户有权限，文件内容会被读取并返回给用户；若没有权限或地址错误，则分别返回相应的错误信息。整个过程中，磁盘指针会被重新定位到原来的位置。

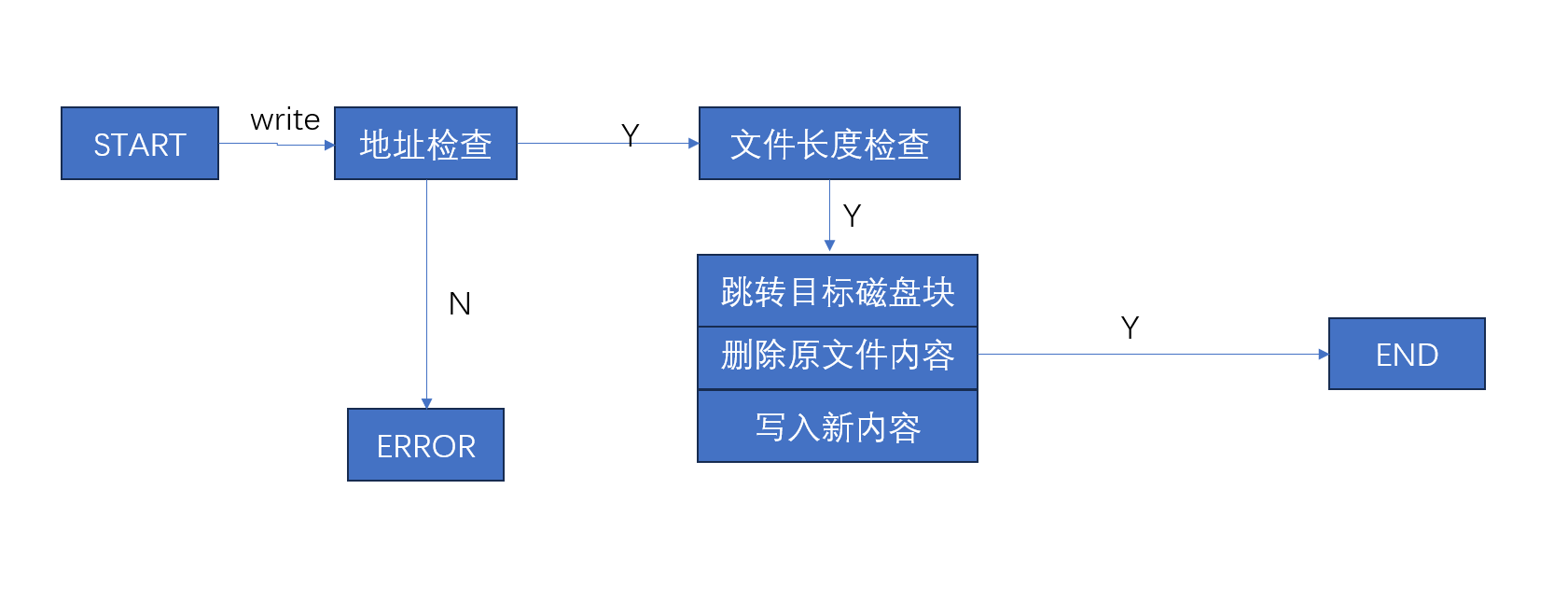
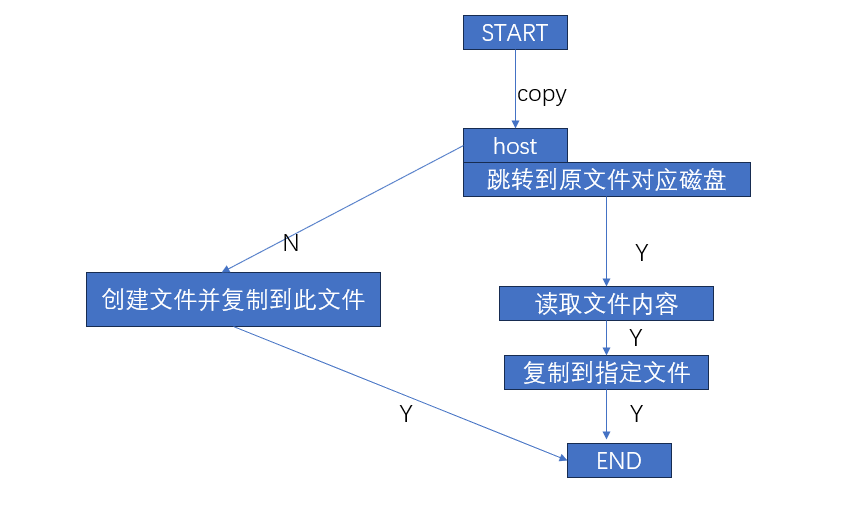


图8

write指令格式为：write+绝对地址+文件内容，用于修改文件。代码实现了用户向指定文件写入内容的操作。首先，用户输入目标文件的地址和待写入的内容。系统检查目标文件是否为普通文件，如果不是，则返回错误信息。如果文件存在，接着检查用户是否有权限写入该文件，并且确保文件内容不超过128个字符。如果用户权限不足或文件内容过多，则会相应返回错误。接着，源文件内容被清空，然后通过延时机制模拟写入过程，并将新的内容写入目标文件。操作完成后，系统返回“写入成功”消息，并将磁盘指针恢复到原位置。



copy函数的输入格式为：copy +源绝对地址+目标绝对地址(+host)。这段代码包含了文件复制操作，涉及多个函数的调用。首先，通过 cat\_HELP 获取源地址（OriAdd）和目标地址（TarAdd）的磁盘块号。如果没有提供额外的参数，系统会调用 GetFileContext 获取源文件内容，并通过 inputcharaddr 将内容写入目标文件。如果目标文件地址无效，系统会返回错误信息。如果参数为 "host"，则通过 GetWindowsFileName 获取文件名，使用 fstream 读取源文件内容，接着调用 CopyNewFile 将内容复制到目标位置。最终，通过 response 返回操作结果。

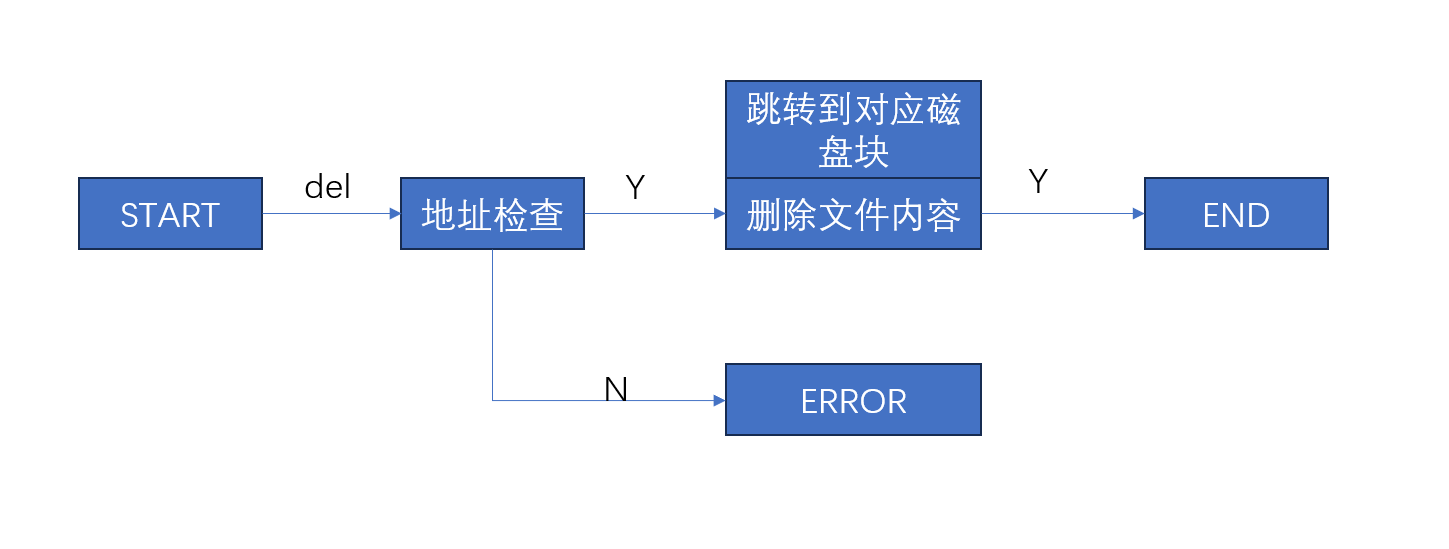


图9

del指令格式为：del +绝对地址，这段代码实现了文件删除功能。首先，它读取用户提供的地址（存储在delBuf中），然后调用cat\_HELP函数获取该文件对应的磁盘块。如果目标文件不是普通文件（通过F\_docTy判断），程序会发送错误消息，并使用jump2whePTR返回到原始磁盘块。如果文件存在（即TarSqe有效），则调用delDoc函数删除该文件，并发送删除成功的消息。如果文件地址无效，则发送地址错误的消息。无论操作结果如何，程序最终都会使用jump2whePTR返回到原始磁盘块。

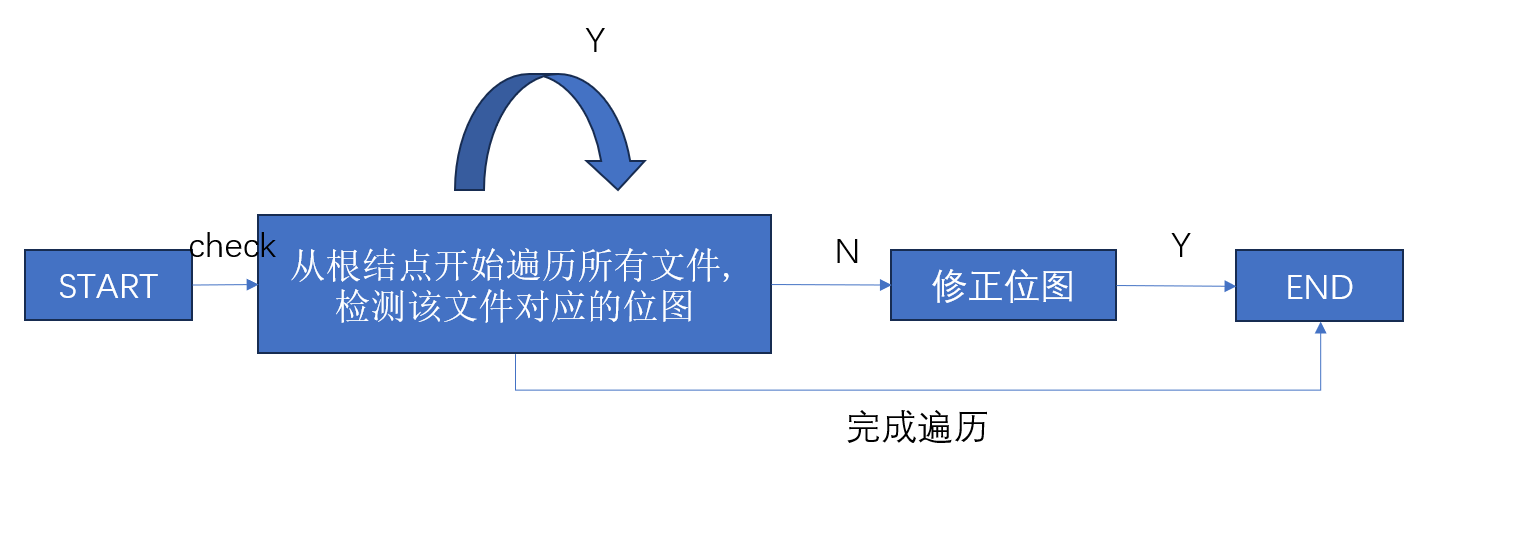
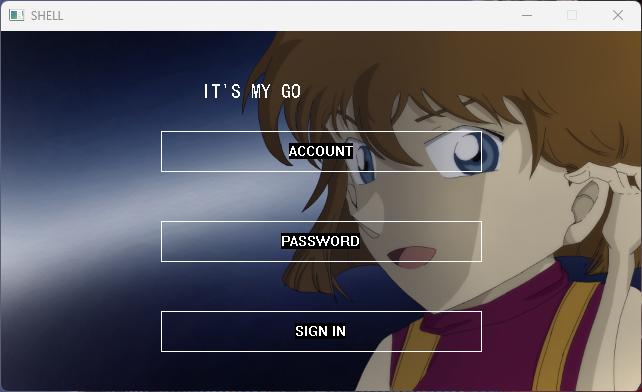


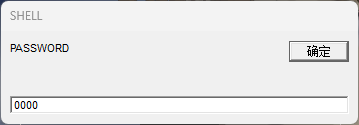
图10

check指令格式:check，这段代码用于执行文件一致性检查操作。首先，它获取当前磁盘指针位置并存储为ORZNUM。然后调用ConsistencyCheck函数，进行文件的一致性检查，检查过程中使用磁盘指针Disk\_Pointer和传递参数101。检查完成后，使用jump2whePTR函数将磁盘指针返回到原始位置ORZNUM，确保磁盘指针指向之前的位置。最后，程序发送一条“文件一致性检查完成！”的消息给用户，提示操作成功。如果命令不符合预期，程序会发送“指令错误！”并继续执行其他操作。

【PARTⅡ】

//SHELL





**void SIGN();**

代码定义了一个用于实现图形界面登录的函数 SIGN()，其功能是展示一个包含账号和密码输入框、以及登录按钮的窗口。首先，代码初始化了一个640x360像素的图形界面，并设置背景为黑色。然后，加载了一张背景图片并将其显示在窗口的顶部位置。接下来，通过绘制矩形框创建了账号输入框、密码输入框和登录按钮，并为这些框添加了对应的文本标签，如“账号”、“密码”和“登录”。

界面上的文本字体使用了“宋体”，字体大小设置为16，并且设置了透明背景以便文字显示更加清晰。除此之外，还在窗口顶部显示了“IT'S MY GO”的提示文字，使用的是楷体字，大小为20。

在界面的交互部分，程序不断循环监听鼠标事件。通过判断鼠标点击的区域，程序识别用户的点击行为。如果用户点击了“登录”按钮（坐标区域在160至480之间，纵坐标在280至320之间），则程序会根据输入的账号和密码做出相应的反应。如果账号和密码正确且符合预设的条件（如账号为“cmd”且密码为“0000”是管理员账号），则进入管理员界面；如果账号为“usr”且密码为“1234”则进入普通用户界面。如果账号和密码不匹配，则弹出一个错误提示框。

此外，程序还为账号框和密码框设置了响应机制，当用户点击这些框时，会弹出输入框供用户输入信息。所有这些事件都通过 getmessage() 函数来处理，以便程序能够实时响应用户操作。

最后，当用户完成登录操作或退出时，调用 closegraph() 关闭图形窗口，结束图形界面的显示。



**void admin\_GUI();**

admin\_GUI() 函数实现了一个管理员交互界面，首先初始化画布并加载背景图像，设置界面为黑色背景。界面包含多个框体，用于显示信息、输入指令以及确认和返回按钮。通过 rectangle() 方法绘制了指令输入框、信息显示框和控制按钮，用户可以在输入框中输入指令并点击确认按钮执行命令。函数通过 getmessage() 获取鼠标点击事件，当用户点击确认按钮时，输入的指令会被处理并通过共享内存与其他模块交互，显示命令执行的结果。返回按钮允许用户返回上一层界面。整个界面交互通过图形界面实现，增强了用户体验。

**void user\_GUI();**

user\_GUI() 函数实现了一个普通用户的交互界面，类似于管理员界面，但提供的功能不同。首先，该函数初始化画布并加载背景图像，同时设置界面为黑色背景。它包含多个框体，用于显示信息、输入指令以及控制按钮。用户可以在输入框中输入指令，点击确认按钮执行命令。指令会通过共享内存进行交互，与系统的其他部分进行数据交换。

在确认按钮点击时，程序会判断输入指令是否涉及到“write”操作，并检查当前文件是否被其他用户写入，如果是，则会提示用户稍后再试。然后，程序通过共享内存获取执行命令后的反馈信息，并在界面上显示相应的结果。如果反馈信息需要换行，界面会自动调整显示的内容。

界面还包括一个返回按钮，点击后可以返回到上一级界面。所有操作都通过图形界面完成，并通过鼠标事件处理器 getmessage() 监听用户输入。

**void TELLerror();**

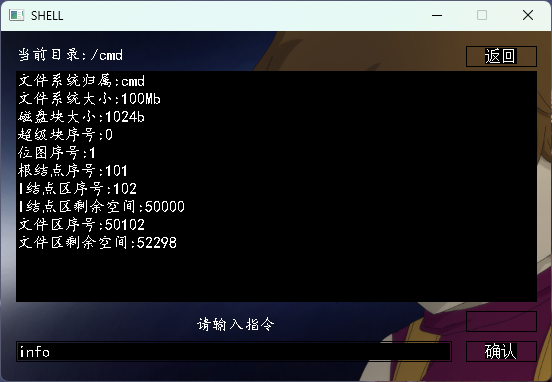
TELLerror() 函数用于在图形界面中显示一个错误提示，并在短暂的时间后自动清除提示。首先，函数通过 settextstyle(12, 0, "宋体") 设置了字体样式为“宋体”，字号为12，确保错误提示文字具有清晰可读的格式。接着，settextcolor(WHITE) 将文字颜色设为白色，以便在黑色背景下清楚地显示错误信息。随后，outtextxy(200, 157, "ERROR") 将“ERROR”文本输出到坐标（200, 157）处，向用户显示错误提示。为了避免后续绘制干扰，settextcolor(BLACK) 将文本颜色改为黑色。接下来，Sleep(300) 会让程序暂停300毫秒，让用户有足够时间看到错误提示。最后，使用 clearrectangle(200, 157, 300, 170) 清除错误提示区域，以便界面恢复原样，确保提示信息不再显示。整体来说，这个函数通过短暂的错误提示反馈，帮助用户意识到发生了错误并自动清除信息，提升了用户体验。

**void SMemoCL(LPVOID& ShareMemoryPointer);**

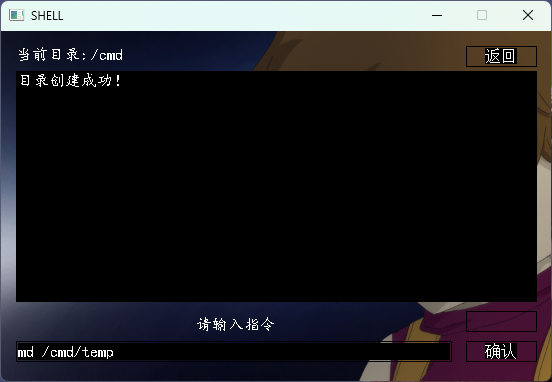
**void SMemoinfoCY(char\* Buffer, LPVOID& ShareMemoryPointer);**

代码包含两个函数：SMemoCL 和 SMemoinfoCY，用于操作共享内存。SMemoCL 清空共享内存中的数据，通过将一个空字符串写入共享内存实现。SMemoinfoCY 则将共享内存中的数据复制到本地缓冲区，并在复制后调用 SMemoCL 清空共享内存，确保数据被提取后不会被重复读取或泄漏。

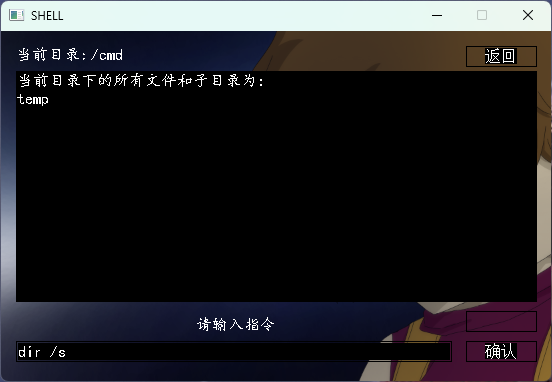
【info】



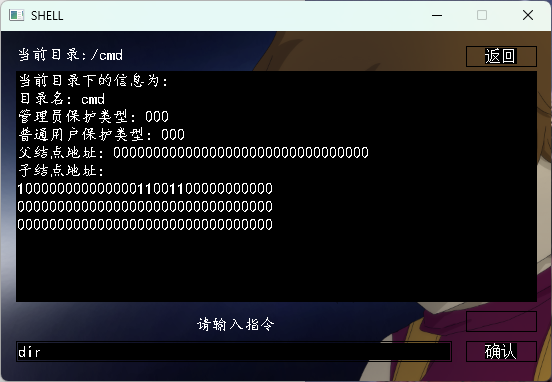
【md】



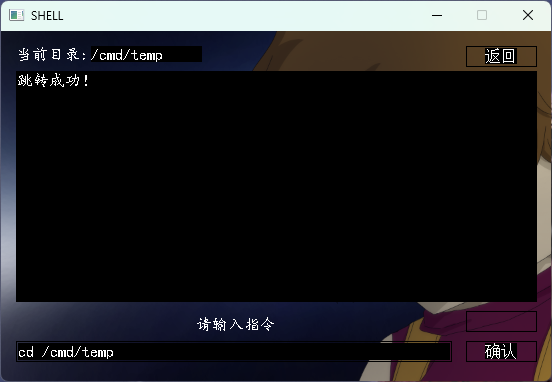
【dir /s】



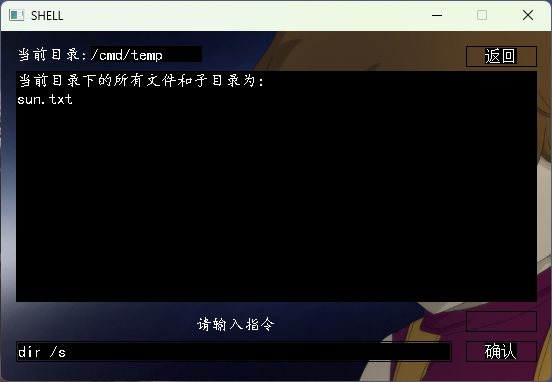
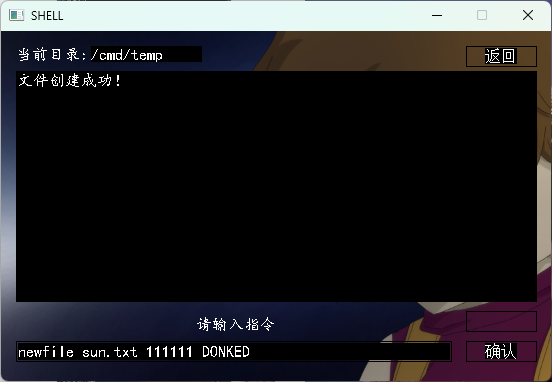
【dir】



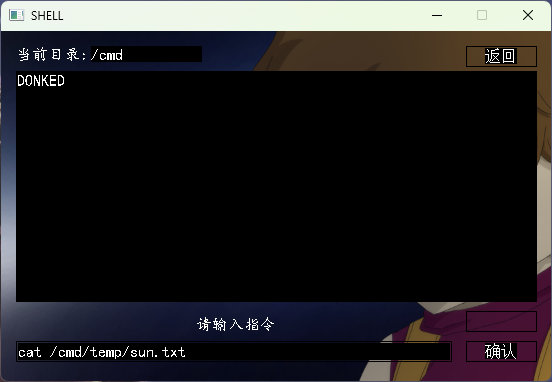
【cd】



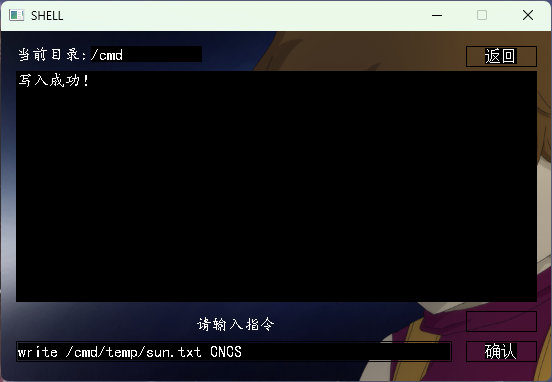
【newfile】

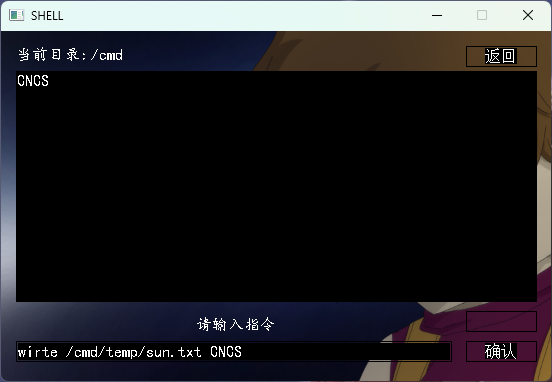


【cat】

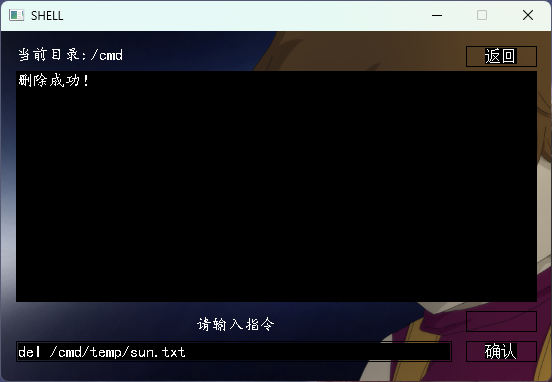


【write】

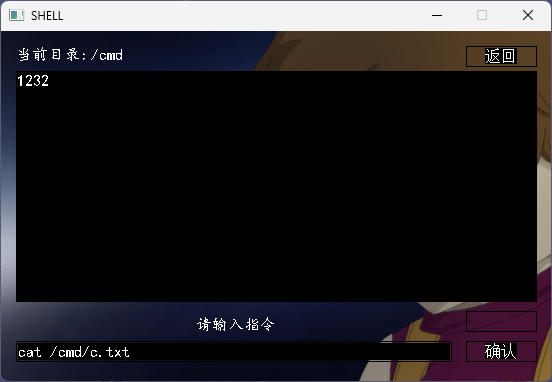


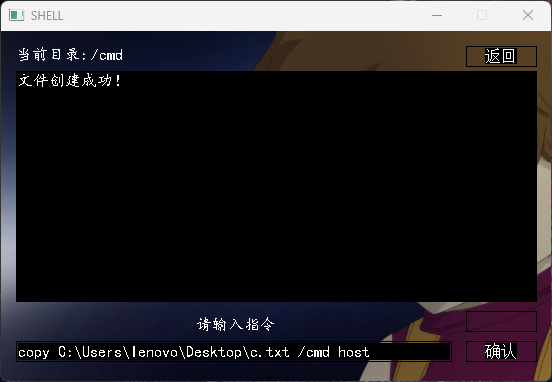


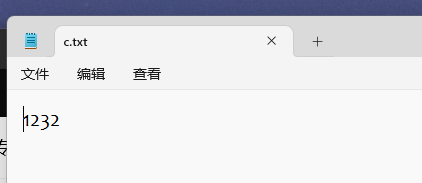
【del】



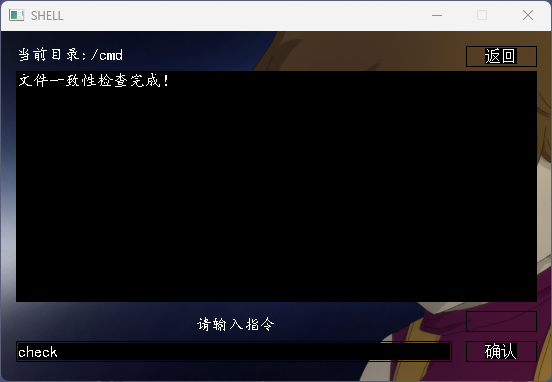
【copy】







【check】



【PARTⅢ】

【文件安全】

设计两份交互过程，操作大致相同，但是管理员允许更多操作，用这种办法使得文件基础权限的保护变得更安全，其中用字符信号区分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。

【进程同步】

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空间中；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”；总体上并没有实现多进程并行的操作，但是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。

**SIMDISK函数的实现目录：**

|  |  |
| --- | --- |
| 黄色 | 翻译函数：将二进制文件和文本文件进行交流 |
| 橙色 | 基本函数：负责基本的调用和查找操作，使二进制文件可使用 |
| 蓝绿色 | 基本的指令函数：用于对应指令 |

|  |  |  |
| --- | --- | --- |
| SEQ | **Function** | tips |
|  | **int\* char2\_8int(char Char)** | 将符号转换为8位二进制整数数组，用于输入文本 |
|  | **char int\_82char(int\* tt)** | 将8位二进制整数数组转换为符号，用于输出文本 |
|  | **Int ptrnum2blocknum(int PointerSqe)** | 将指针序号转换为磁盘块序号 |
|  | **int\* blocknum2addr(int blocknum)** | 将磁盘块序号转换为32位地址 |
|  | **int addr2blocknum(int\* Add)** | 将32位地址转换为磁盘块序号 |
|  | **void Init(fstream& Disk\_Pointer)** | 初始化函数，用于将硬盘进行初始化 |
|  | **int getBLOCK(fstream& Disk\_Pointer)** | 获取对应指针的当前磁盘块序号 |
|  | **int document\_match(fstream& Disk\_Pointer, int blocknum, 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，返回磁盘块序号；如果不匹配，返回0 |
|  | **void jump2whePTR(fstream& Disk\_Pointer, int blocknum, int BlockIn)** | 将指针跳转到指定位置。其中blocknum为磁盘块序号，BlockIn为块内地址 |
|  | **void response(string ReplyInfo, LPVOID& ShareMemoryPointer)** | 用于拷贝回复的信息 |
|  | **void SMemoCL(LPVOID& ShareMemoryPointer)** | 清空共享内存 |
|  | **void SMemoinfoCY(char\* Buffer, LPVOID& ShareMemoryPointer)** | 将共享内存中的内容拷贝出来 |
|  | **void bitmap\_engine(fstream& Disk\_Pointer, int blocknum, int Tag)** | 将位图区域的某一个值置1或者置0 |
|  | **void inputintaddr(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 inputcharaddr(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 blocknum, 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 InputParNode(fstream& Disk\_Pointer, int blocknum, 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 blocknum, string Acc)** | 将保护类型输入到硬盘中 |
|  | **void inputcontT(fstream& Disk\_Pointer, int blocknum, 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 InputFileType(fstream& Disk\_Pointer, int blocknum, int FileType)** | 将文件类型输入到磁盘中 |
|  | **int\* OP32b(fstream& Disk\_Pointer)** | 输出32位二进制数 |
|  | **int\* OP8b(fstream& Disk\_Pointer)** | 输出8位二进制数 |
|  | **int F\_spareInode(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 F\_sparedoc(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 F\_sparechildnode(fstream& Disk\_Pointer, int blocknum)** | 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 |
|  | **char\* F\_docNa(fstream& Disk\_Pointer, int blocknum)** | 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 |
|  | **int F\_docTy(fstream& Disk\_Pointer, int blocknum)** | 寻找某个磁盘块对应的文件类型 |
|  | **int\* F\_docSACC(fstream& Disk\_Pointer, int blocknum)** | 寻找某个磁盘块对应的系统保护类型 |
|  | **int\* F\_docUACC(fstream& Disk\_Pointer, int blocknum)** | 寻找某个磁盘块对应的普通用户保护类型 |
|  | **int\* F\_docFAddr(fstream& Disk\_Pointer, int blocknum)** | 寻找某个磁盘块对应的父结点地址 |
|  | **int\*\* F\_docCHILDddr(fstream& Disk\_Pointer, int blocknum)** | 寻找某个磁盘块对应的子结点地址 |
|  | **int\* F\_doccontnode(fstream& Disk\_Pointer, int blocknum)** | 寻找某个磁盘块对应的内容的地址 |
|  | **int G\_InodeLspace(fstream& Disk\_Pointer)** | 获取i结点的剩余空间--info |
|  | **int G\_docLspace(fstream& Disk\_Pointer)** | 获取文件区的剩余空间--info |
|  | **string cd\_HELP(fstream& Disk\_Pointer, string AddBuf)** | 将指针跳转至对应路径的目录，并返回路径；如果路径不正确，返回错误信息--cd |
|  | **void ShowAllChildNodeName(fstream& Disk\_Pointer, int blocknum, string& ReplyInfo)** | 用于寻找目标磁盘块的所有子结点的名字，并将相应的信息录入到ReplyInfo中--dir |
|  | **void ShowAllInfo(fstream& Disk\_Pointer, int blocknum, string& ReplyInfo)** | 将对应磁盘块的所有信息返回到ReplyInfo中--dir |
|  | **int dir\_HELP(fstream& Disk\_Pointer, string AddBuf)** | 将指针跳转至对应路径的目录；如果路径正确，返回对应文件的磁盘块序号；如果路径不正确，返回0--dir |
|  | **string mdAddLocation(fstream& Disk\_Pointer, string AddBuf)** | 将指针跳转至对应路径的目录，并返回需要创建的文件名；如果路径不正确，返回一个空串--md |
|  | **int IsDirEmpty(fstream& Disk\_Pointer, int blocknum)** | 判断一个目录磁盘块是否为空 |
|  | **int IsFileEmpty(fstream& Disk\_Pointer, int blocknum)** | 盘块一个文件磁盘块是否为空 |
|  | **int rd\_HELP(fstream& Disk\_Pointer, string AddBuf)** | 将指针跳转至对应路径的目录，并返回路径；如果路径不正确，返回错误信息 |
|  | **void rdDir(fstream& Disk\_Pointer, int blocknum)** | 删除一个目录，如果有子目录，一起删了 |
|  | **string FileNameTest(string FileName)** | 判断一个文件名的格式是否正确 |
|  | **string FileAccTest(string Acc)** | 判断保护类型的格式是否正确 |
|  | **string newfile\_HELP(fstream& Disk\_Pointer, string AddBuf)** | 将指针跳转至对应路径的目录，并返回需要创建的文件名；如果路径不正确，返回一个空串 |
|  | **int cat\_HELP(fstream& Disk\_Pointer, string AddBuf)** | 将指针跳转至对应路径的目录，并返回路径；如果路径不正确，返回错误信息 |
|  | **string GetFileContext(fstream& Disk\_Pointer, int blocknum)** | 获取对应磁盘块的内容 |
|  | **void delDoc(fstream& Disk\_Pointer, int blocknum)** | 删除一个文件（指定磁盘块）。 |
|  | **string GetWindowsFileName(string path)** | 获取windows下一个文件的文件名 |
|  | **string CopyNewFile(fstream& Disk\_Pointer, istringstream& Buf)** | 在Copy指令下新建一个文件 |
|  | **void ConsistencyCheck(fstream& Disk\_Pointer, int blocknum)** | 一致性检查 |

**翻译函数**

int\* char2\_8int(char Char);//将符号转换为8位二进制整数数组

char int\_82char(int\* EightInt);//将8位二进制整数数组转换为符号

int\* blocknum2addr(int BlockSqe);//将磁盘块序号转换为32位地址

int addr2blocknum(int\* Add);//将32位地址转换为磁盘块序号

该实现包括字符与二进制数组、地址与磁盘块序号的相互转换。char2\_8int 将字符转为 ASCII 值后生成 8 位二进制数组，int\_82char 将二进制数组转为对应字符。ptrnum2blocknum 通过指针序号除以 1024 计算磁盘块号，addr2blocknum 提取 32 位地址中的 17 位转为磁盘块号，blocknum2addr 则将磁盘块号转为 32 位二进制地址。

**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

Init 函数用于初始化文件系统。首先，在磁盘块 101 处初始化根目录，设置名称为 "cmd"，类型为目录，无父节点、子节点和内容地址。然后初始化超级块区，写入根目录信息、根目录地址、数据区起始地址等关键系统数据，按特定偏移依次写入。最后，初始化位图区，将前 101 个磁盘块标记为已使用（1），其余磁盘块标记为未使用（0），以记录磁盘空间的占用状态。

**int GetBlock(fstream& Disk\_Pointer);/**/获取对应指针的当前磁盘块序号

getBLOCK 函数通过获取当前文件流指针的位置（tellg()），然后调用 ptrnum2blocknum 函数将该指针位置转换为对应的磁盘块序号并返回。具体来说，tellg() 获取文件流的当前读写位置（即指针序号），再通过除法将其转化为对应的磁盘块序号，以便进一步的磁盘操作。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 document\_match(fstream& Disk\_Pointer, int blocknum, string DocName)**

document\_match 函数用于在指定磁盘块中查找与给定文件名匹配的子节点。首先，通过 getBLOCK 获取当前文件指针的位置并保存，接着循环三次检查每个子节点（根据信息子节点最多为3个）。在每次循环中，使用 jump2whePTR 函数跳转到当前子节点的位置，读取该子节点的地址并转换为磁盘块号，然后使用 F\_docNa 函数查找该子节点的文件名。如果找到的文件名与目标文件名匹配，函数会跳转回原始位置并返回匹配的子节点的磁盘块号。如果循环结束后仍未找到匹配项，则返回 0，表示未找到该文件。

**void jump2whePTR(fstream& Disk\_Pointer, int blocknum, int BlockIn);**

jump2whePTR 函数通过调用 seekg 来设置磁盘文件流的读取位置。具体地，它将文件指针移动到由 blocknum 和 BlockIn 决定的目标位置。blocknum 表示目标磁盘块的序号，乘以1024得到块的起始位置；BlockIn 是该块内的偏移量，表示相对于该块的起始位置的具体位置。通过这两个参数的组合，函数能够精确地将文件指针定位到磁盘中的某个具体地址，从而方便对该位置的数据进行读取或写入。

**void response(string ReplyInfo, LPVOID& ShareMemoryPointer);**

response 函数用于将指定的回复信息 ReplyInfo 发送到共享内存区域，并同时输出该信息。具体来说，函数使用 strcpy\_s 将 ReplyInfo 的内容复制到共享内存指针 ShareMemoryPointer 指向的内存位置。strlen(ReplyInfo.c\_str()) + 1 确保包含字符串结束符（\0）。此外，函数还通过 cout 输出回复信息，方便调试和显示返回的内容。

**void SMemoCL(LPVOID& ShareMemoryPointer);**

SMemoCL 函数的功能是清空共享内存中的内容。它通过将一个空字符串（NULLChar[] = ""）复制到共享内存指针指向的内存区域，实现将该内存区域的内容设置为空字符串。函数首先定义一个空字符串 NULLChar，然后使用 strcpy\_s 函数将该空字符串复制到 ShareMemoryPointer 指向的内存中，确保安全地进行内存操作。

**void SMemoinfoCY(char\* Buffer, LPVOID& ShareMemoryPointer);**

SMemoinfoCY 函数的作用是将共享内存中的内容复制到提供的 Buffer 中，并随后清空共享内存。具体来说，函数通过 strcpy\_s 将共享内存指针 ShareMemoryPointer 指向的数据复制到 Buffer 中，复制的长度是共享内存内容的字符串长度加1（以确保包括字符串结束符）。在复制完内容后，函数调用 SMemoCL 来清空共享内存，释放其中的数据，确保下一次使用时没有残留的内容。

**void bitmap\_engine(fstream& Disk\_Pointer, int blocknum, int Tag);**

bitmap\_engine 函数的作用是操作位图中的特定位，将该位置的值设置为给定的 Tag（通常为0或1）。函数的步骤如下：

使用 getBLOCK 函数获取当前磁盘指针的位置（即当前磁盘块序号）。

调用 jump2whePTR 将指针跳转到指定的位图区的 blocknum 位置，这里 blocknum 是传入的磁盘块号。

通过 Disk\_Pointer << Tag 将 Tag 值（0 或 1）写入到位图的当前位置，表示该磁盘块的状态。

再次调用 jump2whePTR 跳转回原来的磁盘块位置，恢复指针的初始状态，确保后续操作不受影响。

这个函数通常用于在磁盘位图中标记磁盘块的使用情况，例如将某个磁盘块标记为已使用或未使用。

**输入函数**

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

inputcharaddr 函数的作用是将一个字符数组（CharArr）中的每个字符依次输入到磁盘中。具体来说，它遍历 CharArr 中的每个字符，并调用 InputChar 函数将每个字符写入磁盘文件。该函数使用了 strlen 函数来获取 CharArr 的长度，并通过 for 循环逐一处理字符。每次处理一个字符，都会调用 InputChar 将该字符转换为二进制形式并写入磁盘。

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

InputChar 函数的作用是将一个字符 Char 转换为其对应的 8 位二进制表示，然后将这些二进制数据写入磁盘。

具体过程如下：

函数首先创建了一个大小为 8 的整数数组 InputInt，用于存储字符的二进制表示。

通过调用 char2\_8int(Char) 函数，将字符 Char 转换为其 ASCII 值的 8 位二进制形式，并将结果存储到 InputInt 数组中。

然后，调用 inputintaddr 函数将这个 8 位的二进制数据写入磁盘。

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

inputintaddr 函数的作用是将一个整数数组 IntArr 中的数据按顺序写入磁盘文件。

具体过程如下：

函数接受两个参数：

Disk\_Pointer：一个 fstream 类型的对象，用于读取或写入磁盘文件。

IntArr：一个整数数组，包含要写入磁盘的数据。

Length：数组的长度，指定要写入的整数个数。

函数使用一个循环遍历数组 IntArr 中的每个元素，并使用 Disk\_Pointer 将每个元素写入磁盘。

Disk\_Pointer << IntArr[i]; 会将 IntArr 中的第 i 个元素写入磁盘。

**void InputDocName(fstream& Disk\_Pointer, int blocknum, 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。

InputDocName 函数的作用是将文件名 DocName 写入到指定的磁盘块 blocknum 中。

具体过程如下：

ORZNUM 保存当前磁盘块的位置，通过 getBLOCK(Disk\_Pointer) 获取。

jump2whePTR(Disk\_Pointer, blocknum, 0) 将文件指针跳转到指定的磁盘块 blocknum 的起始位置。

获取文件名的长度，并通过 new 关键字为临时字符数组 CharDocName 分配内存空间。

使用 strcpy\_s 将传入的文件名 DocName 的内容拷贝到 CharDocName 数组中。

调用 inputcharaddr(Disk\_Pointer, CharDocName) 将文件名字符数组写入磁盘。

最后，通过 jump2whePTR(Disk\_Pointer, ORZNUM, 0) 恢复文件指针的位置。

**void InputParNode(fstream& Disk\_Pointer, int blocknum, 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。

InputParNode 函数的作用是将父节点信息写入指定磁盘块中的指定位置。

具体过程如下：

ORZNUM 变量保存当前磁盘块的位置，通过 getBLOCK(Disk\_Pointer) 获取。

jump2whePTR(Disk\_Pointer, blocknum, 80) 将文件指针跳转到指定的磁盘块 blocknum 中，从第 80 个字节开始的位置。这里假设父节点信息的存储位置是偏移量为 80 的位置。

调用 inputintaddr(Disk\_Pointer, IntArr, 32) 将传入的整数数组 IntArr 写入到该位置，写入的数据长度为 32（假设父节点信息有 32 个整数）。

最后，通过 jump2whePTR(Disk\_Pointer, ORZNUM, 0) 恢复文件指针回到原先的位置。

该函数通常用于更新某个块中记录父节点信息的数据结构。

**void InputAcc(fstream& Disk\_Pointer, int blocknum, string Acc);**//将保护类型输入到硬盘中

InputAcc 函数的作用是将指定的账号信息（Acc）写入指定磁盘块中的特定位置。首先，函数通过 getBLOCK(Disk\_Pointer) 获取当前磁盘块的位置，并将文件指针跳转到目标位置（第 72 字节）。接着，它创建一个大小为 8 的整数数组 TempInputIntArr，并通过两次循环填充该数组：前 3 个字符填充数组的索引 1 到 3，后 3 个字符填充索引 5 到 7（每个字符通过减去 '0' 来转换为整数）。然后，调用 inputintaddr(Disk\_Pointer, TempInputIntArr, 8) 将处理后的整数数组写入磁盘。最后，文件指针被恢复到原来的位置。此函数确保账号信息以正确的格式存储在磁盘的指定位置。

**void inputcontT(fstream& Disk\_Pointer, int blocknum, 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

inputcontT 函数的作用是将指定的文件内容（Context）写入磁盘的特定位置，并更新相关的文档块信息。首先，函数通过 getBLOCK(Disk\_Pointer) 获取当前磁盘块的位置，并将文件指针跳转到目标块（blocknum）的第 208 字节位置。接着，它调用 c c写入磁盘。然后，文件指针跳转到 NewDocBlockSqe 位置，并通过 inputcharaddr(Disk\_Pointer, const\_cast<char\*>(Context.c\_str())) 将文档内容 Context 写入磁盘。最后，函数通过 jump2whePTR(Disk\_Pointer, ORZNUM, 0) 恢复文件指针的位置。此函数确保文档的内容被正确地写入并且相应的文档块地址被更新。

**void InputFileType(fstream& Disk\_Pointer, int blocknum, int FileType);**//将文件类型输入到磁盘中

InputFileType 函数的作用是将文件的类型信息写入磁盘中的指定位置。首先，它通过 getBLOCK(Disk\_Pointer) 获取当前磁盘块的位置，并保存为 ORZNUM。然后，它将文件指针跳转到 blocknum 位置的第 71 字节处，接着通过 Disk\_Pointer << FileType 将文件类型（FileType）写入该位置。最后，文件指针被恢复到原来的位置（ORZNUM）以保证文件操作的正确性。此函数用于更新指定磁盘块中的文件类型信息。

**输出函数**

**int\* OP32b(fstream& Disk\_Pointer);**

**int\* OP8b(fstream& Disk\_Pointer);**

这两个函数 OP32b 和 OP8b 分别用于从文件流 Disk\_Pointer 中读取 32 位和 8 位的二进制数据，并将其转换为整数数组返回。函数通过动态分配内存创建一个整数数组，逐字节读取字符数据（chartpt\_buf），将其 ASCII 值转换为数字（通过减去字符 '0' 的 ASCII 值，即 48），然后存储到对应数组中。最后，函数返回存储结果的数组，用于后续的逻辑处理。

**查找函数**

**int F\_spareInode(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。

F\_spareInode 函数用于在磁盘文件位图中查找空闲的 i 节点。首先保存当前指针位置，然后跳转到位图区 i 节点部分的起始位置（块 1 偏移 102）。从索引 102 遍历到 50102，逐字节读取，判断其是否为 '0'，表示该 i 节点未被占用。如果找到空闲 i 节点，则返回对应的索引，并恢复指针到初始位置；若遍历结束未找到空闲节点，返回 0 表示无可用 i 节点。

**int F\_sparedoc(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

F\_sparedoc 函数用于在磁盘文件位图中查找空闲的文件块。函数首先保存当前指针位置，然后跳转到位图文件部分的起始位置（块 1 偏移 0）。从索引 50102 开始循环读取每个字节，判断其值是否为 '0'，即表示该块未被使用。如果找到空闲块，函数返回对应的索引值，并恢复文件指针到初始位置；若循环结束仍未找到空闲块，则返回 0 表示无空闲文件块。

**int F\_sparechildnode(fstream& Disk\_Pointer, int blocknum);**//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

F\_sparechildnode 函数用于检查某个磁盘块的三个子结点是否空闲并返回空闲的子结点序号（1、2 或 3）。函数首先保存当前位置，逐步跳转到三个子结点的位置（偏移 112、144、176）。每次读取一个字符判断是否为空闲（'0' 表示空闲）。若找到空闲节点，退回文件指针以便覆盖操作，并返回节点序号。若所有节点都被占用，则返回 0 表示无空闲节点。

**char\* F\_docNa(fstream& Disk\_Pointer, int blocknum);**//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

F\_docNa 函数从指定磁盘块号 blocknum 中读取文件名。首先保存当前指针位置，然后跳转到指定块的起始位置。函数逐字节读取 8 个字符，每个字符通过 OP8b 函数转换为二进制数组，并利用 int\_82char 将二进制数组解析为字符，存储在 DocName 中。读取完成后，恢复指针位置并返回文件名数组 DocName。

**int F\_docTy(fstream& Disk\_Pointer, int blocknum);**//寻找某个磁盘块对应的文件类型

F\_docTy 函数用于获取指定磁盘块中的文件类型信息。函数首先保存当前文件指针位置，然后跳转到目标块的偏移位置 71 处读取文件类型的标志字符，将其转换为整数（通过 int(chartpt\_buf) - 48），最后恢复指针位置并返回文件类型值。

**int\* F\_docSACC(fstream& Disk\_Pointer, int blocknum);**//寻找某个磁盘块对应的系统保护类型

F\_docSACC 函数用于读取指定磁盘块中存储的文件系统访问权限信息。函数首先保存当前指针位置，跳转到目标块偏移位置 73 处，逐字节读取三个字符，将其转换为整数（通过 chartpt\_buf - 48），存入动态分配的 SysAcc 数组中。读取完成后，函数恢复文件指针到初始位置，并返回包含访问权限的整数数组。

**int\* F\_docUACC(fstream& Disk\_Pointer, int blocknum);**//寻找某个磁盘块对应的普通用户保护类型

F\_docUACC 函数用于从指定磁盘块 blocknum 中读取文件的用户访问权限（User Access Control）。函数保存当前指针位置，跳转到块内偏移量为 77 的位置，依次读取 3 个字符，并将其转换为整数存储到 ComAcc 数组中。读取完成后，恢复指针位置并返回 ComAcc 数组。

**int\* F\_docFAddr(fstream& Disk\_Pointer, int blocknum);**//寻找某个磁盘块对应的父结点地址

F\_docFAddr 函数从指定磁盘块 blocknum 中读取一个文件的父节点地址。函数先保存当前磁盘指针位置，随后跳转到该块内的偏移量为 80 的位置。然后，它依次读取 32 个字符，将每个字符转换为整数并存入 ParNodeAdd 数组中。最后，恢复指针位置并返回包含父节点地址的 ParNodeAdd 数组。

**int\*\* F\_docCHILDddr(fstream& Disk\_Pointer, int blocknum);**//寻找某个磁盘块对应的子结点地址 F\_docCHILDddr 函数读取指定磁盘块 blocknum 中的 3 个子节点地址。首先，它通过 getBLOCK 获取当前磁盘指针位置，并保存它。然后，跳转到偏移量为 112 的位置，并为 3 个子节点地址分配内存（每个子节点有 32 个整数）。接着，它依次读取这 3 个子节点的 32 个整数，将每个字符转换为整数并存入 ChildNodeAdd 数组。最后，恢复原始指针位置，并返回存储子节点地址的二维数组 ChildNodeAdd。

**int\* F\_doccontnode(fstream& Disk\_Pointer, int blocknum);**//寻找某个磁盘块对应的内容的地址 F\_doccontnode 函数读取指定磁盘块 blocknum 中存储的 32 个节点地址。首先，函数通过 getBLOCK 获取当前磁盘指针的位置，并保存它。接着，跳转到该块的偏移量 208 的位置，并为 ParNodeAdd 数组分配 32 个整数的内存空间。然后，它依次读取 32 个字符，将每个字符转换为整数，并存入 ParNodeAdd 数组中。最后，函数恢复磁盘指针到原始位置，并返回存储这些节点地址的数组 ParNodeAdd。

**SHELL函数实现目录：**

|  |  |
| --- | --- |
| 灰色 | 基于Easyx的交互界面 |
| 粉色 | 共享内存部分 |

|  |  |  |
| --- | --- | --- |
| SEQ | FUNCTION | TIPS |
|  | void SIGN() | 主函数登录界面 |
|  | void admin\_GUI() | 管理者界面 |
|  | void user\_GUI() | 使用者界面 |
|  | void TELLerror() | 报错 |
|  | void SMemoCL(LPVOID& ShareMemoryPointer) | 清空共享内存 |
|  | void SMemoinfoCY(char\* Buffer, LPVOID& ShareMemoryPointer) | 将共享内存中的内容拷贝出来 |

文档导读：

Part1：讲述文件系统设计结构以及simdisk中的函数设计层面，讲解各项指令的大致实现办法。

Part2：介绍shell中的实现流程，以此来介绍文件系统流程，附带运行结果实现代码。

Part3：简单说明关于进程安全和同步的实现情况，利用微观快速的串行达到宏观低速的并行。

补充：功能函数表格以及实现思路