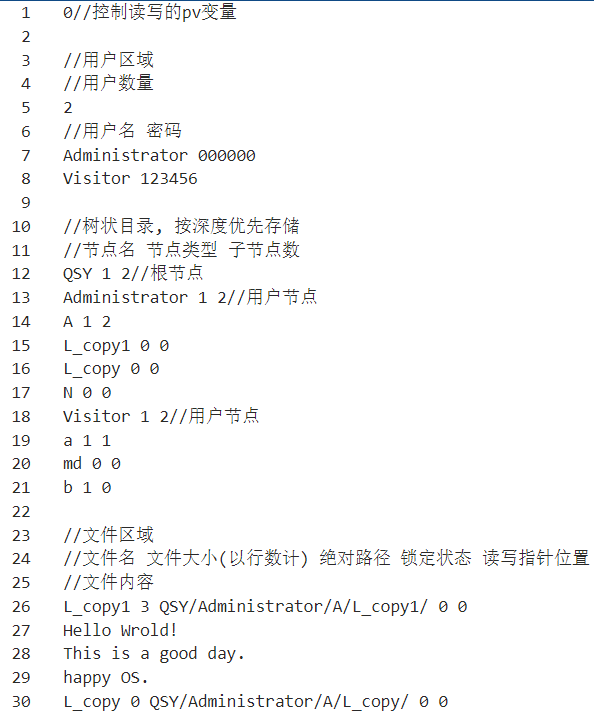
1. **文件系统格式**

数据文件分为4个区域, 公共变量区(进程通信的共享存储区), 用户信息区(存储用户数量和用户密码), 目录区(存储节点的树状结构信息), 文件内容区(存储文件的具体内容).读取时将二进制文件先转化为文本文件, 再进行读取;写入时, 先写入至文本文件再转化至二进制文件. 以文本文件展示的具体存储方式如下所示例:

****

1. **主要数据结构、函数说明**

将数据结构说明和流程图提前至具体的功能实现之前, 以使得之后的功能实现更容易讲解.

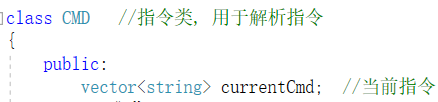
课程设计有四个头文件, Login.h(包括类Login与类User), CMD.h(包括类CMD), Dir.h(包括类Dir与类File), Data.h(包括类Data)

1. Login.h



Login.h主要实现登录系统, User类负责存储用户信息, Login类实现登录系统, 包括登录和注册功能的具体实现. 此外还有一个较重要的函数Login.userAreaCheck(), 负责为每一位新注册或是存在于用户区域却不存在于目录中的用户创建其目录区域.

1. CMD.h



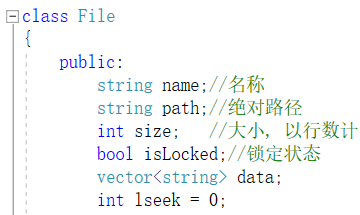
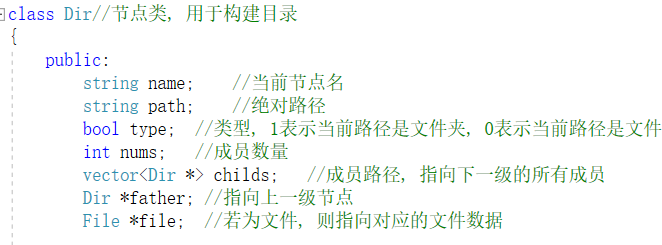
CMD.h主要实现指令相关的功能, 如指令的解析, 和指令执行相关函数的调用.

函数CMD.GetCMD()负责获取指令, 并将其按照空格为界进行拆分.

函数CMD.CheckHelp是help指令的具体实现.

函数CMD.ProcessCmd是指令的执行部分, 负责调用各指令相关的函数.

1. Dir.h

Dir.h主要实现目录相关的功能, 也是本次课程设计的核心部分.

Dir类存储节点信息, 使用指向父级的指针和指向子级的指针来在各节点间移动, 对于文件节点, 则会有一个指向其内容的指针.

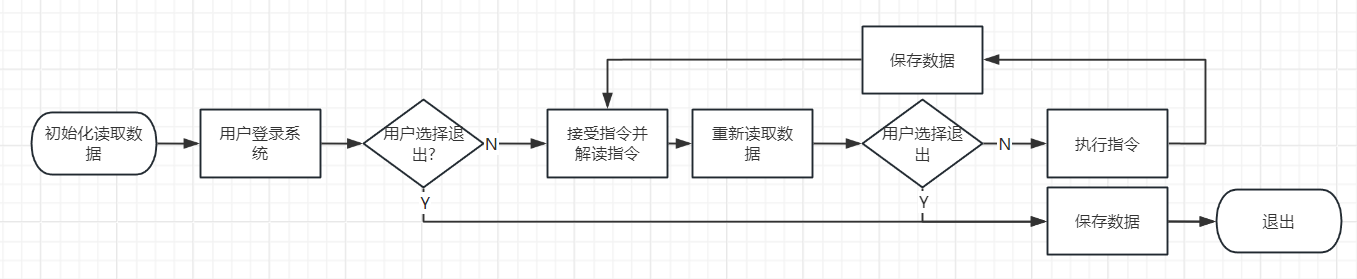
Flie类存储文件内容, 此外也包括文件的各类操作的函数实现.

此外Dir.h中还包括文件读写的pv操作部分和各指令的实际函数实现, 这些函数将在功能实现部分中讲解.

1. Data.h

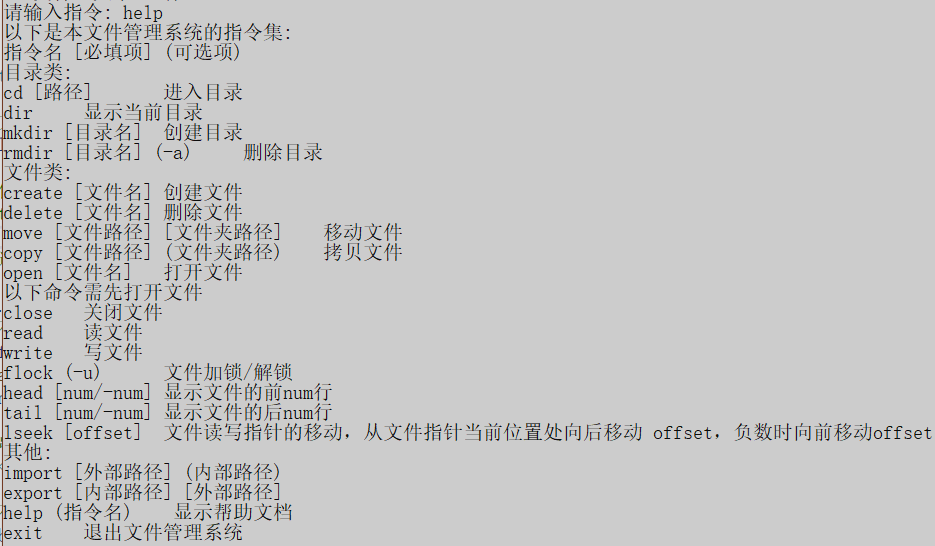
Data.h中的Data类则包括数据的写出和二进制文件与文本文件的相互转化, 不过数据的初始化并不在Data类中, 而是写在各部分类的Init()函数中.

1. **流程图**

****

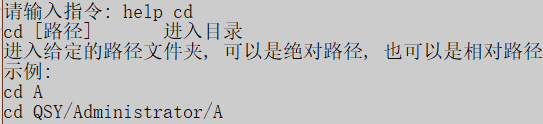
1. **功能实现**
2. **命令**

在所有命令之前, 首先介绍help指令, 单独使用help指令可以查看系统的指令集, 此外, 使用help+指令名可以获取指定指令名的帮助和示例, 接下来的指令讲解将结合help进行.

****

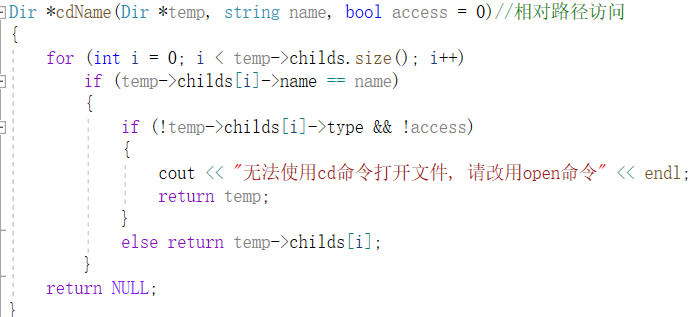
**目录类**

1. cd



cd命令的实现分为两个部分: 当前目录寻址和全局寻址.

当前目录寻址的实现很简单, 遍历当前节点的子级, 寻找有无对应的节点即可.此处的实现还设置了访问类型, 以便区分open和cd指令.

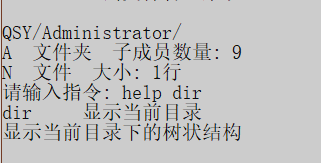


全局寻址则首先需要对绝对地址进行拆分, 这里的处理思路与指令的解读相近, 以/作为分割号分割字符串.得到各级节点名后, 返回根目录进行一级一级的寻址. 此外, 还涉及到根目录和用户级目录的访问控制问题, 不能允许用户使用cd命令绕过登陆系统访问其他用户的文件夹.

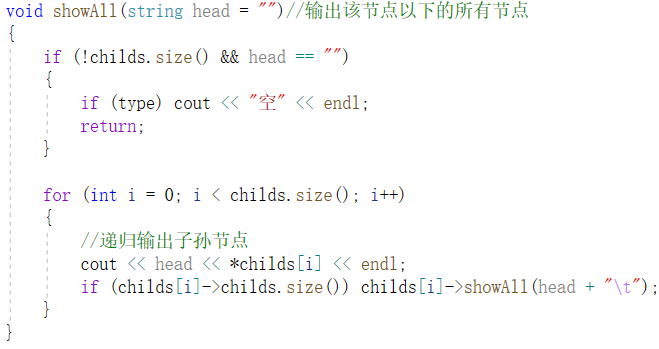


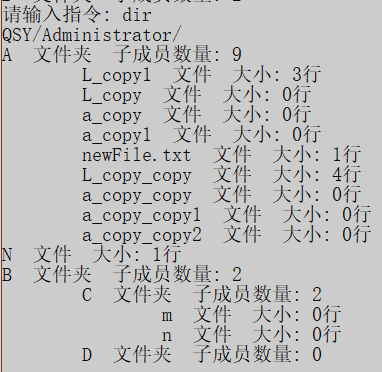
1. dir

这里的实现效果为显示当前目录下的树状结构, 因为显示当前目录路径和子级节点的功能已经设为了每个指令周期都会执行的功能.

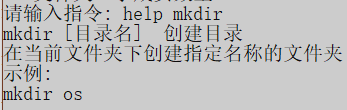


这里的实现思路为递归输出所有子孙节点, 并且函数通过接受一个参数, 实现每一级节点不同缩进的效果.

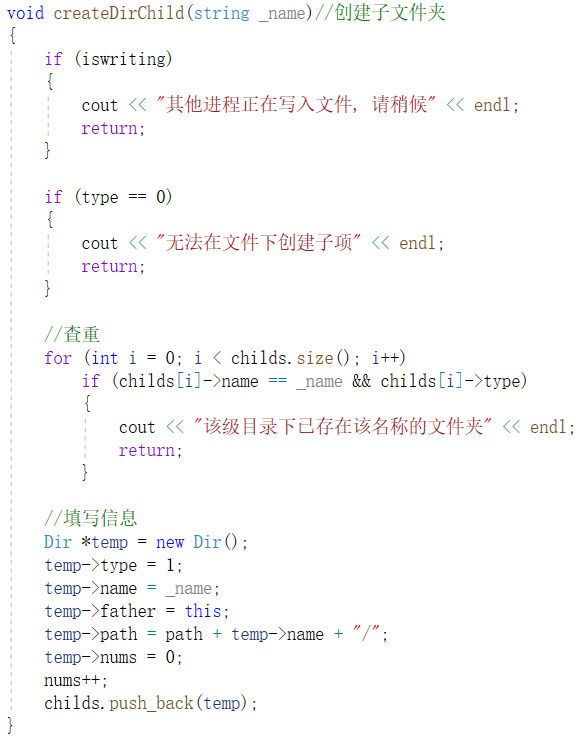




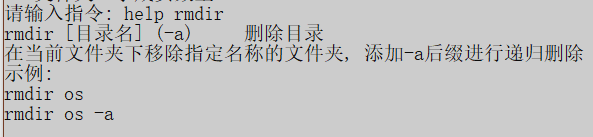
1. mkdir



mkdir在当前文件夹节点下创建一个新的文件夹节点

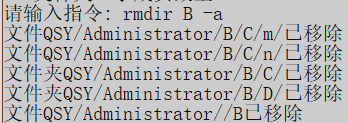


1. rmdir



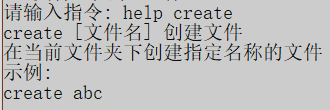
rmdir在不添加-a后缀时, 会直接移除当前文件夹节点, 而添加-a后缀时, 会递归的删除当前文件夹节点及其以下的所有子孙节点



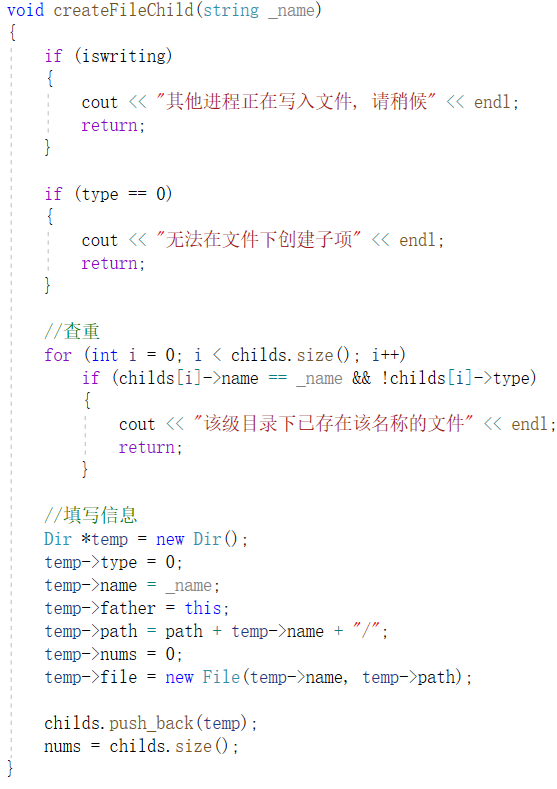


**文件类**

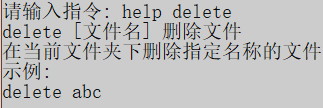
1. create



create指令的实现方法与mkdir相同, 只是创建的节点类型不同



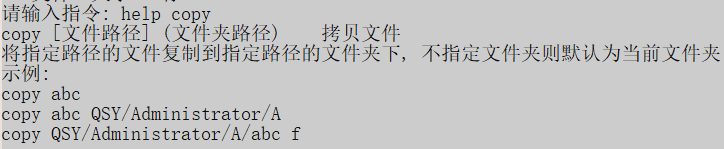
1. delete



delete指令的实现类似于rmdir, 而且不需要考虑子节点

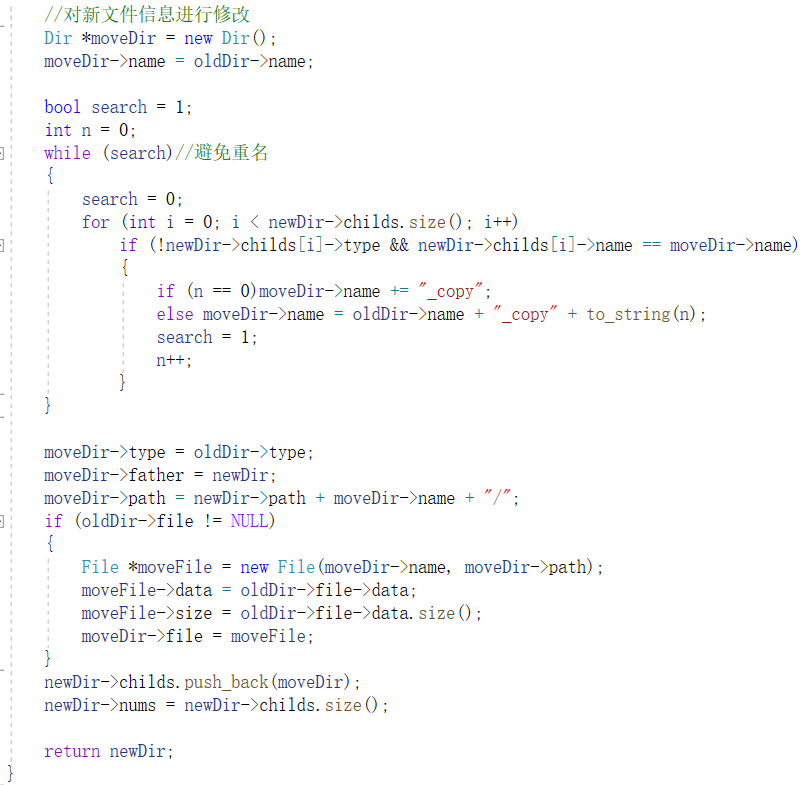


1. copy

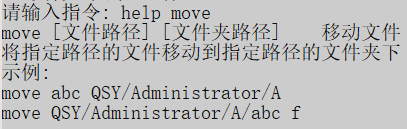


copy指令在目标文件夹下创建一个与目标文件内容相同的文件, 此外还需要注意重名时的命名问题, copy指令会首先给文件加上\_copy后缀, 如何加上后缀后仍重名, 则会继续添加一个数字直至不重名为止.

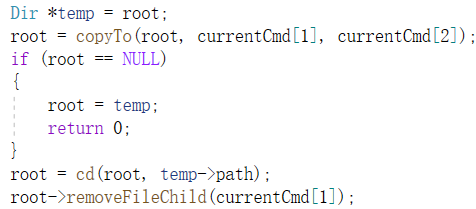




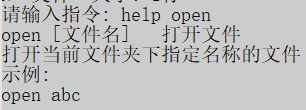
1. move



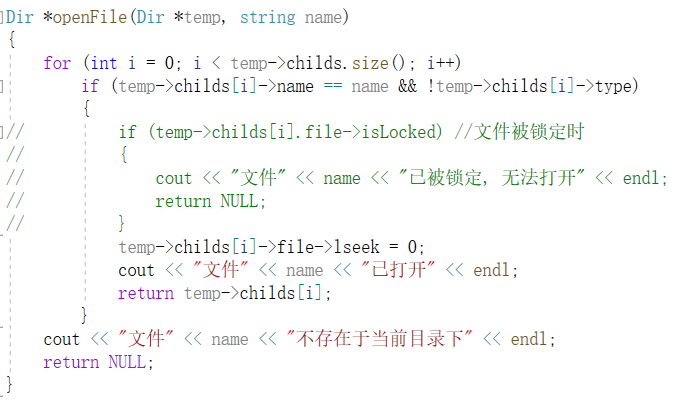
move指令实际上没有独立的函数与之对应, 而是结合了copy指令和delete指令, 因此, 其避免重名的规则也是与copy完全相同的.



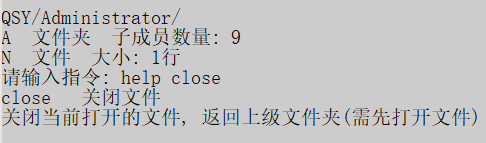
1. open



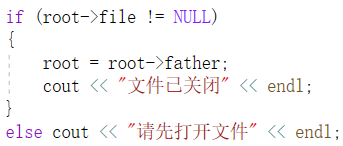
open指令实际上就是开放了文件访问权限的相对路径cd指令



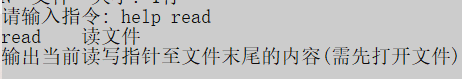
1. close



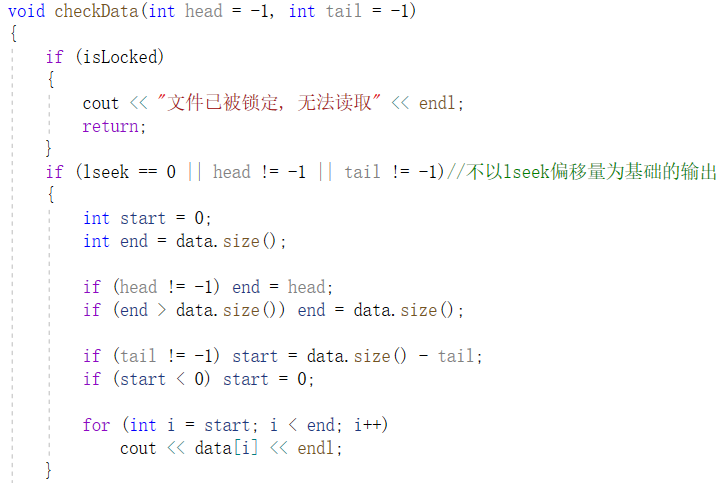
close指令实际上就是返回上级文件夹

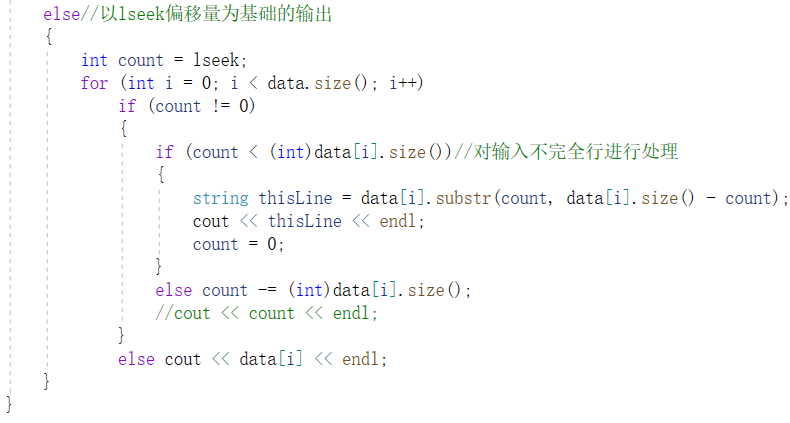


1. read

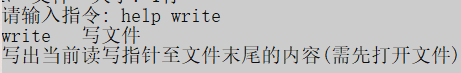


read指令主要要考虑与lseek指令的配合, 需要根据偏移量跳过一些文件的内容, 此外head指令与tail指令也一并集成实现于涉及到的checkData()中.



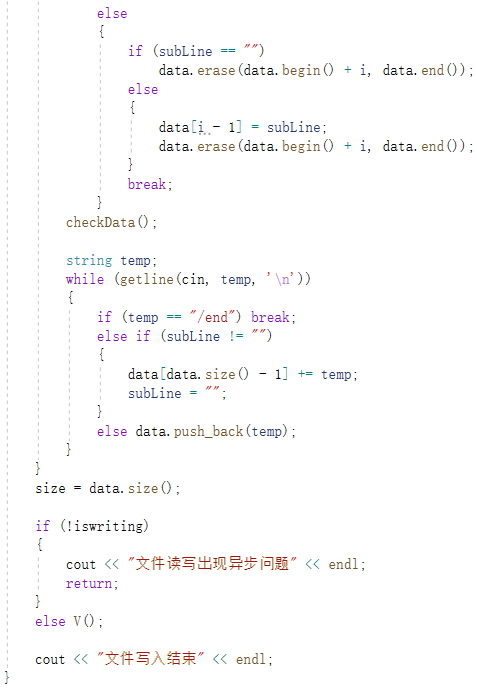


1. write

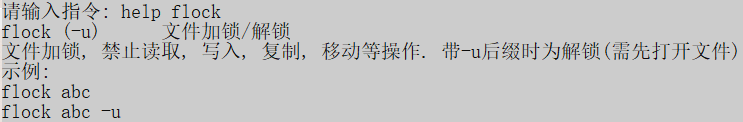


write指令将用户的输入按行读取并写入节点指向的File类, 也需要考虑与lseek的配合, 实现思路与read类似, 只是额外涉及到了行的拼接.

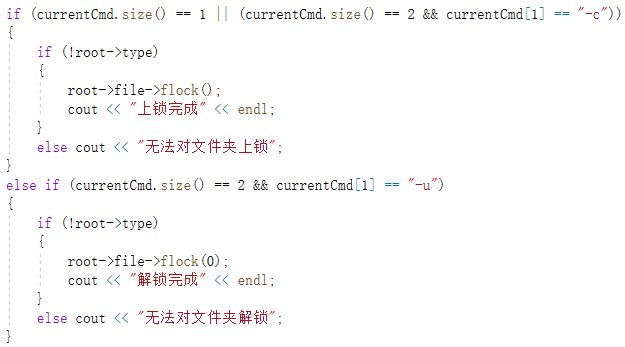




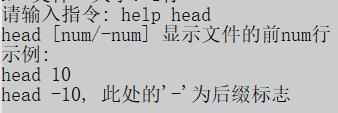
1. flock



flock指令本身只需要切换文件的锁定状态即可, 之后需要在其他涉及到的文件类指令之前添加检测条件, 在文件锁定时不予执行.

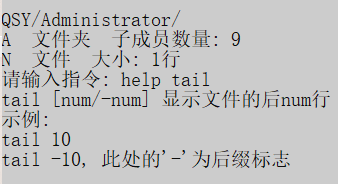


1. head



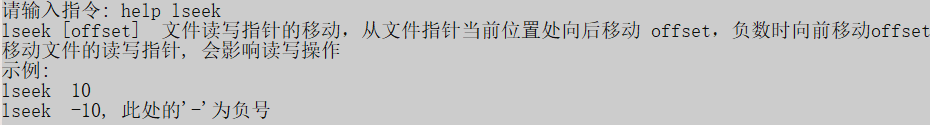
head指令在实现read指令时一并实现.

1. tail

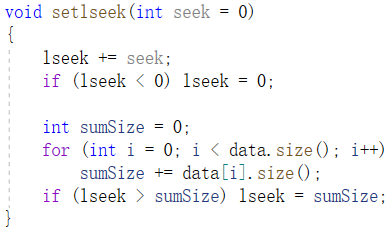


tail指令在实现read指令时一并实现.

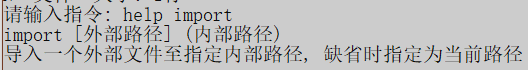
1. lseek



lseek指令涉及的复杂部分都是在其他指令中完成的, 其本身只需要设置Flie类的偏移量属性, 并确保不越界即可.



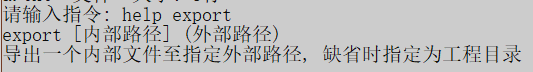
1. **导入导出**
2. import



import指令首先打开一个外部文件, 分离出路径中的文件名并创建新的子文件, 此后按行读取并按行写入子文件节点的文件数据. (此部分测试时导入非本系统导出的txt文件可能出现bug, 推断为换行符编码的不同, 或是在外部的编辑中使用了非换行符进行行切换)



1. export



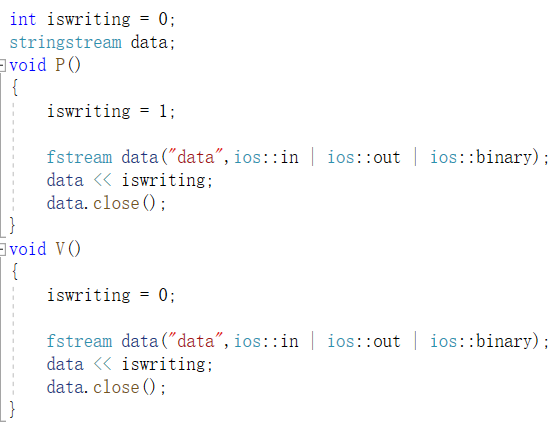
export指令首先创建一个外部文件, 若系统内文件没有后缀, 则会添加”.txt”后缀, 此后按行读取并按行写入外部文件.

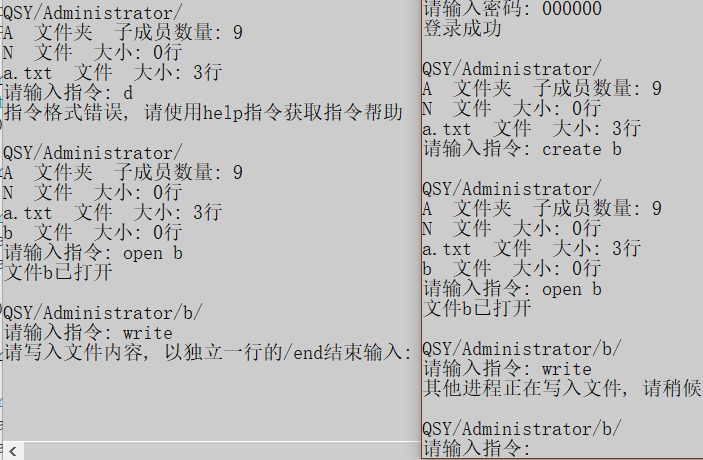


1. **多线程支持**

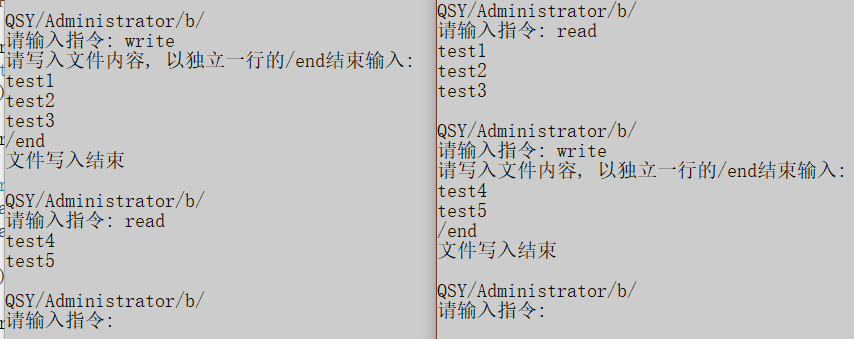
多个进程修改数据的数据共享通过在每个指令周期的前后存取一遍文件实现, 此外使用PV操作来控制所有的写操作(包括节点的创建删除与文件的写入等操作), 避免并发写入的情况发生.

PV变量存储在文件最开头, 便于直接覆写.



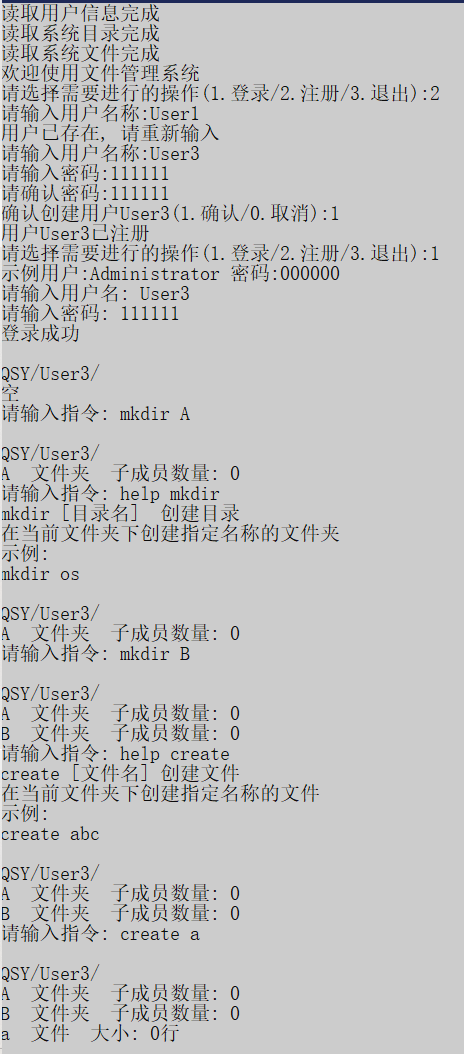


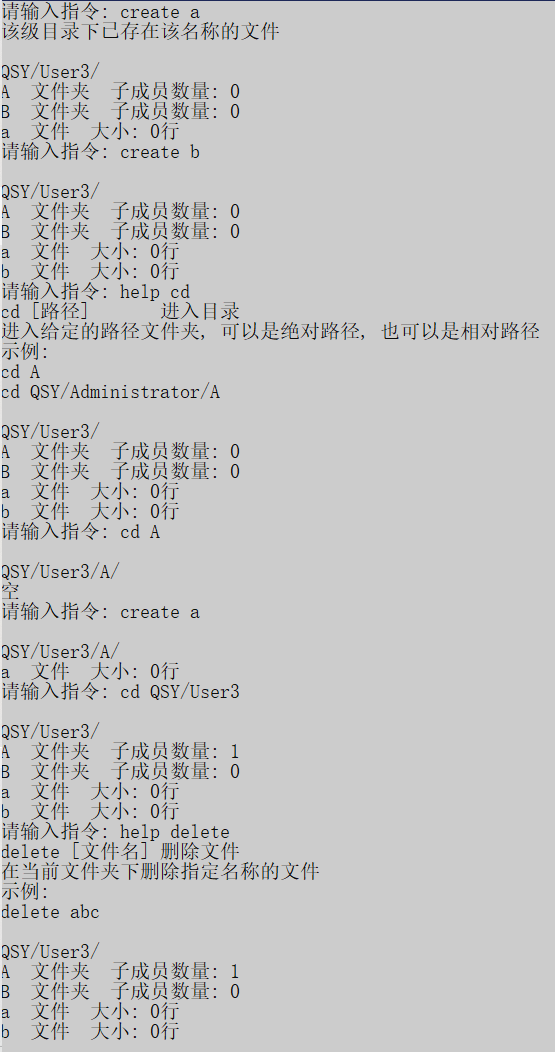
左进程先启动, 右进程创建文件b, 左进程输入错误指令刷新数据, 看到了右进程创建的文件b, 并打开进行写入, 在左进程写入期间, 右进程尝试写入文件b, 被阻止.

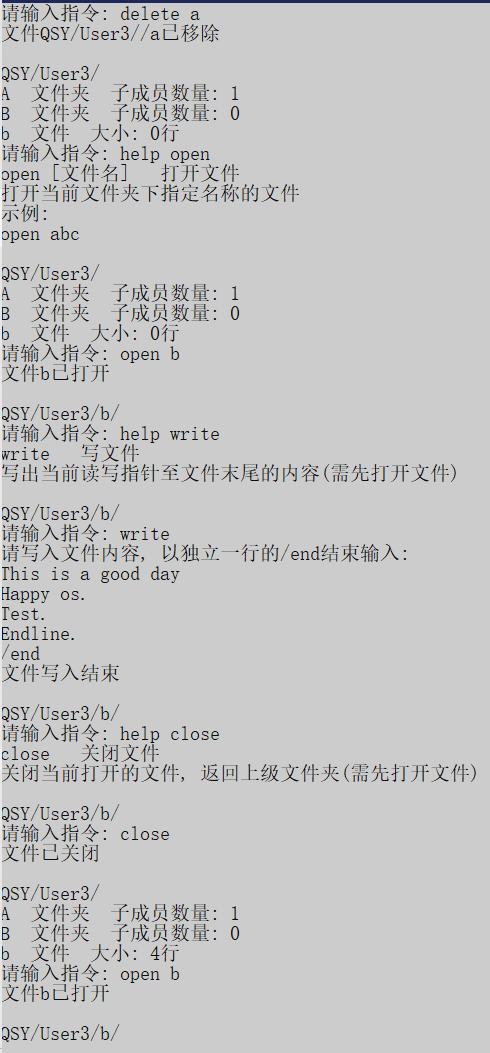


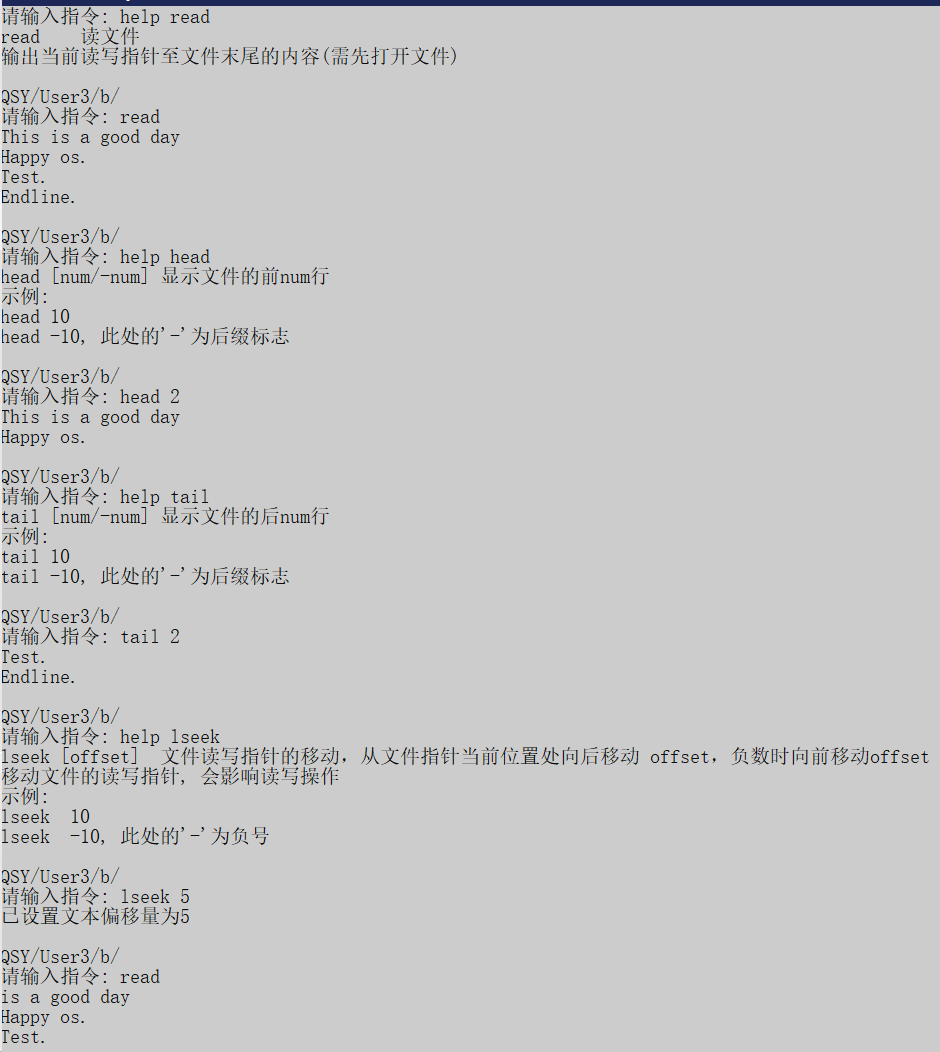
左进程写入完成后, 右进程执行read指令, 看见了左进程写入的内容, 并再次执行write指令, 覆写了左进程写入的内容, 左进程此时再执行read指令, 看见的是右进程写入的内容.

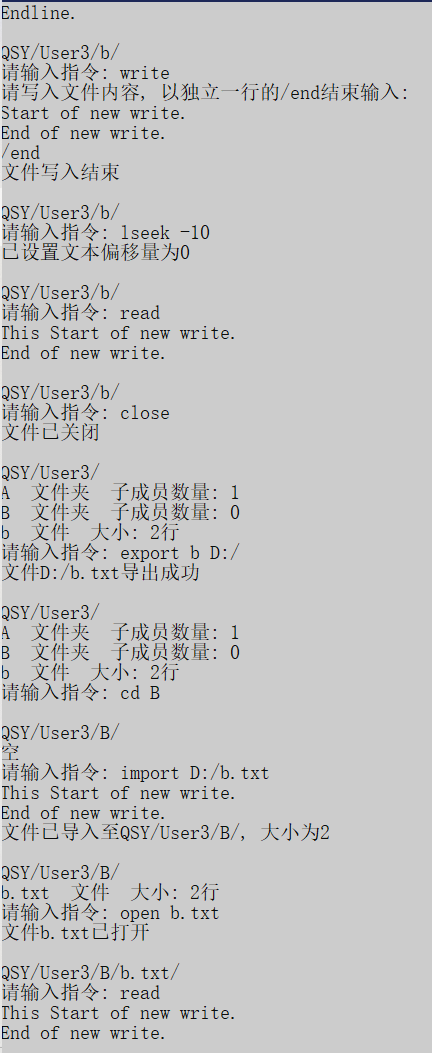
1. **实验结果截图**

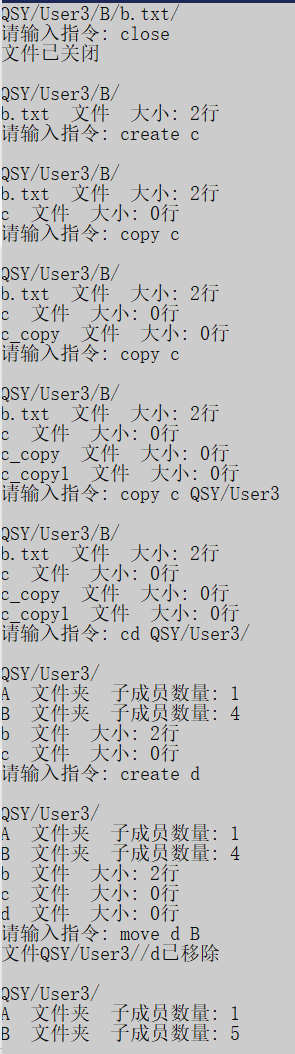


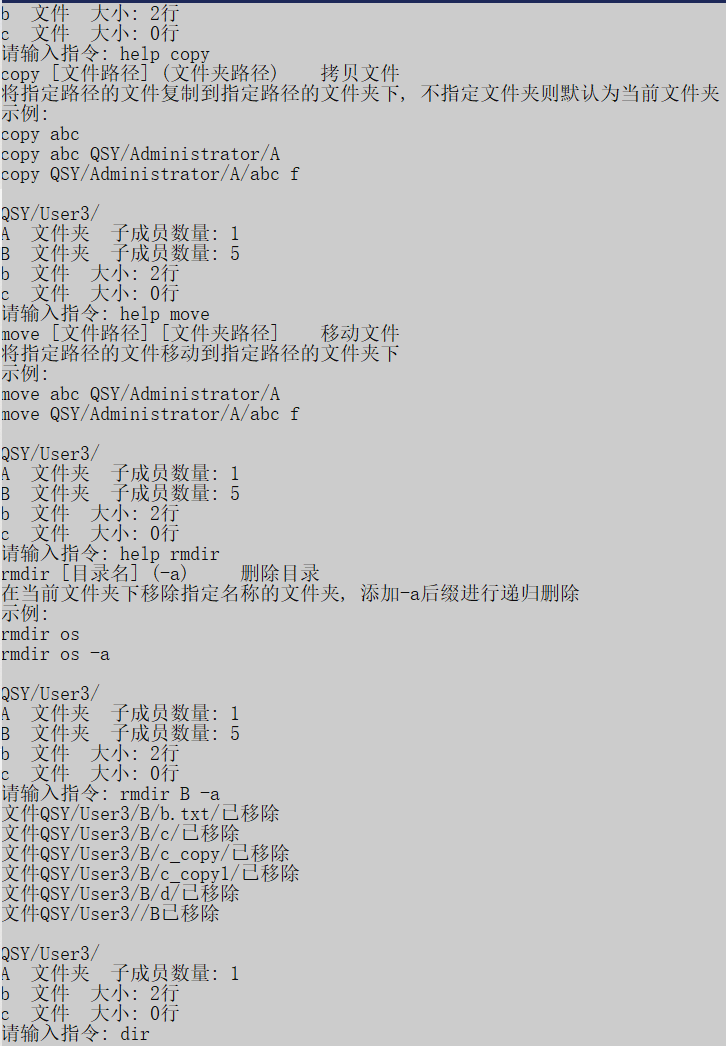


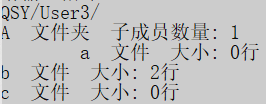












1. **实验分析及感受**
2. 课程设计中尚可改进的点:
3. 二进制读写: 由于程序的读写逻辑还是构建在文本文件上的, 即使进行了二进制封装, 程序运行时仍然会在本地创建一个额外的文本文件, 也依赖于这个文本文件进行各类进程同步与通信操作.为了实现完全的二进制读写, 需要将原本以流为基础读写方式彻底重写.
4. 存储结构: 存储时仍是顺序存储, 多数情况下修改文件需要覆写整个文件. 可以进行存储分区, 规定各部分存储的大小上限, 并引入存储管理算法, 建立相关的分区存储记录表项, 完善存储机制.
5. 进程间通信机制: 目前仅基于共享存储实现了PV变量的共享, 可以考虑引入更复杂的进程间通信机制.
6. 导入文件时的换行符问题: 在编辑外部文件时, 可能会因为编码格式或是其他原因出现换行符无法识别的情况, 影响导入文件的读取效果.
7. 课程设计中已解决的问题及解决办法:
8. 多进程的读写控制问题:

在文件中存储公用的PV变量, 使用PV操作控制文件读写

1. 多进程的数据同步问题:

在每个指令周期的前后存写数据, 确保内存数据与磁盘数据一致

1. cd指令访问不可访问节点的问题:

不做限制的cd指令可以绕过登录系统, 也可直接访问文件, 因此要添加足够的判断条件进行限制. 但同时也有考虑到, 一部分内部函数需要无限制的cd指令进行节点访问, 因此设置了cd指令的不同权限级, 在高权限级下为这些函数提供服务.

1. 重新读写文件导致的存储于内存的文件读写指针位置被刷新问题:

初步引入数据刷新实现进程同步后, 设置的文件指针在每次刷新时会清零, 导致了lseek指令失效, 因此将文件指针位置也写入文件中.

1. Copy指令导致的文件重名问题:

由于同一文件夹下的文件不能重名, copy指令需要采取重名预防机制, 具体的预发机制在相关指令中已介绍.

1. Dir类中直接使用向量存储子节点本身导致的访问问题:

最早的Dir类中存储子节点的向量类直接存储子节点本身, 这导致了Dir类在进行赋值和移动时指针指向为空的问题(原本存储的子节点在程序运行中已被释放并转移至其他位置), 修改为使用向量类存储指向子节点的指针解决了这一问题.

1. 课程设计中的收获与感悟:
2. 复杂项目的顶层设计非常关键:

代码量越多的项目后期的修改与维护就越困难, 因此越需要高瞻远瞩的顶层设计, 走一步看一步的方式很容易引起内部的功能冲突.

1. 底层功能的修改需要更加慎重, 进行更多的调试:

本次课设将二进制读写的实现放在了最后, 这使得后续进行修改时积重难返, 并且修改也使得部分可用的功能在出现了一定的问题, 导致了返工和部分功能上的妥协.

1. 不同类别的功能间应尽可能独立:

为了避免修改a影响b的情况出现, 在构建不同的功能时, 要尽可能独立, 避免相互依赖导致修改不符合预期.

1. 代码规范的重要性:

对功能的实现要有一定的统一规范, 不同的类负责的部分要有区分,.

1. 对操作系统更深刻的认识:

在本次课设中, 对操作系统的功能有了更深的认识, 也对进程间通信, 存储管理等操作系统功能的必要性有了更深刻的体会.