《操作系统原理及应用》课程设计报告

虚拟文件系统模拟

|  |  |  |
| --- | --- | --- |
| 课程目标1 | 课程目标2 | 课程目标3 |
|  |  |  |

学院（系）： 计算机科学与工程学院

班 级： 计科三班 学号 11703990535

学生姓名： 陈松

同组人员： 无

目录

[一、课程设计的内容及要求 3](#_Toc28346407)

[1.1 课题内容及要求 3](#_Toc28346408)

[1.2 实现的功能 3](#_Toc28346409)

[二、设计方案分析 3](#_Toc28346410)

[2.1 磁盘管理模块 3](#_Toc28346411)

[2.2 文件管理模块 4](#_Toc28346412)

[2.3 权限控制模块 4](#_Toc28346413)

[2.4 用户登录模块 4](#_Toc28346414)

[三、选择和使用现代工具 5](#_Toc28346415)

[四、课题实现 5](#_Toc28346416)

[4.1 用户登录模块 5](#_Toc28346417)

[4.1.1 用户登录 5](#_Toc28346418)

[4.2 权限控制模块 5](#_Toc28346419)

[4.2.1 申请操作权限 5](#_Toc28346420)

[4.3 磁盘管理模块 6](#_Toc28346421)

[4.3.1 磁盘块分配回收 6](#_Toc28346422)

[4.4 文件管理模块 7](#_Toc28346423)

[4.4.1 初始化引导目录 7](#_Toc28346424)

[4.4.2 创建目录文件 8](#_Toc28346425)

[4.4.3 创建数据文件 9](#_Toc28346426)

[4.4.4 文件删除 9](#_Toc28346427)

[4.4.5 文件打开 11](#_Toc28346428)

[4.4.6 文件修改 11](#_Toc28346429)

[4.4.7 文件系统格式化 12](#_Toc28346430)

[五、课题实现结果 13](#_Toc28346431)

[4.1 用户登录模块 13](#_Toc28346432)

[4.1.1 用户登录 13](#_Toc28346433)

[4.1.2 用户切换 15](#_Toc28346434)

[4.1.3 用户注销 15](#_Toc28346435)

[4.2 文件管理模块 16](#_Toc28346436)

[4.2.1 创建目录文件 16](#_Toc28346437)

[4.2.2 创建数据文件 17](#_Toc28346438)

[4.2.3 文件打开 18](#_Toc28346439)

[4.2.4 文件修改 18](#_Toc28346440)

[4.2.5 文件删除 19](#_Toc28346441)

[4.3 磁盘管理模块 20](#_Toc28346442)

[4.3.1 磁盘分配 20](#_Toc28346443)

[4.3.2 磁盘空间回收 21](#_Toc28346444)

[4.4 权限管理模块 22](#_Toc28346445)

[4.4.1 文件创建及删除权限 22](#_Toc28346446)

[六、结果分析 23](#_Toc28346447)

[6.1 优点 23](#_Toc28346448)

[6.2 缺点及不足 23](#_Toc28346449)

[七、参考文献 23](#_Toc28346450)

# 一、课程设计的内容及要求

## 1.1 课题内容及要求

设计实现一个虚拟文件系统，所设计的文件系统要有文件存储空间的管理和目录管理，要具备文件系统格式化、用户登录、用户注销、显示目录内容、文件创建、文件删除、目录创建和目录删除等功能，创建的文件不要求格式和内容，但要有相应的权限控制。要有目录结构的设计和磁盘空间的分配与回收等文件系统核心功能。

提示：可以用一个文件或在内存中开辟一个虚拟磁盘空间来模拟文件系统的磁盘空间。

## 1.2 实现的功能

本次虚拟系统模拟课设，主要完成了四个模块。

磁盘管理模块，实现用户进行文件修改、删除、创建时，文件系统自动分配，回收磁盘空间的功能，即实现文件系统底层，磁盘分配回收对用户的透明。

文件管理模块，面向用户编程，即提供用户操纵磁盘空间的接口，文件系统格式化，显示目录内容、文件创建、文件删除、目录创建和目录删除等功能。

权限控制模块，提供了两种权限控制，即文件创建者的对文件的读写权限和其他用户对文件的读写权限，如drwr~ 表示目录文件的创建者对目录有创建下级目录、文件的权限及对当前目录的删除修改权限，而其他用户，没有对当前目录修改的权限。

用户登录模块，提供了用户登录、注销文件系统的功能。

# 二、设计方案分析

## 2.1 磁盘管理模块

磁盘盘块化,将磁盘空间逻辑上划分为大小为4kB的各个盘块，默认第一盘块用来磁盘块是否被使用的信息，方便文件系统启动时，获得空闲盘块的索引。

采用空闲链表法实现对磁盘块的分配与回收，即启动文件系统时，读取第一盘块存储的磁盘信息，将所有空闲盘块拉成一条空闲盘块链。当用户因创建文件而请求分配存储空间时，系统从链首开始，依次摘下适当数目的空闲块分配给用户；当用户因删除文件而释放存储空间时，系统将回收的盘块依次挂在空闲盘块的末尾。

合理性及可行性：空闲链表法使对磁盘块的分配回收的过程非常简单，除了文件系统启动时需要读取磁盘中的空闲表信息外，其他时候仅对空闲链表操作即可，时间复杂度几乎降至o(1)，大大提高了磁盘的分配回收速度，而且容易实现，空闲链表中只需存空闲块的编号即可，分配时可以通过块号直接定位到对应的磁盘块进行相应的操作。

结论:方案可行，且易实现。

## 2.2 文件管理模块

本系统仅定义了两种文件结构，即目录文件和数据文件。

文件的基础信息都存放在索引节点中，包括文件名（12B），文件父目录的索引节点编号（4B），文件大小（4B），文件所占块数(4B)，文件类型(4B)，文件拥有者(12B),文件类型（11B）,文件权限(1B),文件索引项(8个索引项,每个索引项4B)存放磁盘块编号。每个索引节点占80B的磁盘空间，即10个4kB磁盘块可存放512个文件索引节点，考虑到文件操作时对索引节点的频繁访问，系统初始化时，就占用了第2到第11共10个磁盘块的空间用于顺序存放索引节点。

目录文件：索引节点的类型为目录,即对应的地址项为8个直接地址，直接指向磁盘块，对应的数据结构:文件名(12B),对应索引节点编号(4B),即每个目录项占16B,每个4KB磁盘块可存放256个目录项，即每个目录下最高可创建8\*256个目录项。注意:磁盘块的分配采用了动态分配空间的方式，目录创建时，仅建立对应的索引(索引节点以及父目录目录项)，不为其分配空间，当目录下创建下级目录文件时，才开始为其分配磁盘块。

数据文件: 索引节点的类型为目录,对应的地址项为5个直接地址,2个一级间址,1个二级间址，磁盘块编号占4B,即每个数据文件最多存放(5+4kB/4B\*2+4kB/4B\*4kB/4B)\*4KB = 20KB + 8MB + 4GB 的数据，同上，动态分配空间，仅当数据文件写入数据时，才分配空间。

合理性及可行性：采用了动态分配空间的方式，减少了对磁盘空间的浪费，检索目录下的文件时，只用到了文件名，提高了目录检索的速度。数据文件的存储采用多级索引组织方式，使得对较大数据文件的读写变得方便，且易实现。

结论:方案可行，且易实现。

## 2.3 权限控制模块

权限控制模块，对每一次文件操作加锁，即先判断文件是否有执行权限。

每个文件的索引项都有对应的权限信息，本系统将权限分为两组，文件创建者的读写权限和其他用户读写权限。对目录，没有对应的读权限，则查看文件系统时，不显示该目录，没有对应的写权限，则无法对目录进行修改，对数据文件同理。Ps:root用户和文件创建者拥有强制读权限(尽管没有对应读权限)，强制文件权限修改的权限(尽管没有对应写权限)。

合理性及可行性:权限设置详细，准确。提高了文件系统的安全性和可靠性。

结论:方案可行，且易实现。

## 2.4 用户登录模块

用户登录：登录系统后才能对文件系统进行操作，结合权限控制模块，给定相应权限。

用户切换: 更换当前文件系统的操作者,且定位到其工作目录

用户注销: 文件系统关闭。

合理性及可行性: 功能简单，易实现。

结论:方案可行，且易实现。

# 三、选择和使用现代工具

平台: window10

语言: python3.6

UI实现工具:pyqt5

# 四、课题实现

## 4.1 用户登录模块

### 4.1.1 用户登录

# 登录控制

def login(self, name, password):

userDict = {'root': 'root', 'Curious': 'root', 'Uncle': 'root'}

if name in userDict:

if userDict[name] == password:

self.user = name

return True

return False

-------------------------------------------------------------------------------

**功能:检查用户名是否存在，若存在检查密码是否正确，正确则返回True登录成功，错误，则返回False,登录失败。**

## 4.2 权限控制模块

### 4.2.1 申请操作权限

# 用户权限控制

def getPower(self, owner, power, op):

if op == 'open':

# root用户,和文件夹拥有者拥有所有权限

if self.user == 'root' or self.user == owner:

return True

# 可读，则可打开

if len(power) >= 5 and power[-2] == 'r':

return True

if op in ['mkdir', 'rm', 'touch']:

# root用户,和文件夹拥有者拥有所有权限

if self.user == 'root' or self.user == owner:

return True

# 可写，则可创建

if len(power) >= 5 and power[-1] == 'w':

return True

if op == 'change':

if self.user == owner and power[2] == 'w':

return True

if self.user != owner and power[-1] == 'w':

return True

# 格式化磁盘权限

if op == 'clear' and self.user == 'root':

return True

# 权限不足

print("{}没有{:^8s}权限{}".format('\*' \* 20, op, '\*' \* 20))

return False

---------------------------------------------------------------------

输入：文件拥有者，权限，操作，（注意:当前用户为成员变量）

输出: True or False 申请权限是否陈宫

功能: 规则判断当前用户是否有当前操作的权限

## 4.3 磁盘管理模块

### 4.3.1 磁盘块分配回收

def alloc(self, index\_node: IndexNode, baseINFO):

# 获得应分有的磁盘块

block\_num = index\_node.BLOCKNUM

# 获得已占用的磁盘块

occupy\_indexs = list(filter(lambda x: x != -1, index\_node.indexs))

occupy\_num = len(occupy\_indexs)

if block\_num > occupy\_num:

# 还需要几块

occupy\_indexs.extend(self.findFreeBlock(block\_num - occupy\_num))

elif block\_num < occupy\_num:

# 释放超出的内存块

t = block\_num - occupy\_num

self.freeBlock(occupy\_indexs[t:])

occupy\_indexs = occupy\_indexs[:t]

# 重写

for i in occupy\_indexs:

self.block\_list[i] = baseINFO

# 重新编辑索引项

index\_node.indexs = np.full([8, ], -1)

for i, v in enumerate(occupy\_indexs):

index\_node.indexs[i] = v

-----------------------------------------------------------------------

输入:操作文件的索引节点

功能: 根据当前文件大小，计算所需要的内存块数，不够在已有的内存块的基础上再分配所需的内存块。超出则将多余的磁盘块释放，即将磁盘块编号放入空闲链表中。

## 4.4 文件管理模块

### 4.4.1 初始化引导目录

def initCatlog(self, length=10):

#######################

# 1、初始化索引栈

#######################

# 引导目录索引

root\_node = IndexNode(0, '引导目录', 'catalog', 'drwr\_', 'root', block\_num=1) # 抽象化索引信息块

# 定义索引节点在磁盘中的位置

self.IS = IndexStack(list([root\_node]))

# 分配固定的索引空间

for i in range(-length, 0, 1):

print(i)

self.BLOCKLIST[i] = self.IS

self.free\_blocks.pop()

# 分配固定的索引表

self.BLOCKLIST[0] = self.free\_blocks

self.free\_blocks.remove(0)

part\_catalog = Catalog()

# 分配根目录项存放空间，即建立索引项 索引节点编号 ->磁盘编号

self.DM.alloc(root\_node, part\_catalog)

#######################

# 2、添加目录

#######################

# 目录1:root

self.mkdir('root', 'drwr\_', 'root', 0)

# 目录2:home

self.mkdir('home', 'drwr\_', 'root', 0)

# 目录3:Curious

self.mkdir('Curious', 'drwr\_', 'Curious', 2)

# 目录4:Uncle

self.mkdir('Uncle', 'drwr\_', 'Uncle', 2)

-------------------------------------------------------------------------------

功能 : 文件系统初次启动、文件系统格式化时启动，定义引导目录，及目录下面的用户工作目录，文件系统中所有文件的最终节点，及其父目录的索引节点编号为0(引导目录的索引节点编号),方便系统向上遍历出当前文件的绝对路径。

### 4.4.2 创建目录文件

def mkdir(self, name, mode, owner, pn):

"""

创建文件夹

"""

# 1、建立索引节点

index\_node = IndexNode(pn, name, 'catalog', mode, owner, block\_num=1)

cn = self.getID() # 获得索引节点编号

# 2、更新父目录

if not self.updataDir(pn, name, cn): # 目录项:文件名 - >索引节点编号

# 重名

return False

# 3、更新索引栈 ：将索引节点加入索引栈中

if cn < len(self.IS.nodes):

self.IS.nodes[cn] = index\_node

else:

# 将索引节点加入索引栈中

self.IS.nodes.append(index\_node)

return True

-------------------------------------------------------------------------------

输入: 目录名,权限，拥有者，父目录索引节点编号

输出: True or False 目录是否创建成功

功能: **建立索引**。通过接受的信息创建索引节点，然后打开父目录的目录项，顺序检索，若出现命名重复，则目录创建失败，若无重复，则获得索引节点栈对应的ID,在父目录下建立对应的目录项，在索引栈对应位置添加目录的索引节点。

### 4.4.3 创建数据文件

def touch(self, name, mode, owner, pn):

"""

创建数据文件

创建时，不需要分配空间,只需要添加目录项即可

:param pn 当前目录索引ID

"""

# 添加索引项

indexNode = IndexNode(pn, name, 'file', mode, owner)

cn = self.getID() # 获得索引节点编号

# 更新当前目录

if not self.updataDir(pn, name, cn): # 文件名 - >索引节点编号

# 重名

return False

if cn < len(self.IS.nodes):

self.IS.nodes[cn] = indexNode

else:

# 将索引节点加入索引栈中

self.IS.nodes.append(indexNode)

return True

-------------------------------------------------------------------------------

输入: 目录名,权限，拥有者，父目录索引节点编号

输出: True or False 数据文件是否创建成功

功能: **建立索引**，步骤同上述目录创建。

### 4.4.4 文件删除

def rm(self, pn, cn):

"""

:param pn: 父目录编号

:param cn: 操作文件索引编号

"""

# 索引节点编号->磁盘块编号->目录项文件信息

parentDict = self.BLOCKLIST[self.IS.nodes[pn].indexs[0]].files

# 获得操作文件的索引节点块(FCB)

root\_node = self.IS.nodes[cn]

# 从父目录的目录文件信息中移除目录项

parentDict.pop(root\_node.NAME)

# 向上递归修改文件大小

pn = root\_node.pn

# 向上修改所有父目录的大小

while True:

# 父目录索引

indexNode = self.IS.nodes[pn]

indexNode: IndexNode

# 修改目录大小

indexNode.SIZE = indexNode.SIZE - root\_node.SIZE

# 到达引导目录，结束

if pn == 0:

break

# 改为上一层ID

pn = indexNode.pn

# 获得应该删除的所有索引

del\_indexs = list()

if root\_node.TYPE == 'catalog':

# 递归的删除索引节点

# 抽象化的目录项信息,获得其中一个，即可抽象获得不同磁盘块的所有目录项

v = root\_node.indexs[0]

# v:磁盘块编号

# 从磁盘块中取出目录项信息

catalog = self.BLOCKLIST[v]

catalog: Catalog

# 获得所有目录项

t = [(name, ID) for name, ID in catalog.files.items()]

for item in t:

name, ID = item # ID:索引编号

# 获得目录项对应的索引节点

index\_node = self.IS.nodes[ID]

index\_node: IndexNode

if index\_node.TYPE == 'catalog':

self.rm(cn, ID)

# 添加该文件占用的所用磁盘块编号

del\_indexs.extend(filter(lambda v: v != -1, root\_node.indexs))

# 链接删除，即将空闲磁盘块加入空闲栈

self.DM.freeBlock(del\_indexs)

# 删除当前索引节点

self.IS.nodes[cn] = -1

-------------------------------------------------------------------------------

输入: 删除文件的ID,即索引节点编号。

功能: 如果为目录文件，需要向下遍历删除对应的所有下级文件，数据文件者直接删除。然后向上遍历，修改对应的所有上级目录的大小（即减去文件对应的大小），最后移除父目录对应的目录项即可。

### 4.4.5 文件打开

def open(self, ID):

"""

针对文件

"""

# 通过ID找到索引节点

indexNode = self.IS.nodes[ID]

indexNode: IndexNode

# 以json形式返回文件信息

inform = {'name': indexNode.NAME, 'TYPE': indexNode.TYPE, 'power': indexNode.MODE,

'blocknum': indexNode.BLOCKNUM, 'SIZE': indexNode.SIZE,

'indexs': [int(i) for i in indexNode.indexs]}

inform = json.dumps(inform)

return inform

-------------------------------------------------------------------------------

输入: 文件ID,即唯一对应的索引节点编号

输出：文件的所有信息

功能: 通过文件ID,找到对应的索引节点，获得文件的基础信息，并通过基础信息中的地址项，获得其数据部分对应的磁盘块，从而获得文件所有的信息。

### 4.4.6 文件修改

def vim(self, ID, SIZE):

"""

写入数据：

按索引，修改所有父目录大小

注意:磁盘分配问题没有解决

:param ID:

:return:

"""

# 获得文件属性

indexNode = self.IS.nodes[ID]

indexNode: IndexNode

# 修改文件大小

indexNode.SIZE = indexNode.SIZE + SIZE

# 没有分配空间

if indexNode.BLOCKNUM == 0:

# 创建文件数据块

part\_file = File()

part\_file.occupy = SIZE

else:

part\_file = self.BLOCKLIST[indexNode.indexs[0]]

# 修改占用磁盘块数

import math

indexNode.BLOCKNUM = math.ceil(indexNode.SIZE / self.BLOCK\_SIZE)

# 索引编号 - 磁盘块

self.DM.alloc(indexNode, part\_file)

pn = indexNode.pn

# 向上修改所有父目录的大小

while True:

# 父目录索引

indexNode = self.IS.nodes[pn]

indexNode: IndexNode

# 修改目录大小

indexNode.SIZE = indexNode.SIZE + SIZE

# 到达引导目录，结束

if pn == 0:

break

# 改为上一层ID

pn = indexNode.pn

-------------------------------------------------------------------------------

输入: 文件ID, 处于内存中的文件内容content

功能: 根据文件内容，修改对应的索引节点大小，然后向上遍历修改所有上级目录的大小，然后由磁盘管理模块自动分配回收磁盘块写入文件内容。

### 4.4.7 文件系统格式化

def empty\_format(self):

if self.getPower('', '', 'clear'):

# 清空磁盘

for i, v in enumerate(self.BLOCKLIST):

if v != -1:

self.BLOCKLIST[i] = -1

# 清空索引

self.IS.nodes.clear()

# 重新建立空闲表

self.free\_blocks.clear()

self.init\_free\_block()

# 重建引导区

self.initCatlog()

-------------------------------------------------------------------------------

功能: 该操作仅root用户拥有其使用权限，步骤如上所示，先直接清空磁盘和所有索引，然后重建引导目录。

# 五、课题实现结果

## 4.1 用户登录模块

### 4.1.1 用户登录

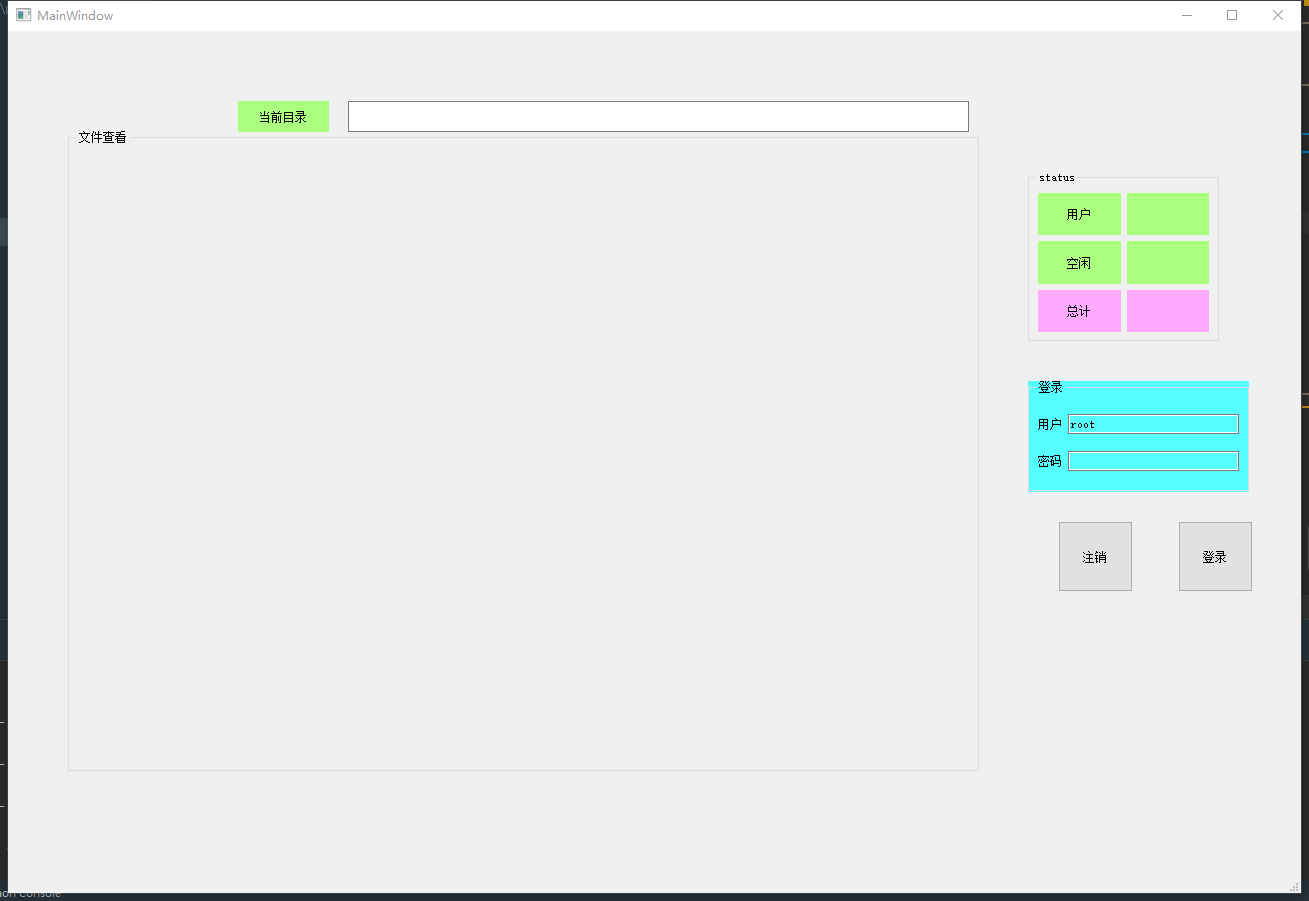


图1 用户登录前

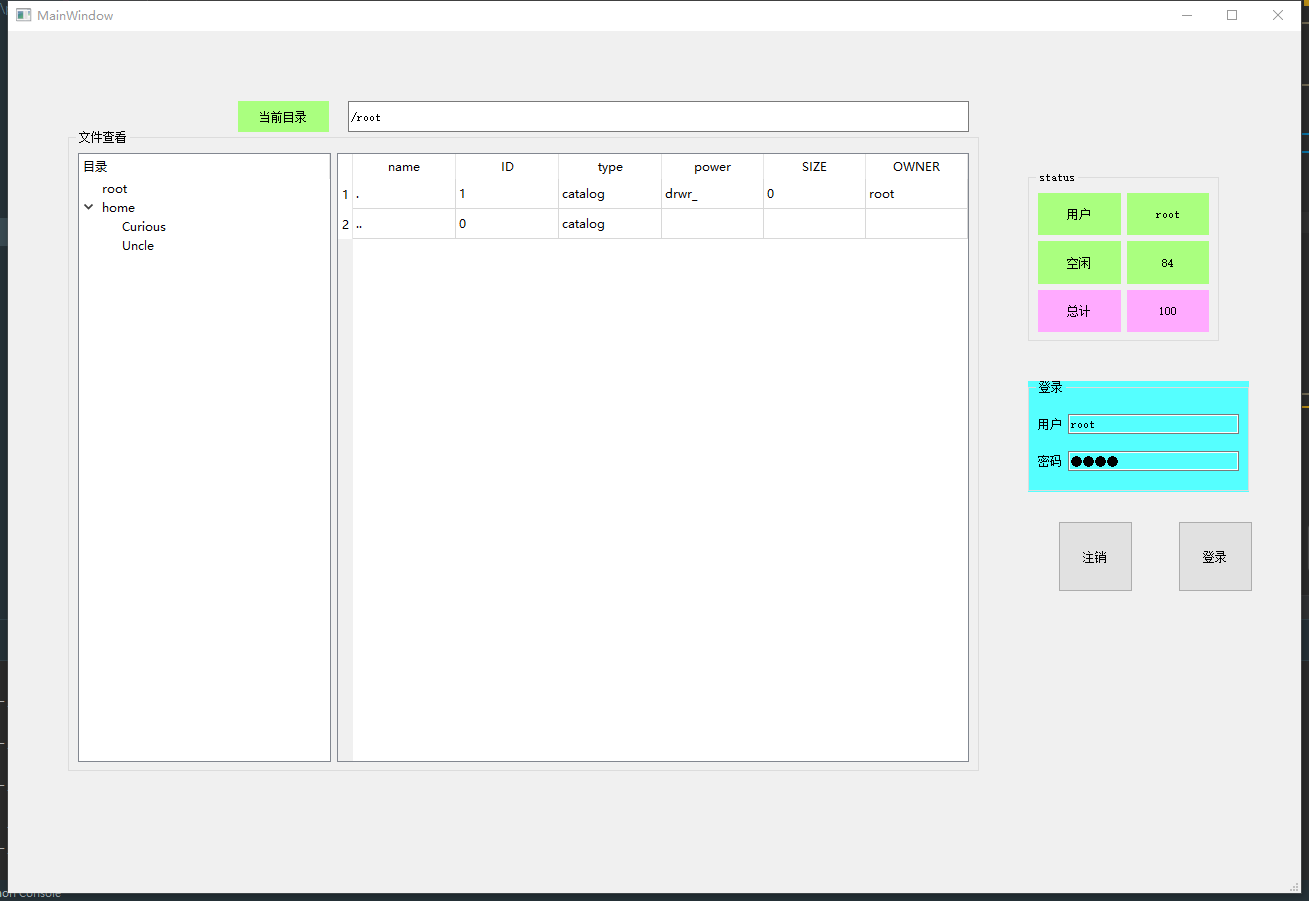


图2 登录成功后

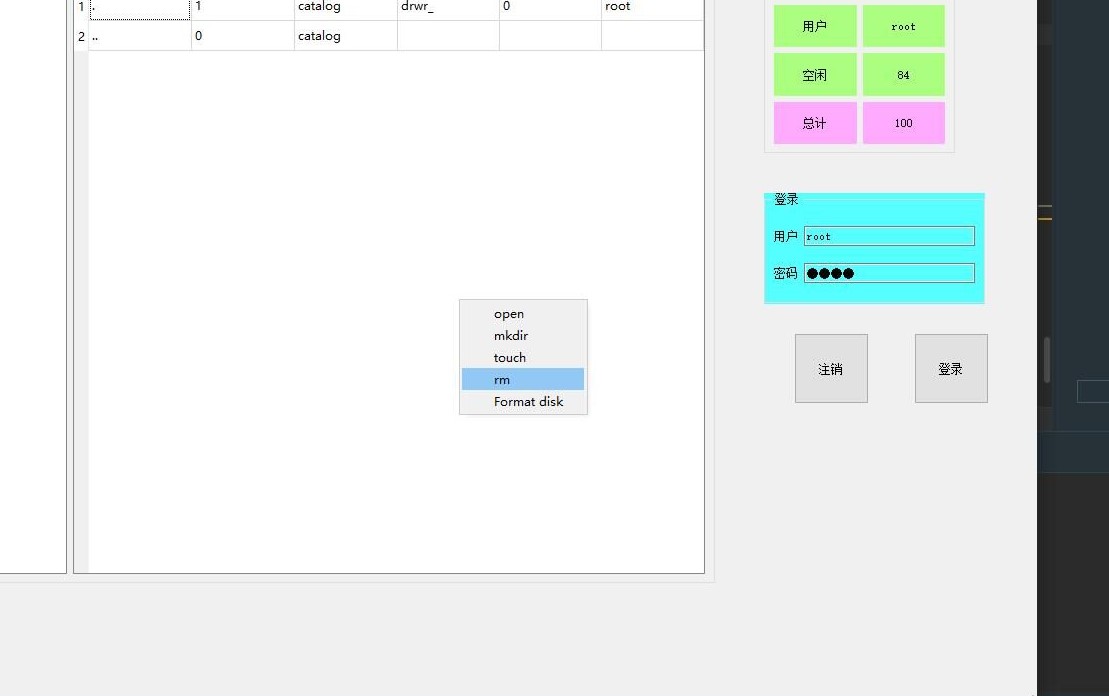


图 3 右键菜单栏

-------------------------------------------------------------------------------

输入：账户名和密码，并点击登录按钮

输出: 获得文件系统对应的执行权限，如图2 status 状态栏所示当前用户变为了root,如图3所示，右键可触发所有文件操作的菜单栏 。

### 4.1.2 用户切换

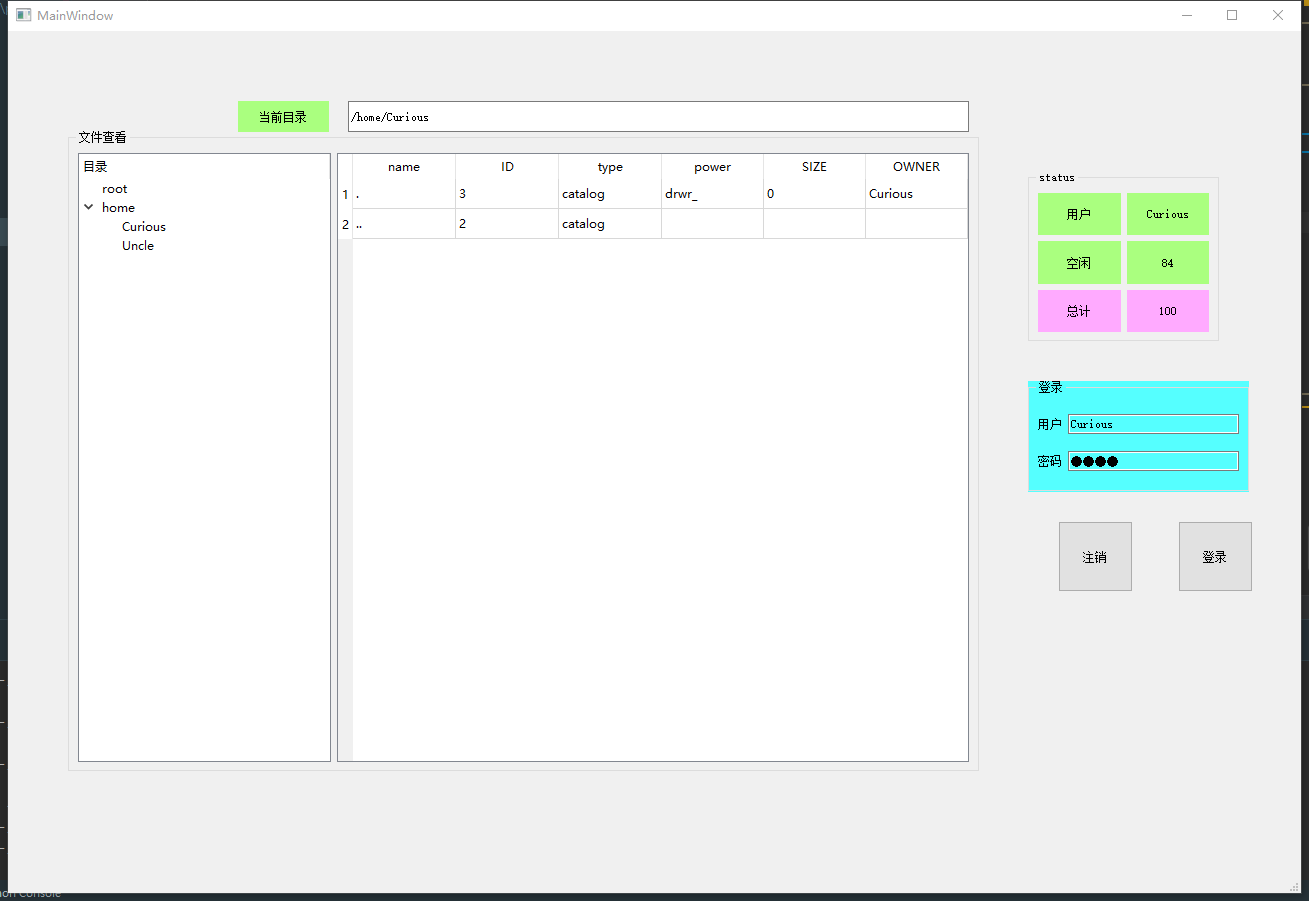


图4 切换用户

-------------------------------------------------------------------------------

输入: 其他账户名(Curious)和密码(root)，点击登录按钮

输出: 当前用户变为Curious,UI界面切换到用户的工作目录,如上图4所示，status栏中的用户变为了Curious,当前目录变为 /home/Curious.

### 4.1.3 用户注销

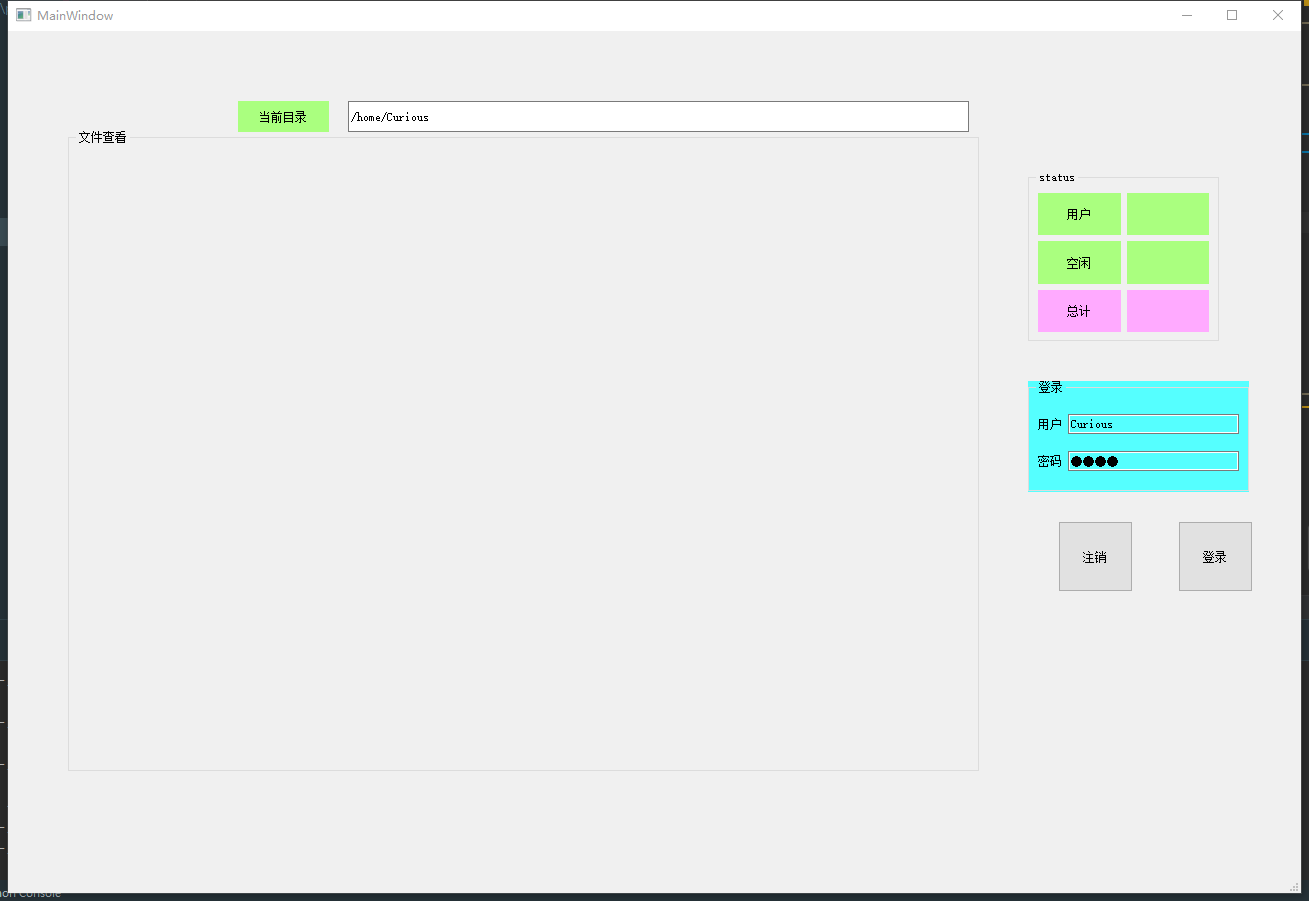


图5 用户注销

-------------------------------------------------------------------------------

输入: 点击注销按钮

输出: 文件系统不可查看，只能进行登录操作。如上图5所示，文件产看模块无内容，status状态栏为空，即无用户登录。

## 4.2 文件管理模块

### 4.2.1 创建目录文件

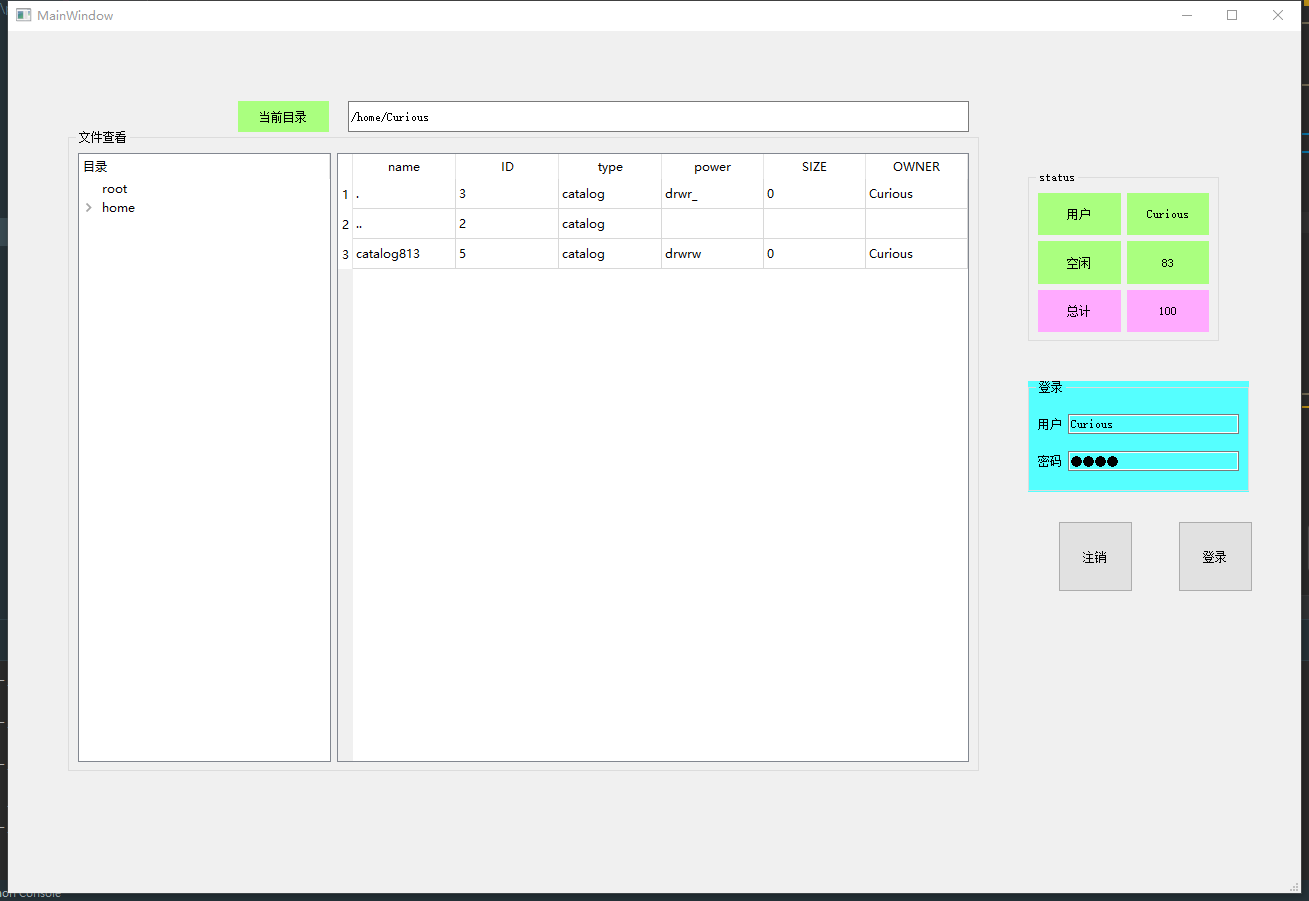


图6 目录文件创建

-------------------------------------------------------------------------------

输入: 右键打开菜单栏，选择并点击mkdir

输出: 随机生成目录名，并以此创建新的目录，如上图6所示，在/home/Curious目录下创建了一个名为calaog813的目录。

注意: 目录对应的数据目录项虽然会占用空间，但SIZE大小不计算目录的目录项占用的大小，只计算目录下的数据文件的大小

### 4.2.2 创建数据文件

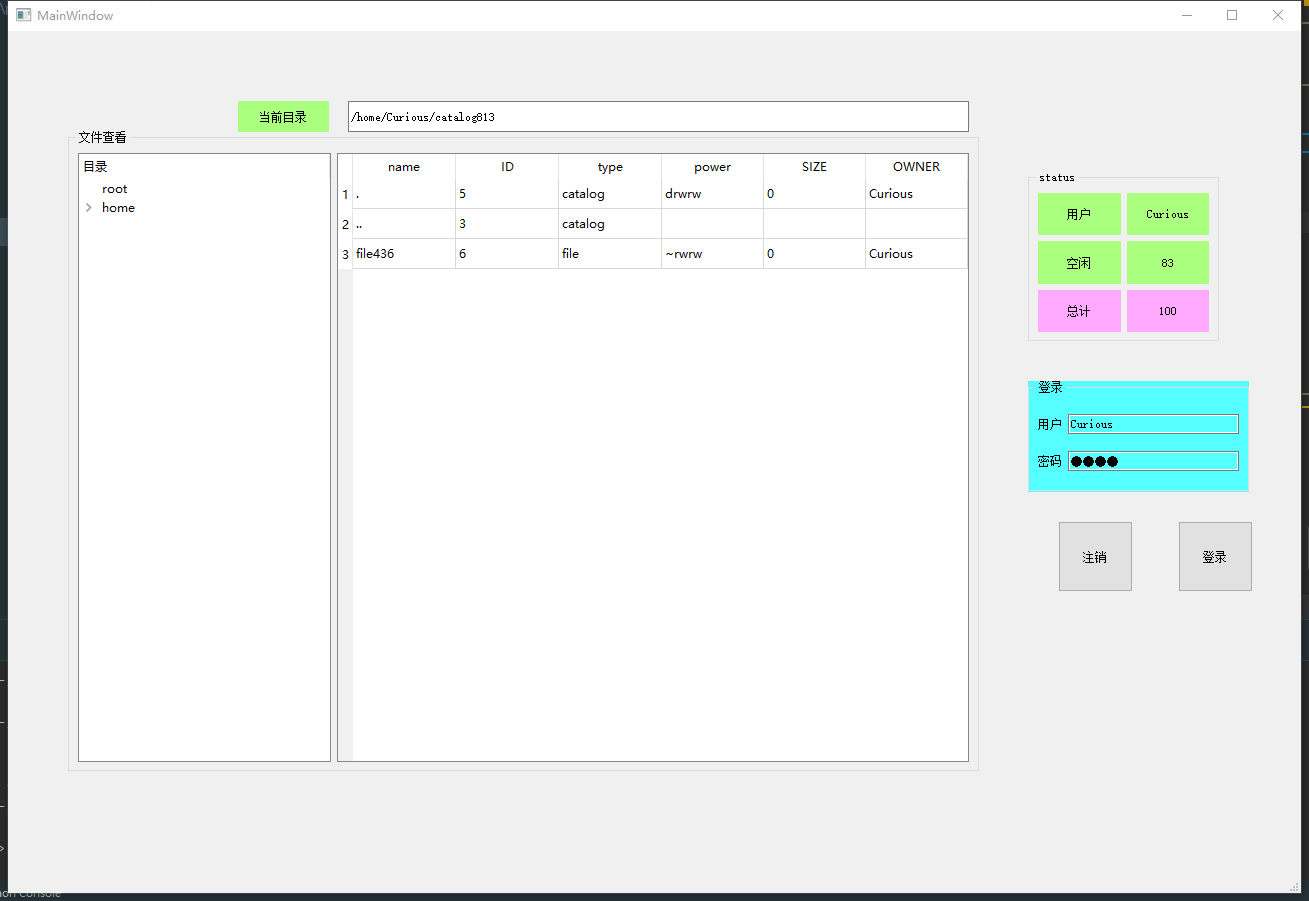


图7 数据文件创建

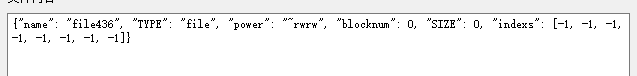


图8 file436文件的基础信息

-------------------------------------------------------------------------------

输入：切换到新创建的目录catalog813下，右键选择touch并点击。

输出: 如上图7 所示，建立了一个名为file436的数据文件

注意: 创建数据文件时，不分配空间给数据文件，touch操作仅建立了索引，没有分配空间给数据文件，如8所示，地址项‘indexs’里的值全为-1,即没有分配磁盘块。

### 4.2.3 文件打开

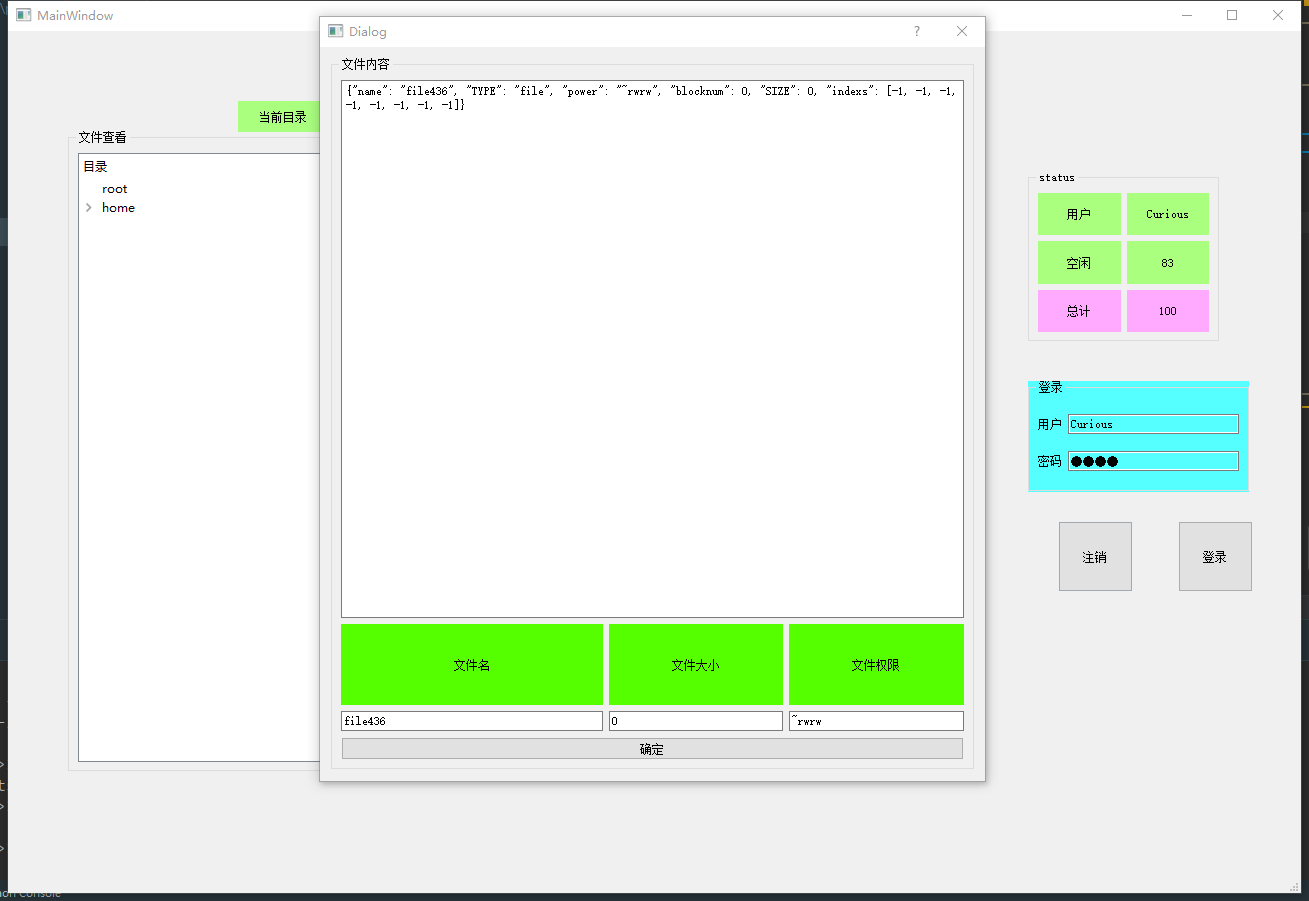


图9 文件打开

-------------------------------------------------------------------------------

输入: 选中文件file436后，右键选择并点击open。

输出: 如图9所示,可查看文件的所有信息，若有写权限，则可修改文件名和文件权限，点击确认按钮可完成修改。

### 4.2.4 文件修改

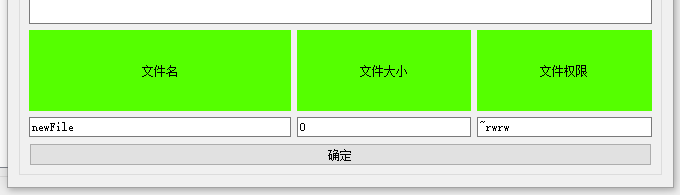


图 10 修改文件名

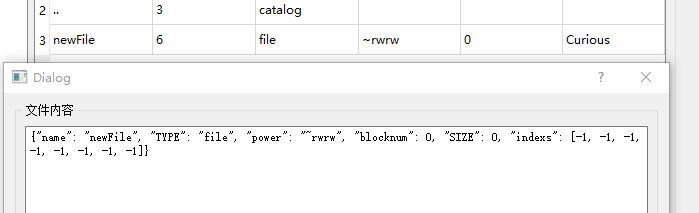


图 11 修改完成

-------------------------------------------------------------------------------

输入:open文件后，修改文件名下的输入框的值为newFile,点击确认。

输出:文件名修改为newFile，如图11所示文件名修改成功

### 4.2.5 文件删除

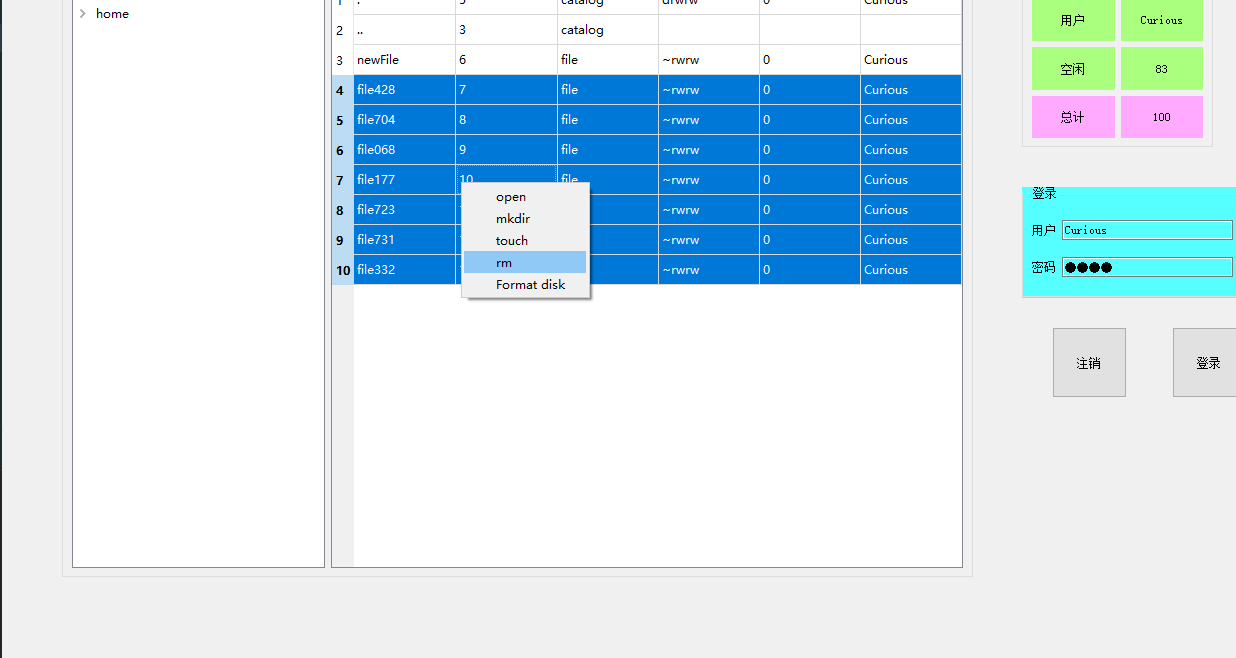


图 12 选中并右键删除

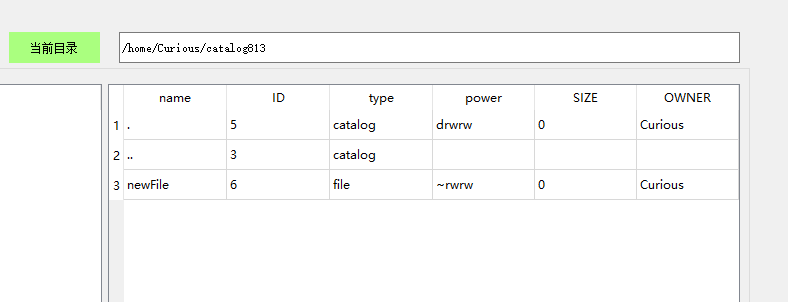


图 13 删除后

-------------------------------------------------------------------------------

输入: 选中要删除的文件，右键选中并点击rm

输出: 如图13所示，文件修改完成。

## 4.3 磁盘管理模块

### 4.3.1 磁盘分配

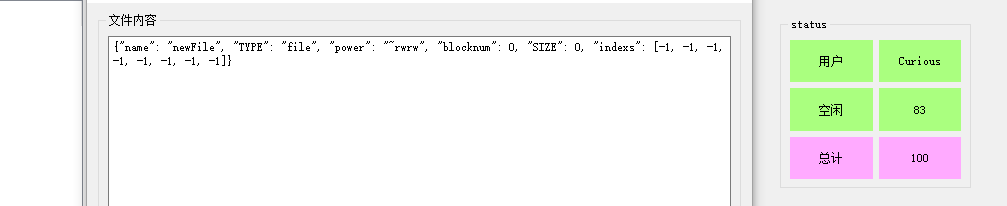


图14 未修改前

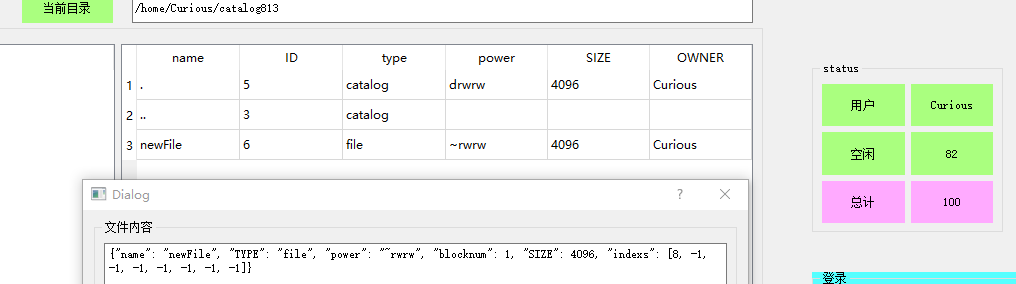


图15 修改大小为4096

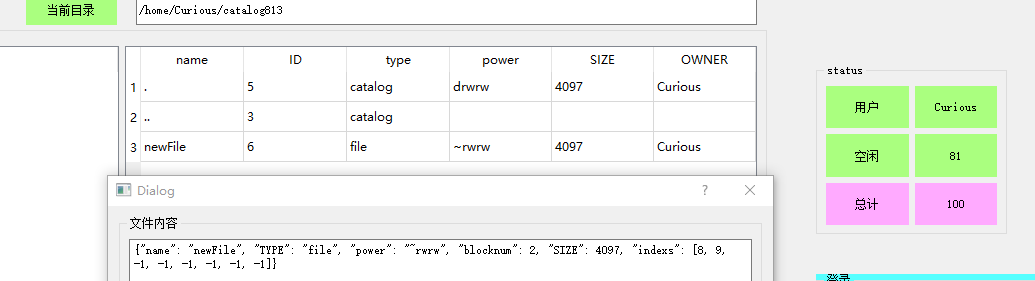


图 16 修改大小为4097

---------------------------------------------------------------------------

输入: 打开文件newFile，进行两次大小修改：4096，4097

输出: 当文件大小为4096(4K)时，占用了一个磁盘块,编号为8,如图14，15所示，状态栏里的空闲块数减一了（83->82;当文件大小为4097时，占用了两个磁盘块,编号为8，9,如图14，16所示，状态栏里的空闲块数减二了（83->81），父目录大小变为了4087.

注意：文件大小以字节为单位，而一个磁盘块4kB大小，所以4097B(4KB+1B)需要占两个磁盘块。

### 4.3.2 磁盘空间回收



图 17 修改文件大小为0

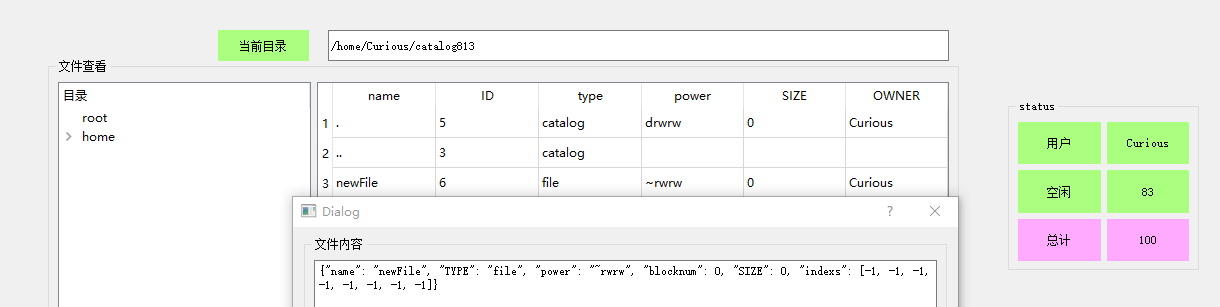


图 18 确认修改后

---------------------------------------------------------------------------

输入: 打开文件newFile，将文件大小修改为0

输出: 当文件大小为0时，文件数据不占用磁盘空间，如图16，18所示，空闲磁盘块数加2（81->83）,地址项全变为-1,即表示没有占用磁盘空间。

## 4.4 权限管理模块

### 4.4.1 文件创建及删除权限

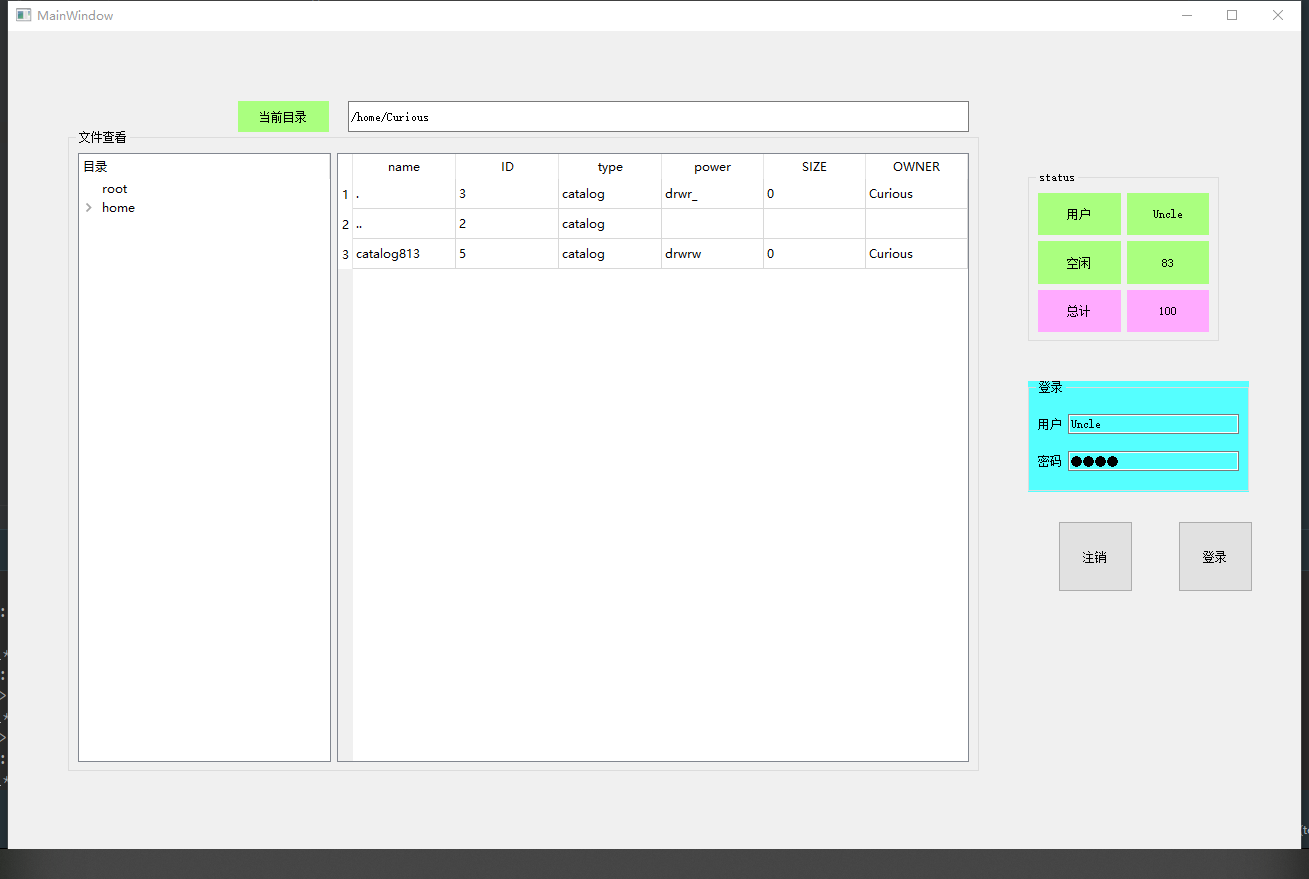


图19 权限控制

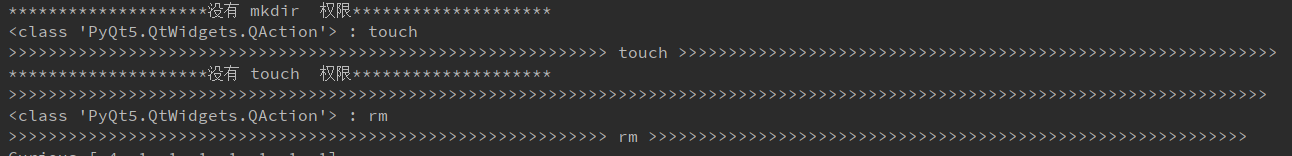


图20 权限申请失败

---------------------------------------------------------------------------

输入: Uncle用户，在Curious的工作目录下,进行文件创建，删除等操作

输出: 如19，20所示，当目录权限设置为仅拥有则可写时(drwr\_),其他用户便没有在该目录下创建删除文件的权限

# 六、结果分析

本课题设计实现了一个虚拟文件系统，功能完善，设计了文件系统格式化、用户登录、用户注销、显示目录内容、文件创建、文件删除、目录创建和目录删除等功能，并对文件有相应的权限控制。

## 6.1 优点

（1）目录项仅含目录下的文件名和其对应的索引节点编号，即进行目录检索时仅检索文件名，不包含其他文件信息，加快了顺序检索的速度

（2）目录为树形结构目录，允许不同目录下文件名重复，即创建文件时，仅需检查当前目录中是否有重复的文件名即可。

（3）对文件数据的分配采用了动态分配空间的方式，即文件有数据时，才为其分配空间，无数据时，不分配空间，节约了磁盘空间。

（4）数据文件地址项为5个直接地址,2个一级间址,1个二级间址，磁盘块编号占4B,即每个数据文件最多可存放(5+4kB/4B\*2+4kB/4B\*4kB/4B)\*4KB = 20KB + 8MB + 4GB 的数据,即提高了对较大数据文件的检索速度，8个地址项(4B\*8=32B)可建立对4GB大的文件的索引

（5）仿windos10的文件操作界面，UI界面直观，清晰，具有傻瓜式操作的特点，用户也容易接受。

## 6.2 缺点及不足

（1）使用了空闲链表法进行空间分配，虽然使得分配回收变得简单，但在为一个文件分配盘块时，可能要重复操纵多次，分配和回收的效率很低

（2）以盘块为单位，相应的空闲盘块链会很长

（3）本虚拟文件系统是在内存上开辟了一个虚拟磁盘空间来模拟的磁盘空间，即没有具体考虑真正磁盘空间的物理结构来优化程序。

# 七、参考文献

[1] python编写模拟文件系统<https://blog.csdn.net/sb_Ihateyou/article/details/78885342>

[2] PyQt5高级界面控件之QListWidget(三)

<https://blog.csdn.net/jia666666/article/details/81624837>

[3] python + pyqt5 QTreeWidget 实现文件资源管理

<https://blog.csdn.net/qq_41398808/article/details/102838427>

[4] 文件系统与虚拟文件系统 <https://blog.csdn.net/woainilixuhao/article/details/98881044>

[5] 【Linux】Linux的虚拟文件系统

<https://blog.csdn.net/qq_38410730/article/details/81368186>

[6] 操作系统总结之磁盘管理 <https://blog.csdn.net/liushengxi_root/article/details/80962979>