## 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。

**实验环境**

windows10

visual studio 2017（c++）

**实验过程**

设计思路

设计任务需要在任一os下，建立一个文件，模拟简单的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
4. 硬盘空间100MB

所以，可以将磁盘设计为由100个数据块组成，每个数据块由1024个盘块组成，每个盘块1024B，所以磁盘空间为100 \* 1024 \* 1024 B = 100MB。然后，要求有超级块，i结点，根目录，采用位图法管理空闲空间，还需要用户权限来保护文件的读写。上述信息需要定义合适的数据结构并存放在磁盘特定位置。为方便查找，将用户列表、数据块组、i-结点位图、i-结点表、盘块位图依次存放在磁盘的开始位置（如下图所示）。因此，实际文件的大小必然会大于100MB。

文件实际结构

|  |
| --- |
| 用户列表 |
| 数据块组 |
| i-结点位图 |
| i-结点表 |
| 盘块位图 |
| 磁盘空间（100MB）  根目录 |

**数据结构**

类的设计：

**用户类**

用户名，权限，密码

**数据块组类**

超级块

数据块信息：

数据块位图索引

数据块首地址

i-结点位图索引

i-结点表索引

空闲i-结点数

空闲盘块数

**超级块类**

空闲数据块总数

空闲i-结点总数

**目录类**

i-结点号

该目录下的子文件的i结点的数量

子文件的i结点

**i-结点**

文件名（目录名）

文件所有者索引号

类型（文件或目录）

文件大小

占用磁盘块数

起始地址

父目录i-结点号

代码

1. //用户类
2. **class** User
3. {
4. **public**:
5. string uName;  //用户名
6. string password; //密码
7. **int** uType;   //用户类型
8. };
10. //目录类
11. **class** Dir
12. {
13. **public**:
14. **int** iNode;      //i结点表索引
15. **int** iNode\_number; //子文件i结点数量
16. **int** iNodes[MAX\_SUBFILE];      //子文件i结点数组
18. };
20. //超级块
21. **class** SuperBlock
22. {
23. **public**:
24. **int** all\_freeBlocks;   //空闲盘块总数
25. **int** all\_freeiNodes;   //空闲i结点总数
27. };
29. //数据块
30. **class** BlockGroup
31. {
32. **public**:
33. SuperBlock sBlock;  //超级块
34. //数据块信息
35. **int** BlockMap;    //数据块开始位置的位图索引值
36. **int** BlockAddr;   //首地址
37. **int** iNodeMap;    //i结点位图索引
38. **int** iNodeTable;  //i结点表索引
39. **int** freeiNode;   //空闲i结点
40. **int** freeBlocks;  //空闲盘块总数
42. };
44. //i-结点
45. **class** i\_Node
46. {
47. **public**:
48. string Name;   //文件名
49. **int** uNumber;    //文件所有者索引
50. **int** fileType;    //文件类型
51. **int** size;        //文件大小
52. **int** BlockNumber; //占用盘块数
53. **int** BlockAddr;   //起始地址
54. **int** father\_iNode;//上一级目录i结点索引
55. };

使用的数据结构

1. **int** iNode\_bitmap[INODE\_NUMBER];  //i结点位图
2. i\_Node iNode\_table[INODE\_NUMBER];  //i结点表
3. **int** blocks\_bitmap[BlOCKS];       //盘块位图
4. BlockGroup blockGroups[BLOCK\_GROUP\_NUMBER];  //数据块表

7. Dir curDir;   //当前目录
8. **int** curUser = -1;  //当前用户
9. fstream f\_simdisk;    //虚拟磁盘文件
10. User users[MAX\_USER];  //用户列表

宏变量

1. #define BLOCK\_SIZE          1024  //盘块大小为1KB
2. #define BLOCK\_NUMBER        1024  //1024个盘块组成数据块
3. #define BLOCK\_GROUP\_NUMBER  100   //100个数据块组成磁盘
4. #define BlOCKS              (BLOCK\_NUMBER \* BLOCK\_GROUP\_NUMBER)  //盘块总数
6. #define INODE\_NUMBER        (BLOCK\_NUMBER \* BLOCK\_GROUP\_NUMBER)  //每个盘块都有i结点
8. #define MAX\_SUBFILE         300   //目录下最大子文件数目
10. #define DIR\_SIZE            (sizeof(Dir)/BLOCK\_SIZE + 1)  //一个目录需要的盘块数
12. //位图
13. #define USED                1
14. #define NOT\_USED            0

17. //i结点类型
18. #define DIRECTORY\_TYPE      0            //目录
19. #define FILE\_TYPE           1            //文件
21. //用户类型
22. #define ROOT\_USER           3
23. #define SYSTEM\_USER         2
24. #define USER                1

27. //文件属性
28. #define SHARED              0
29. #define READ\_ONLY           1
30. #define READ\_WRITE          2
31. #define SYSTEM              3
33. //最大用户数量
34. #define MAX\_USER            10
36. //数据区初始地址
37. #define DATA\_ADDR           (sizeof(BlockGroup) \* BLOCK\_GROUP\_NUMBER + sizeof(int) \* (INODE\_NUMBER + BlOCKS) + sizeof(i\_Node) \* INODE\_NUMBER + sizeof(User) \* MAX\_USER)

**文件结构**

文件地址管理

磁盘使用二进制文件进行保存。文件实际大小为105MB，可用数据部分为100MB。数据区的起始地址为DATA\_ADDR，为6147800B后的空间。100MB的空间可分为102400个盘块，盘块大小为1024B，盘块位图用一个长度为102400的整型数组保存。所以，可以对每个盘块进行编号0~102399（盘块位图数组的下标）， 每个盘块的首地址就可表示为DATA\_ADDR + n \* 1024，n为盘块号。

磁盘初始化

初始化数据块、i-结点表、i-结点位图、盘块位图、用户列表的信息、并设置当前目录为root，创建root用户，密码是password。

1. **int** i;
2. **for** (i = 0; i < BLOCK\_GROUP\_NUMBER; i++)  //初始化数据块
3. {
4. blockGroups[i].sBlock.all\_freeBlocks = BlOCKS - DIR\_SIZE; //空闲块总数减根目录的块数
5. blockGroups[i].sBlock.all\_freeiNodes = INODE\_NUMBER - 1;  //减根目录的i结点
6. blockGroups[i].BlockAddr = DATA\_ADDR + i \* BLOCK\_SIZE \* BLOCK\_NUMBER;
7. blockGroups[i].BlockMap = i \* BLOCK\_NUMBER;
8. blockGroups[i].freeBlocks = BLOCK\_NUMBER;
9. blockGroups[i].freeiNode = BLOCK\_NUMBER;
10. blockGroups[i].iNodeMap = i \* BLOCK\_NUMBER;
11. blockGroups[i].iNodeTable = i \* BLOCK\_NUMBER;
12. }
13. **for** (i = 0; i < INODE\_NUMBER; i++)
14. {
15. iNode\_bitmap[i] = NOT\_USED;  //初始化i结点位图
16. blocks\_bitmap[i] = NOT\_USED;  //初始化盘块位图
17. //初始化i结点表
18. iNode\_table[i].BlockAddr = 0;
19. iNode\_table[i].BlockNumber = 0;
20. iNode\_table[i].fileType = -1;
21. iNode\_table[i].Name = "new file";
22. iNode\_table[i].size = 0;
23. iNode\_table[i].father\_iNode = -1;
24. }
25. //用户列表
26. **for** (i = 0; i < MAX\_USER; i++)
27. {
28. users[i].uName = "";
29. users[i].password = "";
30. users[i].uType = NOT\_USED;
31. }
32. //当前用户为root用户
33. users[0].uName = "root";
34. users[0].password = "password";
35. users[0].uType = ROOT\_USER;

38. //当前目录为根目录
39. curDir.iNode = 0;
40. curDir.iNode\_number = 0;
42. blockGroups[0].freeBlocks -= DIR\_SIZE;
43. blockGroups[0].freeiNode -= 1;
44. **for** (i = 0; i < DIR\_SIZE; i++)
45. {
46. blocks\_bitmap[i] = USED;
47. }
48. iNode\_bitmap[0] = USED;
49. iNode\_table[0].BlockAddr = DATA\_ADDR;
50. iNode\_table[0].BlockNumber = DIR\_SIZE;
51. iNode\_table[0].fileType = DIRECTORY\_TYPE;
52. iNode\_table[0].Name = "root";
53. iNode\_table[0].size = **sizeof**(Dir);
54. iNode\_table[0].father\_iNode = 0;
55. iNode\_table[0].uNumber = 0;

将初始化好的信息写入到文件，char大小为1B，所以分配1024 \* 1024 \* 100个char的空间即为100MB。

1. //保存到文件
2. f\_simdisk.open("simdisk", ios::out | ios::binary);
3. **if** (!f\_simdisk) {
4. cout << "error\_disk\_init" << endl;
5. }
6. **for** (i = 0; i < MAX\_USER; i++)  //用户列表
7. {
8. f\_simdisk.write(**reinterpret\_cast**<**char** \*>(&users[i]), **sizeof**(User));
9. }
10. **for** (i = 0; i < BLOCK\_GROUP\_NUMBER; i++) //数据块组
11. {
12. f\_simdisk.write(**reinterpret\_cast**<**char** \*>(&blockGroups[i]), **sizeof**(BlockGroup));
13. }
14. **for** (i = 0; i < INODE\_NUMBER; i++)  //i-结点位图
15. {
16. f\_simdisk.write(**reinterpret\_cast**<**char** \*>(&iNode\_bitmap[i]), **sizeof**(**int**));
17. }
18. **for** (i = 0; i < INODE\_NUMBER; i++)  //i-结点表
19. {
20. f\_simdisk.write(**reinterpret\_cast**<**char** \*>(&iNode\_table[i]), **sizeof**(i\_Node));
21. }
22. **for** (i = 0; i < BlOCKS; i++)   //盘块位图
23. {
24. f\_simdisk.write(**reinterpret\_cast**<**char** \*>(&blocks\_bitmap[i]), **sizeof**(**int**));
25. }
26. //根目录
27. f\_simdisk.seekp(iNode\_table[curDir.iNode].BlockAddr, ios::beg);
28. f\_simdisk.write(**reinterpret\_cast**<**char** \*>(&curDir), **sizeof**(Dir));
29. //分配100MB数据空间
30. **int** dataSize = BlOCKS \* BLOCK\_SIZE;
31. **char** \*buffer = **new** **char**[dataSize];
32. f\_simdisk.write(**reinterpret\_cast**<**char** \*>(buffer), **sizeof**(**char**) \* dataSize);
33. f\_simdisk.close();
34. **delete**(buffer);

磁盘初始化完成



**函数及功能**

function.h

|  |  |
| --- | --- |
| 函数 | 功能 |
| void disk\_init(); | //初始化 |
| void read\_info(); | //读取磁盘信息 |
| string get\_path(Dir); | //返回目录Dir的完整路径 |
| int check\_iNode(); | //返回可用的i-结点号 |
| int check\_blocks(int size); | //根据需要的盘块数size，返回可用的盘块号 |
| int find\_by\_pathName(string pathName); | //根据路径pathName，返回最终文件或目录的i-结点号 |
| int addr\_to\_bitmap(int); | //地址到盘块号的转换 |
| void delete\_file(int); | //删除文件，参数为i-结点号 |
| string host\_path(string pathName); | //宿主机路径转换，<host>去掉 |
| string host\_filename(string pathName); | //根据路径获取宿主机文件名 |
| int login(); | //登录 |
| void add\_user(string uName, string password, string uType); | //添加用户，参数为用户信息 |
| void delete\_user(string uName); | //参数为用户名，删除用户 |

order.h

|  |  |
| --- | --- |
| 函数 | 功能 |
| void process(string order); | //处理输入的命令，执行相应的命令 |
| void order\_info(); | //执行info命令 |
| void order\_cd(string pathName); | //执行cd命令，cd到路径pathName |
| void order\_md(string dirName, Dir &mdDir); | //执行md命令，在mdDir目录下创建目录，目录名为dirName |
| void order\_dir(); | //执行dir命令 |
| void order\_rd(string pathName); | //执行rd命令，删除pathName的目录或文件 |
| void order\_newfile(string filename); | //执行new\_file命令，文件名为filename |
| void order\_cat(string filename); | //执行cat命令，文件名为filename |
| void order\_copy(string, string); | //执行copy命令，将第一个参数的文件复制到第二个参数的路径 |
| void order\_copy\_host\_to\_sim(string, string); | //从宿主机copy到simdisk的函数 |
| void order\_copy\_sim\_to\_host(string, string); | //从simdisk复制到宿主机的函数 |
| void order\_users(); | //users命令，显示当前所有用户 |
| void order\_check(); | //执行check命令 |

**关键算法及实现**

i-结点：

因为i-结点储存了文件的信息，所以，可以通过i-结点来直接或间接获取文件或目录的所有信息。所以，查找文件或目录时的关键在于找到文件或目录的i-结点号。

通过目录获取完整路径：

函数get\_path(curDir)可以通过目录获取完整路径。因为i-结点记录了父目录的i-结点号，所以，只需要循环查找可以很轻易到获取到目录的完整路径。

处理输入的命令：

先通过函数get\_path(curDir)输出当前目录的完整路径。

循环获取输入命令后，利用函数process(order)处理。函数内使用if-else结构判断命令，然后进入对应的函数。

while (1)

{

cout << get\_path(curDir) << "/>";

cin >> order;

if (order == "EXIT" || order == "exit")

{

break;

}

process(order);

}

路径处理：

因为路径由/分隔，路径使用string类型，所以，使用string的find函数，循环查找/的位置，以确定路径所表示的文件或目录。

过程实现：

1. **int** pos = 0, n = curDir.iNode\_number, diriNode; //pos为目录名开始的位置
2. Dir tempDir = curDir;  //tempDir记录当前目录
3. string str;   //tempDir记录当前目录
4. **while** (pathName.find('/', pos) != pathName.npos) //循环查找/
5. {
6. str = pathName.substr(pos, pathName.find('/', pos) - pos);  //目录名开始位置pos到/的之间的内容为目录名
7. n = tempDir.iNode\_number;   //当前目录的子文件
8. **while** (n)   //循环查找当前目录下是否存到目录名为str的目录
9. {
10. **if** (iNode\_table[tempDir.iNodes[n - 1]].Name == str)
11. {
12. diriNode = tempDir.iNodes[n - 1]; //若找到，则记录i-结点号
13. **break**;
14. }
15. n--;
16. }
17. **if** (n == 0) //目录不存在
18. {
19. cout << "不存在目录或文件" << str << endl;
20. **return** -1;
21. }
22. //若存在则根据i-结点号从磁盘读取目录为当前目录
23. f\_simdisk.seekg(iNode\_table[diriNode].BlockAddr, ios::beg);
24. f\_simdisk.read(**reinterpret\_cast**<**char** \*>(&tempDir), **sizeof**(Dir));
25. //下一个目录名开始的位置为/的下一个字符
26. pos = pathName.find('/', pos) + 1;
27. }

cd命令：

对输入的路径进行处理，即上述路径处理的过程，得到了最终目录的i-结点号，读取磁盘，将当前目录curDir设置为读取到的目录即可。

info命令：

直接输出超级块记录的磁盘的信息。

md命令：

过程：1.分配空间

2.创建新目录，修改分配到的i-结点的信息

3.修改超级块、i-结点位图、盘块位图信息

4.根据父目录，判断重命名。无重命名则修改父目录信息

5.修改过的信息写入磁盘

关键代码：

1. //分配空间
2. newiNode = check\_iNode();              //i-结点号
3. newBlocks = check\_blocks(DIR\_SIZE);    //盘块开始号
4. **if** (newBlocks == -1 || newiNode == -1)
5. {
6. cout << "磁盘空间不足" << endl;
7. }
8. //cout << newiNode << endl;
9. //cout << newBlocks << endl;
10. //父目录
11. mdDir.iNodes[mdDir.iNode\_number] = newiNode;
12. mdDir.iNode\_number++;//父目录子i-结点增加
14. //新目录
15. Dir newDir;
16. newDir.iNode = newiNode;
17. newDir.iNode\_number = 0;
18. **for** (i = 0, j = newBlocks; i < DIR\_SIZE; i++, j++) //盘块位图
19. {
20. blocks\_bitmap[j] = USED;
21. }
22. //i-结点
23. iNode\_bitmap[newiNode] = USED;
24. iNode\_table[newiNode].BlockAddr = DATA\_ADDR + (newBlocks \* BLOCK\_SIZE);
25. iNode\_table[newiNode].BlockNumber = DIR\_SIZE;
26. iNode\_table[newiNode].fileType = DIRECTORY\_TYPE;
27. iNode\_table[newiNode].Name = dirName;
28. iNode\_table[newiNode].size = **sizeof**(Dir);
29. iNode\_table[newiNode].father\_iNode = mdDir.iNode;
30. iNode\_table[newiNode].uNumber = curUser;
31. //超级块
32. **for** (i = 0; i < BLOCK\_GROUP\_NUMBER; i++)
33. {
34. blockGroups[i].sBlock.all\_freeBlocks -= DIR\_SIZE; //空闲块总数减目录的块数
35. blockGroups[i].sBlock.all\_freeiNodes -= 1;  //减目录的i结点
36. }

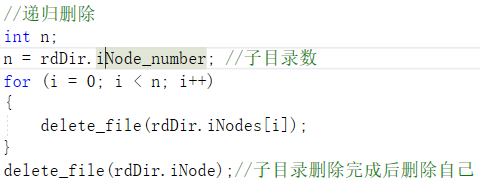
dir命令：

每个目录都有一个数组记录了自己的子文件的i-结点号，遍历数组，读取文件即可获取目录下的文件或目录。

rd命令

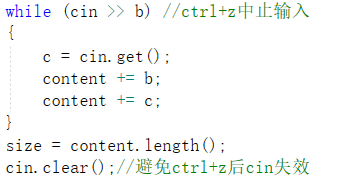
先通过路径找到要删除的目录。如果目录下没有子目录或文件。只要将目录的i-结点设为NOT\_USED，再将父目录的相关信息修改，即可认为该目录已删除。然后修改超级块和盘块位图的信息以释放空间。

如果目录下有文件或目录，需要提醒用户是否确认删除，用户确认后。对要删除的目录进行深度遍历删除。调用递归过程，删除子目录，直到要删除的目录为空，停止递归。



newfile命令

获取输入内容，cin.get()获取回车，输入完成后ctrl+z完成输入。



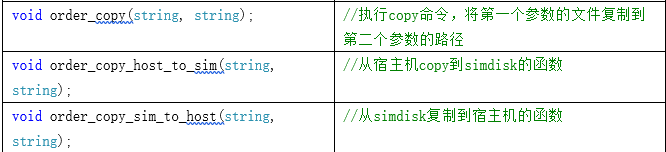
创建过程类似md，先分配空间，获取分配到的地址后，读取父目录判断重命名。修改父目录、超级块、位图信息。然后将修改过的信息和文件内容写入到磁盘从相应的位置。

cat命令

路径处理，根据路径得到文件的i-结点号。根据i-结点记录的地址和大小直接读取内容并输出。

copy命令

copy命令可分为三种。



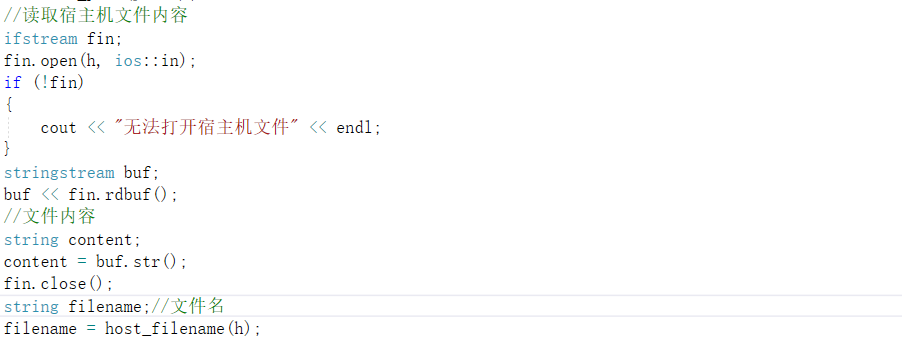
1.simdisk内部的复制。

因为前面已经实现了文件的写入和读取，即newfile和cat命令。所以，copy命令结合newfile和cat的代码再稍作修改即可。

函数host\_path()将宿主机地址前的<host>去掉

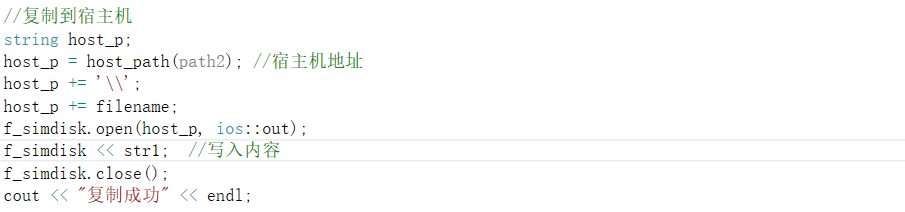
2.从宿主机复制到simdisk

先读取宿主机的文件内容，保存到string。然后利用newfile命令的代码修改即可。



3.从simdisk复制到宿主机

读取simdisk文件的内容（用cat命令的代码改），再写入到宿主机。

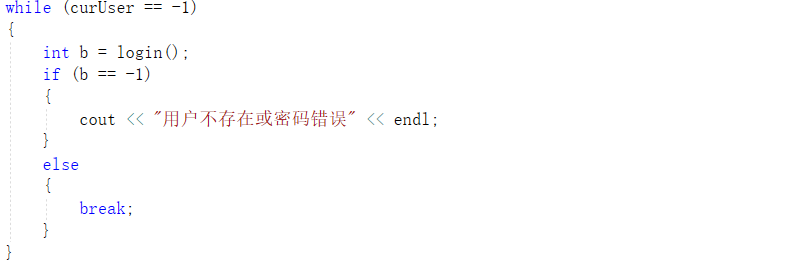


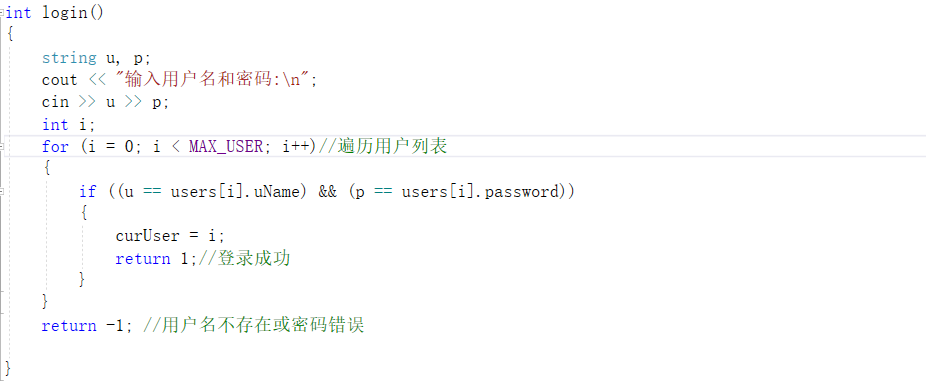
权限控制：

将用户分为3类，数值越大，权限越高。



使用系统前需要登录，登录成功后，将用curUser记录当前用户在用户列表的下标。

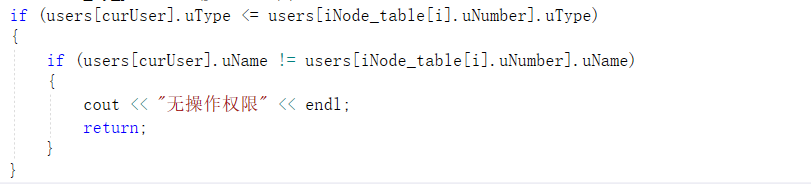




符合下列情况可以修改文件：

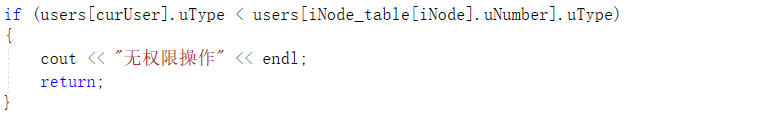
1.当前用户的权限大于文件所有者权限

2.当前用户权限等于文件所有者权限且用户名相同（即自己的文件）



符合下列情况可以访问文件：

1.当前用户权限大于等于文件所有者权限



root用户可以使用adduser命令创建新用户，参数为用户名 密码 用户类型

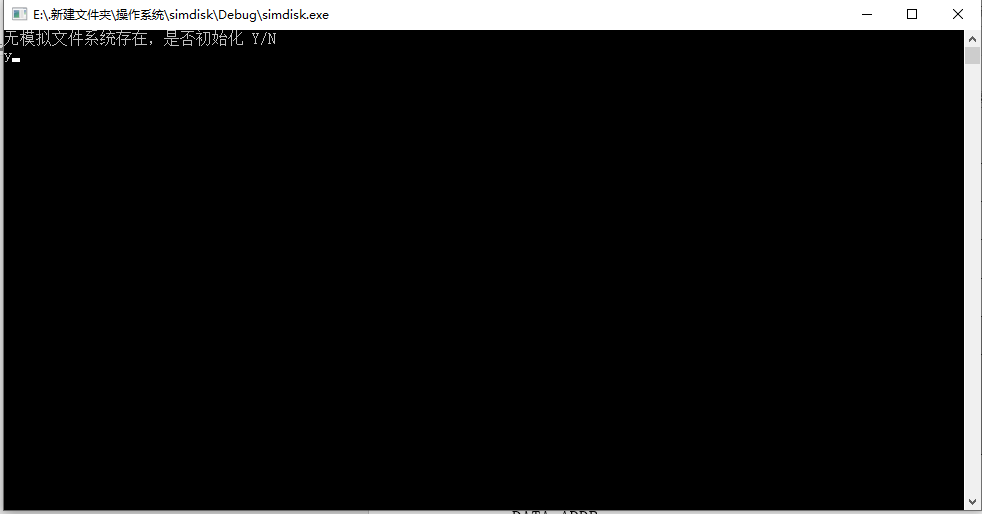
deluser命令可以删除用户。

添加和删除用户的操作只用root用户可以进行。

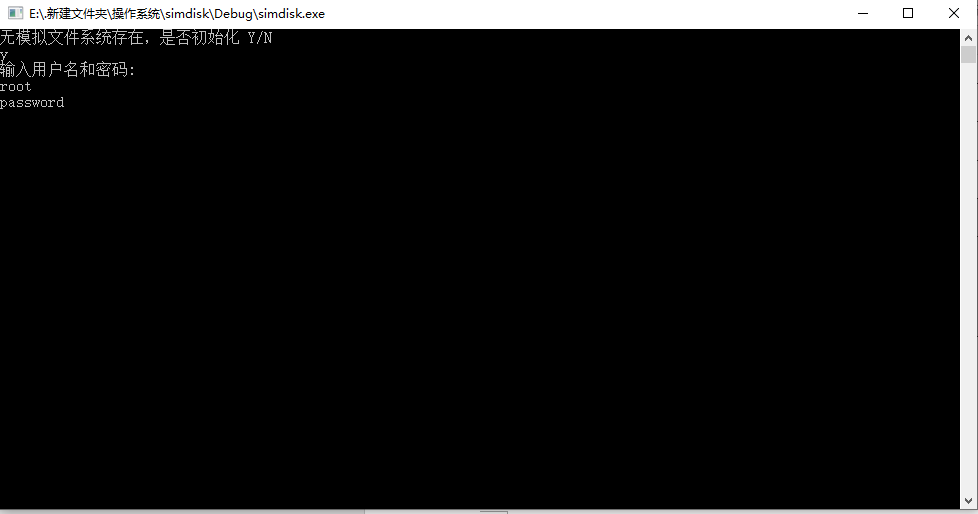


**程序测试**

如果没有找到simdisk文件，则需要初始化。



初始用户为root，密码是password



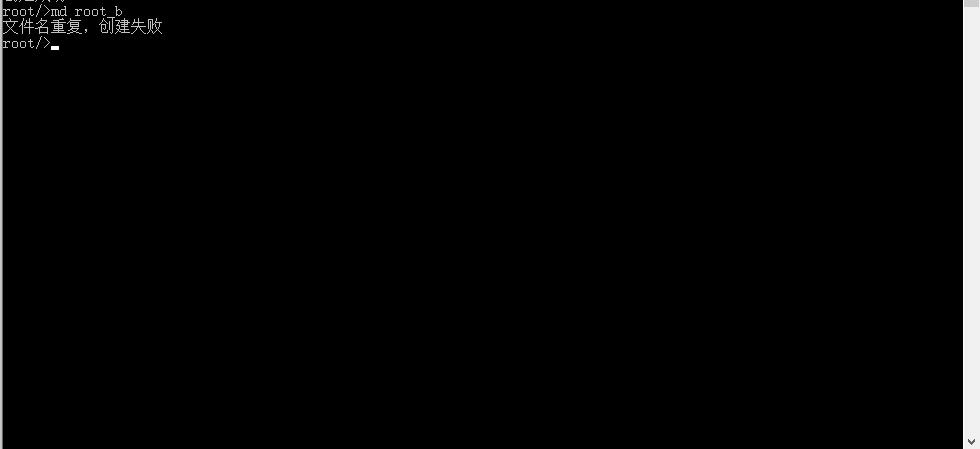
info命令，数据无误



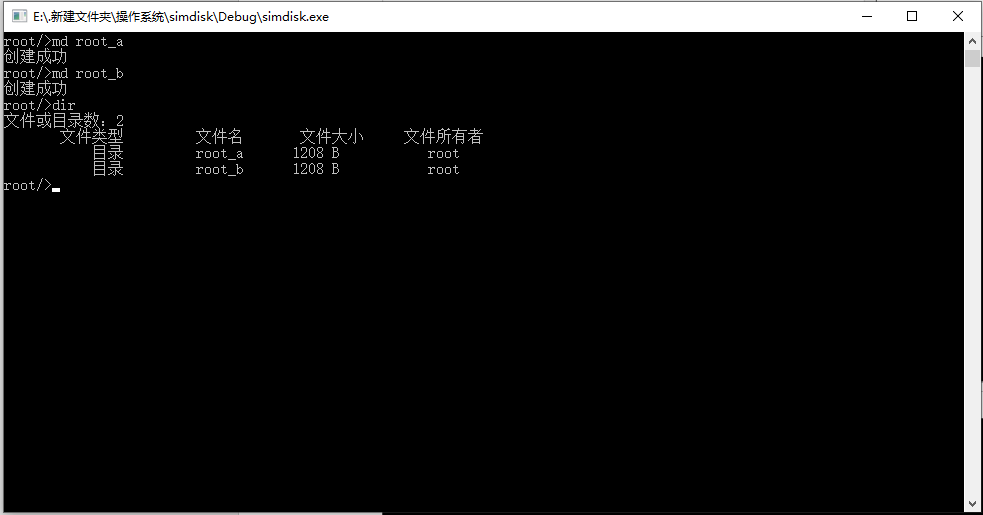
md命令，在root目录下创建目录root\_a, root\_b



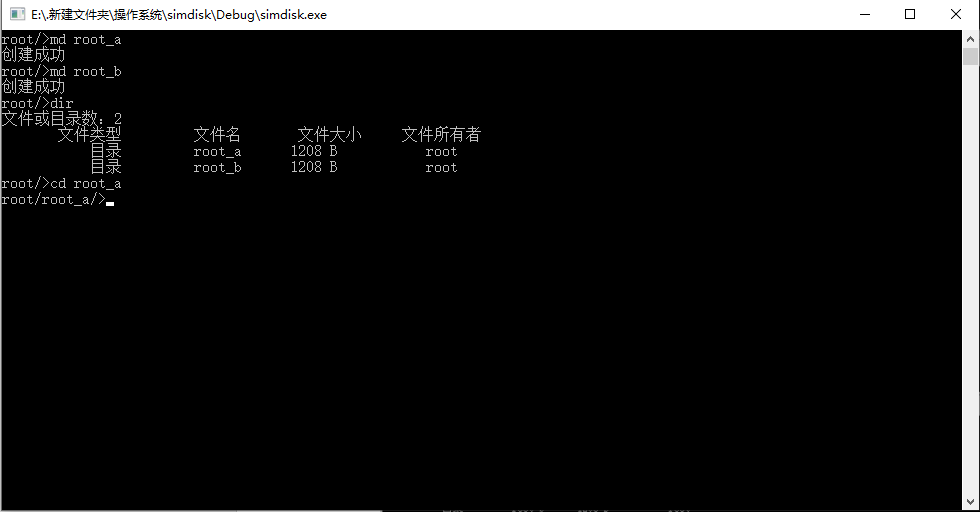
再次创建root\_b时，提示命名重复创建失败



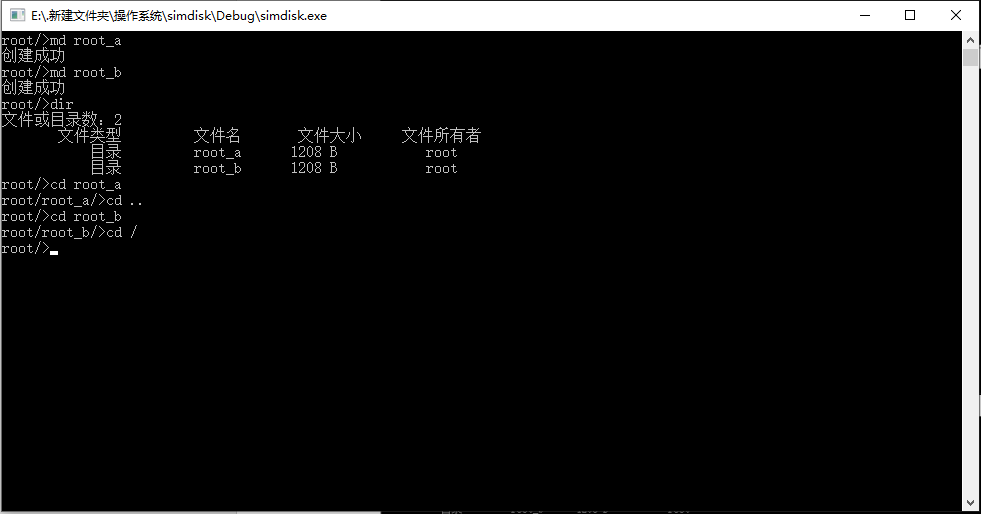
dir命令。显示子目录和文件



cd命令，进入root\_a目录，进入成功

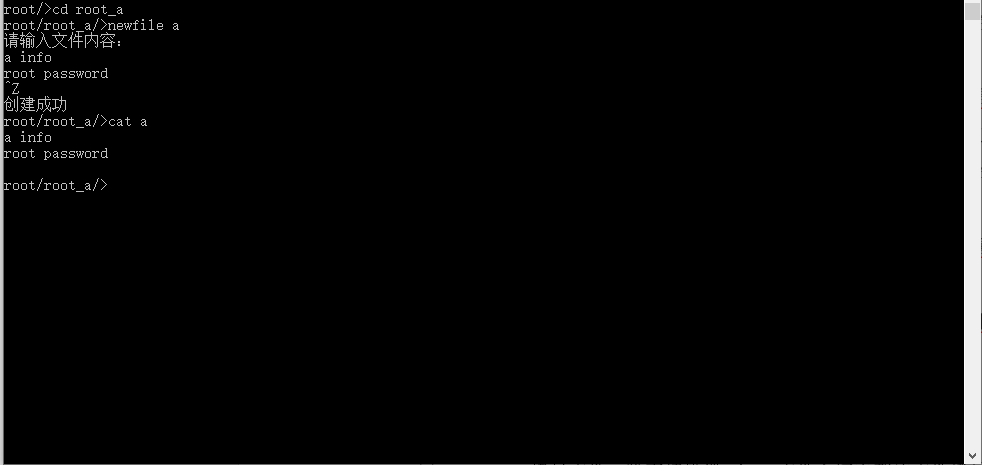


cd ..返回上一级目录，cd /返回根目录

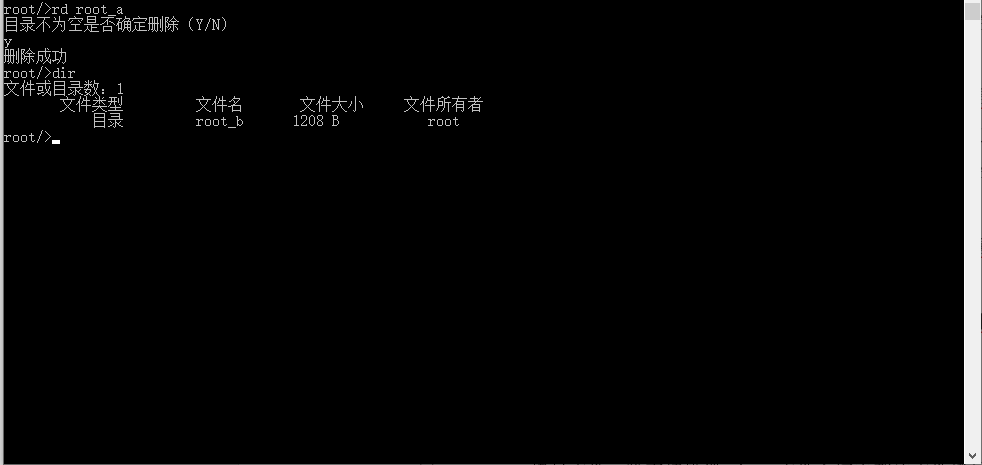


newfile 在目录root\_a下创建了文件a并输入文件内容

cat命令，读取文件内容并成功输出无误

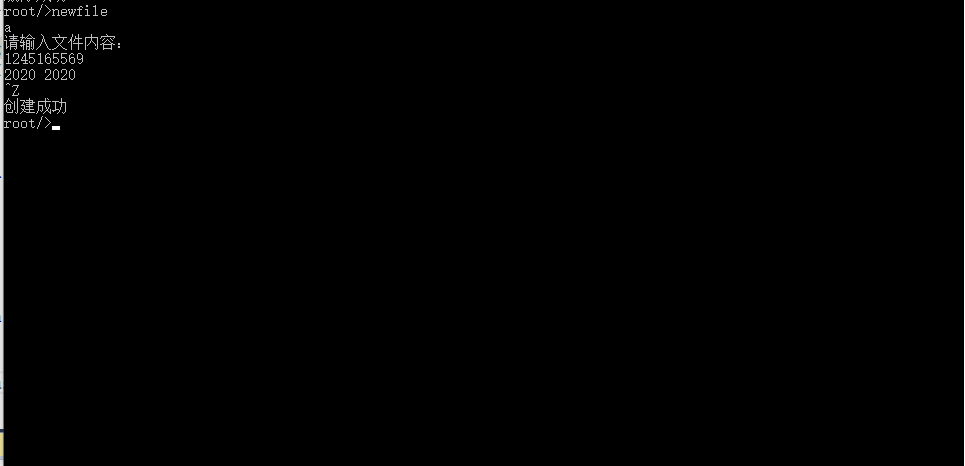


rd命令，删除root\_a，再用dir命令查看，确认删除成功

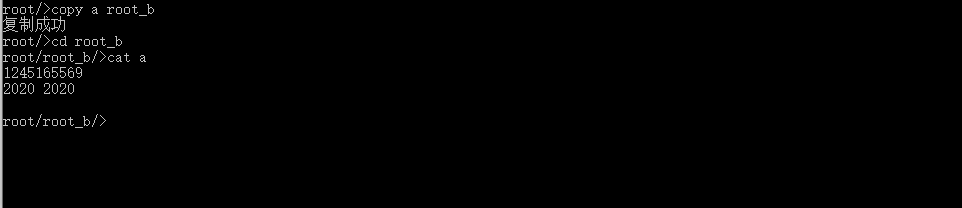


copy命令

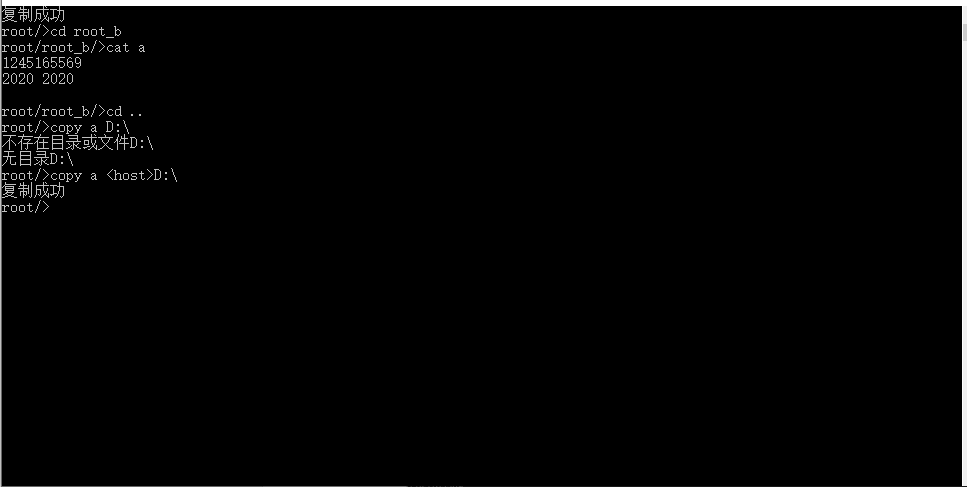
先在root目录下创建文件a



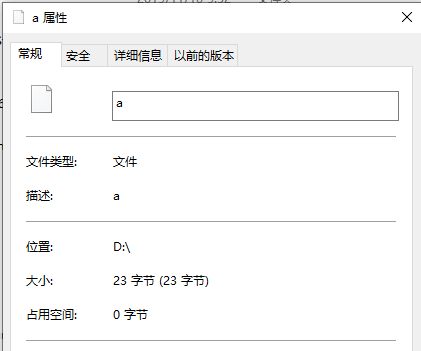
将文件a复制到目录root\_b，并cat a，内容一样

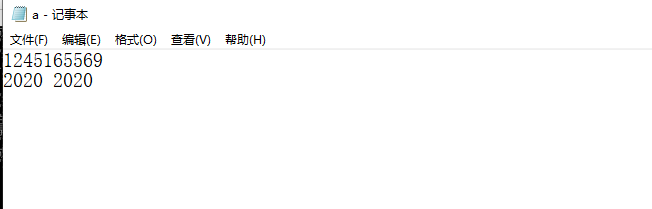


将文件a复制到宿主机D盘

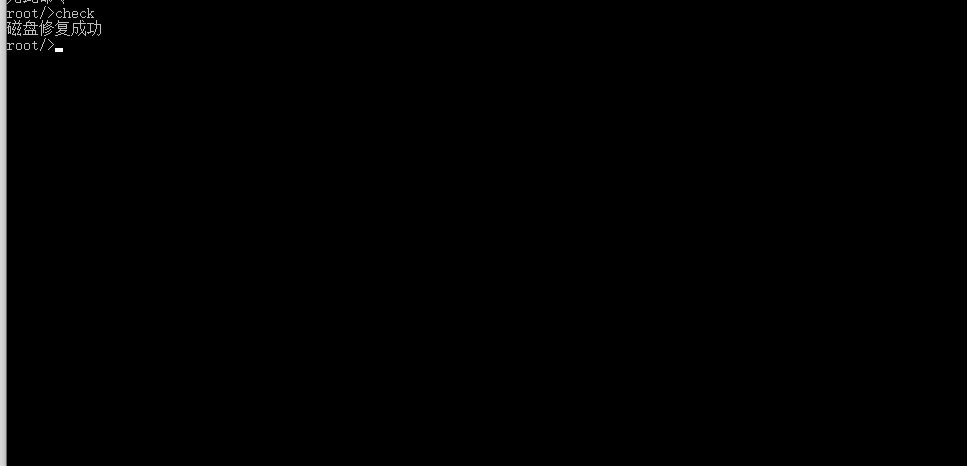


内容正确





check命令，将内存的信息写入到磁盘以修复磁盘的错误。



**实验总结**

通过这次实验用c++实现了一个简单的模拟linux文件系统。加深了对linux文件系统的认识，在实现的过程中，加强了对文件系统的知识的掌握，对文件系统的相关知识更加的熟悉，如i-结点、位图等的知识。理解了linux文件系统的管理方式。