海 南 大 学

信息科学技术学院

《操作系统》

课程设计

题 目：

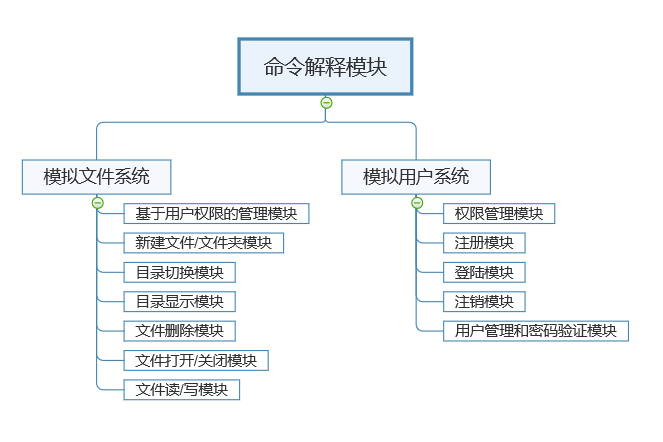
学 号：

姓 名：

专 业：

指导教师：

1. 算法设计
   1. 功能模块图



1.2算法思路

1.2.1实现方法

void help(); //命令目录

void shell(); //用来解释命令

void signup(); //用户注册

void login(string namein); //用户登录

bool check\_login(); //登陆状态检查

bool existrecentFolder(string filename); //检查文件是否在当前目录

void open(string namein); //打开文件

void read(string namein); //读文件

void write(string namein); //写文件

void creatFile(string name); //创建文件

void creatFolder(string name); //创建文件夹

void init(); //启动初始化

void cd(string address); //执行cd命令

void dir(r); //执行dir命令

void deleteFile(string namein); //执行del命令

void exit(); //退出

bool user::creat\_account();//新建用户

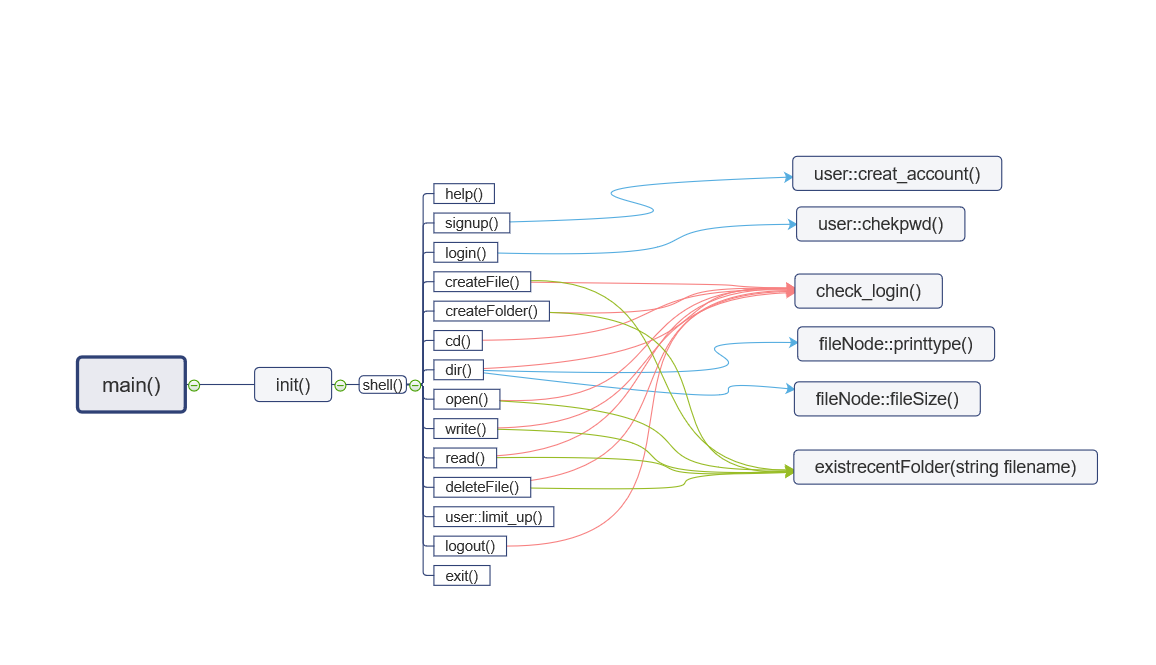
bool user::chekpwd(); //密码检查

void user::limit\_up(); //用户权限提升

bool fileNode::changeType(); //文件读写属性改写

int fileNode::fileSize(); //文件大小获取

string fileNode::printtype(); //输出读写属性



函数调用图

1.2.2 设计思想

建立一刻目录树，根据输入的命令字符串，对该目录树进行增删等操作。

**（1）定义全局变量**

fileNode \*recent\_ptr; //指向当前目录节点

fileNode root; //文件系统根节点/root

int recent\_grade; //recent\_ptr所指向的文件相对于/root的层级（root为0）

list <user> userList; //用户列表

string loginId; //目前已经登陆的用户的ID

**（2）主函数模块**

调用init模块初始化，然后使用while循环，调用shell模块进行命令输入解释并执行操作，执行完成后再次进入循环，直到用户使用exit退出。

**（3）其他模块说明（用文字描述，不要代码）**

删除模块：调用existrecentFolder函数检索节点是否存在于目录，然后进行删除节点操作，并注意特殊情况。

新建目录/文件模块：调用existrecentFolder函数检索节点是否存在于目录，若不存在就进行增加节点。

切换目录模块：调用existrecentFolder函数检索节点是否存在于目录，若存在则令recent\_ptr等于它的地址

显示目录模块：循环遍历该节点的孩子并显示。

退出模块：执行exit函数。

1.2.3 重要代码

#include<iostream>

#include<string>

#include<list>

#include<iomanip>

#include <cstdlib>

#include<fstream>

#include<sys/stat.h>

#include<sys/types.h>

#include<fcntl.h>

#include<cstdio>

#include<unistd.h>

using namespace std;

class fileNode; //文件节点 用来存储文件的信息 并以树的形式构成文件系统

class user; //文件系统的用户，用于实现简单的依据用户权限的访问

string loginId; //目前已经登陆的用户的ID

list <user> userList; //用户列表

int recent\_grade; //recent\_ptr所指向的文件相对于/root的层级（root为0）

class fileNode

{

public:

fileNode \* father;

list<fileNode> children; string owner;

string name;

bool type[3] = { true,true,true }; //rwx权限，默认为true,true,true

string src;

int grade;

int filetype;

bool changeType()

{

if (loginId == owner || loginId == "root")

{

string a;

cout << "输入文件读写状态码，默认状态为-r-w-x(000)" << endl;

cin >> a;

if (a[0] = 0)

type[0] = false;

else

type[0] = true;

if (a[1] = 0)

type[1] = false;

else

type[1] = true;

if (a[2] = 0)

type[2] = false;

else

type[2] = true;

return true;

}

return false;

};

int fileSize()

{

struct stat statbuf;

stat(name.data(), &statbuf);

int size = statbuf.st\_size;

return size;

};

string printtype()

{

string a;

if (type[0])

a += "r";

else

a += "-r";

if (type[1])

a += "w";

else

a += "-w";

if (type[0])

a += "x";

else

a += "-x";

return a;

}

};

class user

{

public:

string user\_id;

void limit\_up()

{

cout << "提权中。。。" << endl;

limit = "root";

cout << "成功" << endl;

}

void limit\_up\_to\_root()

{

limit = "root";

}

bool creat\_accout()

{

string namein;

cout << "要注册的用户名：";

cin >> namein;

for (auto i = userList.begin(); i != userList.end(); i++)

if (i->user\_id == namein)

{

cout << "错误！该用户已存在" << endl;

return false;

}

passwd.inputPassword();

if (user\_id != "root")

{

user\_id = namein;

limit = user\_id;

cout << "用户创建成功!";

getchar();

return true;

}

else

{

cout << " \'root\'账户是系统保留账户" << endl;

return false;

}

};

bool checkpwd()

{

string password;

password = getpass("请输入密码:");

if (password == passwd.outputpass())

return true;

else

return false;

}

private:

string limit;

class Password

{

public:

Password()

{

psw = "root";//初始化密码为空"";

}

void inputPassword()

{

psw = getpass("请输入要注册的用户的密码：");

}

string outputpass()//返回一个密码字符串

{

return psw;

}

private:

string psw;//用于存字条串密码的字符串；

}passwd;

};

fileNode \*recent\_ptr; //目前所在目录

fileNode root; //文件系统根节点/root

void help()

{

cout << "------------------------------------------------------------------------\n";

cout << "函数表:\n\t1. signup 注册\n\t2. login + user\_id 登陆id\n\t3. touch + file\_name 新建文件\n\t4. mkdir + folder\_name 新建文件夹\n\t5. cd + 相对地址 （目前只支持一层）\n\t6. dir等效ls -l \n\t7. open + file\_name 打开文件\n\t8. write + file\_name 编辑文件\n\t9. read + file\_name 读取文件 \n\t10. delete + file\_name 删除文件/文件夹 \n\t11. promote 当前用户提权 \n\t12. logout 注销当前用户\n\t13. exit 退出系统\n";

cout << "------------------------------------------------------------------------\n";

cout << endl;

}

//登陆检查

bool check\_login()

{

if (loginId.size() != 0)

return true;

return false;

}

//扫描输入的文件是否在当前目录中

bool existrecentFolder(string filename)

{

for (auto i = recent\_ptr->children.begin(); i != recent\_ptr->children.end(); i++)

if ((\*i).name == filename)

return true;

return false;

}

//打开文件

void open(string namein)

{

if (!check\_login())

{

cout << "请先登录" ;

return;

}

if (existrecentFolder(namein))

{

for (auto i = recent\_ptr->children.begin(); i != recent\_ptr->children.end(); i++)

if ((\*i).name == namein && (\*i).filetype == 1) //类型判断

{

if (((\*i).owner == "all" || (\*i).owner == loginId || loginId == "root") && ((\*i).type[0] && (\*i).type[1]))

{

ofstream filestream(namein.data());

cout << "访问成功!";

}

else

{

cout << "权限检查失败" ;

return;

}

}

else if ((\*i).name == namein && (\*i).filetype == 0)

{

cout << "不支持对文件夹操作";

return;

}

}

else

cout << "未找到文件" ;

}

//读文件

void read(string namein)

{

if (!check\_login())

{

cout << "请先登录";

return;

}

if (existrecentFolder(namein))

{

for (auto i = recent\_ptr->children.begin(); i != recent\_ptr->children.end(); i++)

if ((\*i).name == namein && (\*i).filetype == 1)

{

if (((\*i).owner == "all" || (\*i).owner == loginId || loginId == "root") && (\*i).type[0])

{

ifstream filestream(namein.data());

if (!filestream.is\_open())

{

cout << "文件打开失败"; exit(1);

}

while (!filestream.eof())

{

cout << "读取到的内容是:";

string content;

getline(filestream, content);

cout << content;

}

}

else

{

cout << "权限检查失败";

return;

}

}

else if ((\*i).name == namein && (\*i).filetype == 0)

{

cout << "不支持对文件夹操作" ;

return;

}

}

else

cout << "未找到文件";

}

//写文件

void write(string namein)

{

if (!check\_login())

{

cout << "请先登录";

return;

}

if (existrecentFolder(namein))

{

for (auto i = recent\_ptr->children.begin(); i != recent\_ptr->children.end(); i++)

if ((\*i).name == namein && (\*i).filetype == 1)

{

if (((\*i).owner == "all" || (\*i).owner == loginId || loginId == "root") && (\*i).type[1])

{

ofstream filestream(namein.data());

if (filestream.is\_open())

{

string content;

cout << "输入要写入的内容:"<<endl ;

cin >> content;

filestream << content ;

filestream.close();

getchar();

cout << "写入成功！";

}

}

else

{

cout << "权限检查失败";

return;

}

}

else if ((\*i).name == namein && (\*i).filetype == 0)

{

cout << "不支持对文件夹操作";

return;

}

}

else

cout << "未找到文件" ;

}

//新建文件

void creatFile(string name)

{

if (!check\_login())

{

cout << "请先登录";

return;

}

fileNode IN;

if ((loginId == recent\_ptr->owner || recent\_ptr->owner == "all"||loginId=="root") && recent\_ptr->type[1])

{

if (!existrecentFolder(name))

{

IN.owner = loginId;

IN.father = recent\_ptr;

IN.name = name;

IN.src = recent\_ptr->src + "/" + name;

IN.grade = recent\_grade + 1;

IN.filetype = 1;

IN.type[0] = true;

IN.type[1] = true;

IN.type[2] = true;

recent\_ptr->children.push\_back(IN);

string command = "touch " + name;

system(command.data());

cout << "创建成功！";

}

else

{

cout << "命名冲突" ;

}

}

else

cout << "权限错误";

}

//新建文件夹

void creatFolder(string name)

{

if (!check\_login())

{

cout << "请先登录" ;

return;

}

fileNode IN;

if ((loginId == recent\_ptr->owner || recent\_ptr->owner == "all" || loginId == "root") && recent\_ptr->type[1])

{

if (!existrecentFolder(name))

{

IN.owner = loginId;

IN.father = recent\_ptr;

IN.name = name;

IN.src = recent\_ptr->src + "/" + name;

IN.grade = recent\_grade + 1;

IN.filetype = 0;

IN.type[0] = true;

IN.type[1] = true;

IN.type[2] = false;

recent\_ptr->children.push\_back(IN);

string command = "mkdir " + name;

system(command.data());

cout << "创建成功！";

}

else

{

cout << "命名冲突" ;

return;

}

}

else

cout << "权限错误";

}

//初始化

void init()

{

//创建根目录

{

root.owner = "all";

root.filetype = 0;

root.src = "./root";

root.grade = 0;

root.name = "root";

system("mkdir root");

chdir("root");

recent\_grade = root.grade;

recent\_ptr = &root;

}

{

user su;

su.user\_id = "root";

su.limit\_up\_to\_root();

userList.push\_back(su);

}

}

//登陆

void login(string namein)

{

bool flag;

if (loginId == namein)

{

cout << "您已登录，请勿重复尝试";

return;

}

else if (loginId.size() != 0)

{

cout << "请先注销当前账户" ;

return;

}

else

{

for (auto i = userList.begin(); i != userList.end(); i++)

if ((\*i).user\_id == namein)

{

int count = 3;

if ((\*i).checkpwd())

{

flag = true;

break;

}

else

{

do

{

cout << "密码错误，你还有" << count-2 << "次机会" << endl;

count--;

flag = false;

if (count == 0)

break;

} while (!(\*i).checkpwd());

}

}

else

flag = false;

if (flag)

{

loginId = namein;

cout << "登录成功!" ;

}

else

cout << "登陆失败" ;

}

}

//注册

void signup()

{

user newUser;

if (newUser.creat\_accout())

userList.push\_back(newUser);

}

//cd命令（只实现了一层目录）

void cd(string address)

{

bool flaga = false, flagb = false;

list<fileNode>::iterator a;

if (!check\_login())

{

cout << "请先登录";

return;

}

if (address == "..")

{

recent\_grade--;

recent\_ptr = recent\_ptr->father;

chdir("..");

}

else if (address == ".");

else

{

for (auto i = recent\_ptr->children.begin(); i != recent\_ptr->children.end(); i++)

{

if ((\*i).name == address)

{

flaga = true;

if ((\*i).filetype == 0)

{

flaga = true;

flagb = true;

a = i;

break;

}

else

flagb = false;

}

else

if (flagb != true)

flagb = false;

}

if (flaga&&flagb)

{

recent\_grade++;

chdir(address.data());

recent\_ptr = &(\*a);

}

else if (flaga && !flagb)

{

cout << "不支持对文件操作";

return;

}

else if (!flaga)

{

cout << "未找到文件夹" ;

return;

}

}

cout << "切换成功!";

}

//ls

void dir()

{

if (!check\_login())

{

cout << "请先登录" ;

return;

}

if (recent\_ptr->children.size() != 0)

{

cout << std::left << setw(18) << "文件名" << std::left << setw(25) << "文件所有者" << std::left << setw(36) << "文件读写类型" << std::left << setw(24) << "文件地址" << std::left << setw(19) << "文件大小" << endl;

for (auto i = recent\_ptr->children.begin(); i != recent\_ptr->children.end(); i++)

cout << std::left << setw(15) << (\*i).name << std::left << setw(20) << (\*i).owner << std::left << setw(30) << (\*i).printtype() << std::left << setw(20) << (\*i).src << std::left << setw(15) << (\*i).fileSize() << endl;

}

else

cout << "空目录";

}

//删除文件

void deleteFile(string namein)

{

if (!check\_login())

{

cout << "请先登录" ;

return;

}

if (existrecentFolder(namein))

{

for (auto i = recent\_ptr->children.begin(); i != recent\_ptr->children.end(); i++)

if (((\*i).owner == "all" || (\*i).owner == loginId || loginId == "root") && (\*i).type[1])

{

if ((\*i).name == namein)

{

recent\_ptr->children.erase(i);

string command = "rm -rf " + namein;

system(command.data());

cout << "删除成功!";

return;

}

}

else

{

cout << "权限检查失败" ;

return;

}

}

else

cout << "未找到文件" ;

}

//注销用户

void logout() {

if (!check\_login())

{

cout << "请先登录" ;

return;

}

loginId.clear();

cout << "用户已退出!";

}

//退出

void exit()

{

for (int i= recent\_grade+1; i != 0; i--)

chdir("..");

system("rm -rf root");

cout << "目录删除成功!即将退出系统."<<endl;

std::exit(0);

}

//命令解释

void shell()

{

int orders;

string order, option, command;

cout << ">";

getline(cin, order);

for (auto i = 0; i != order.size(); i++)

if (order[i] == ' '&&i != order.size() - 1)

{

command = order.substr(0, i);

option = order.substr(i + 1, order.size());

break;

}

else if (i == order.size() - 1)

{

command = order;

break;

}

if (command == "help")

orders = 0;

else if (command == "signup")

orders = 1;

else if (command == "login")

orders = 2;

else if (command == "touch")

orders = 3;

else if (command == "mkdir")

orders = 4;

else if (command == "cd")

orders = 5;

else if (command == "dir")

orders = 6;

else if (command == "open")

orders = 7;

else if (command == "write")

orders = 8;

else if (command == "read")

orders = 9;

else if (command == "delete")

orders = 10;

else if (command == "promote")

orders = 11;

else if (command == "logout")

orders = 12;

else if (command == "exit")

orders = 13;

else

orders = -1;

switch (orders)

{

case 0:help(); break;

case 1:signup(); break;

case 2:login(option); break;

case 3:creatFile(option); break;

case 4:creatFolder(option); break;

case 5:cd(option); break;

case 6:dir(); break;

case 7:open(option); break;

case 8:write(option); break;

case 9:read(option); break;

case 10:deleteFile(option); break;

case 11:for (auto i = userList.begin(); i != userList.end(); i++)

if ((\*i).user\_id == loginId)

(\*i).limit\_up();

break;

case 12:logout(); break;

case 13:exit(); break;

default:cout << "无效命令" ; break;

}

cout << endl;

}

int main()

{

init();

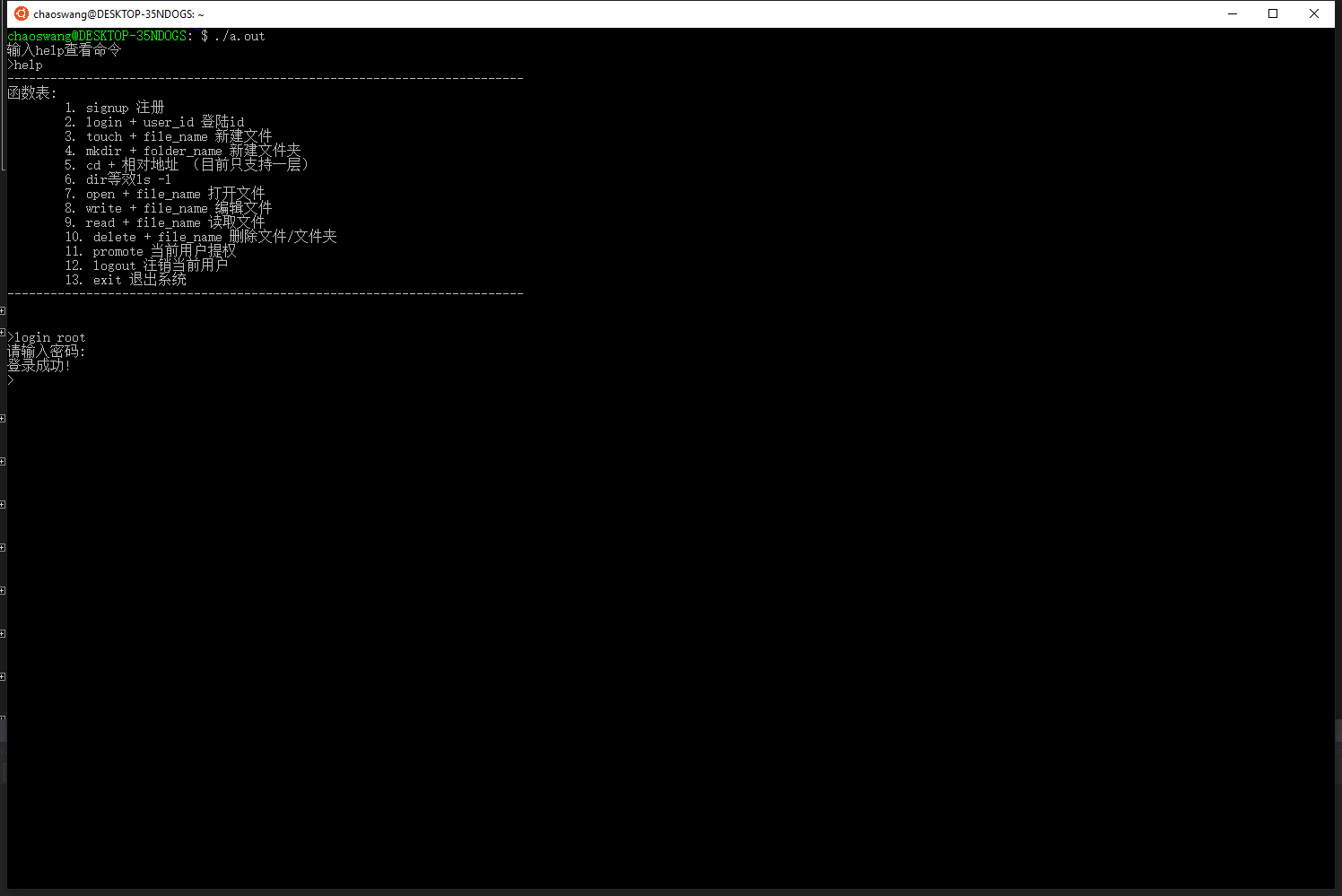
cout << "输入help查看命令" << endl;

while (1)

shell();

return 0;

}1.3 效果演示



1.4 反思

1、只实现了预想中的部分功能，并且许多功能还有不完善的地方；

2、代码规范，整洁程度有待提高；

3、部分功能的实现过程中为了能够尽快完成，使用了一些可能存在可移植性或安全性问题的方法；

4、shell()函数的建立使用了很笨拙的方法，使得要增添新功能时工作量大。

教师评语：

得分：