# 操作系统课程设计实验报告

LinuxFileMangaer模拟文件管理系统

姓名：黄金凯 学号：201526810206 班级：计算机实验班1501

1. **实验环境**

操作系统：Linux(运行)，Windows 10(开发)

开发工具：Visual Studio 2017，git

开发语言：C++

代码地址：<https://github.com/ProphetHJK/LinuxFileManager>

1. **实验设计**

**2.1 简介**

本次实验实现了一个模拟的文件管理系统，采用树状设计。系统运行在Linux操作系统下，所有的操作发生在内存中，没有使用Linux本身的文件系统。相关的功能及指令完全仿照Linux系统，如rm，mv，ls等，但由于时间与精力有限，对部分功能进行了简化，还有部分功能未实现。同时为了保证兼容性，所有提示信息都使用英文。

本项目一共定义了三个类，

knode类：文件信息类，类成员包含nodeid、文件名、大小、创建时间、修改时间、父knode指针、子knode指针向量、是否为目录、文本信息指针等，在linux的文件系统实现里，inode类和文件信息类是分开的，因为有链接功能的存在，一个文件可以对应多个inode，为了减少工作量将两个类写到了一起。每一个文件对应一个knode，每次创建时会分配一个独立nodeid号，每次修改时会导致修改时间和大小的变动。文件信息类不包含文本信息，只包含了文本数据的指针。由于使用long类型表示文件大小，单个文件大小上限为4GB。

Context类：文本内容类，由于是在内存中操作，不得不建立了这个类，如果不在内存中，文本内容可以直接放入磁盘里。Context类目前只实现类字符串类型，可以扩充为二进制类型，整数类型等。当文件信息被删除时，对象会被自动删除，并释放空间。实际上，如果文件是在磁盘中则不需要删除这个操作，只需删除文件信息就行，之后如果空间不够会覆盖原有被删文件。

FileManager类：文件管理类，用于knode，类成员包含knode指针向量（类似inode表），最大空间，可用空间，已用空间，当前目录等。文件及目录的增、删、移、改都由类成员函数提供。每次初始化是都会建立一个根目录目录名为/，之后建立的文件及目录都只能从属于根目录。数据结构方面用指针向量制作了knode表，记录所有文件信息（包括目录），新建文件进表，删除文件出表，复制移动同理。由于用户使用时是按照文件名使用的，所以要使用findFile()来寻找文件名对应的knode。由于文件和目录的不同，相关函数也区分文件和目录版本。比如删除函数，当删除目录时需要递归删除所有子目录和子文件，复制移动也是同理。

**2.2 实现功能**

使用help指令能在系统中显示所有可用指令，以下斜体表示实现的指令，全大写表示参数，之后表示其功能介绍。

*ls* 显示当前目录下文件及目录

*pwd*  显示绝对路径

*cd DIRECTORY* 进入目录

*mkdir DIRECTORY*  创建一个新目录，如果目录不存在

*touch FILE* 创建一个新文件，如果文件不存在

*rm FILE* 移除文件

*rmd DIRECTORY* 递归移除目录

*mv SOURCE TARGET* 移动文件至目标目录

*mvd SOURCEDIRECTORY TARGET* (递归)移动目录至目标目录

*cp SOURCE TARGET* 复制文件至目标目录

*cpd SOURCEDIRECTORY TARGET* 递归复制目录到目标目录

*vi FILE* 编辑文件，需要输入新的文本

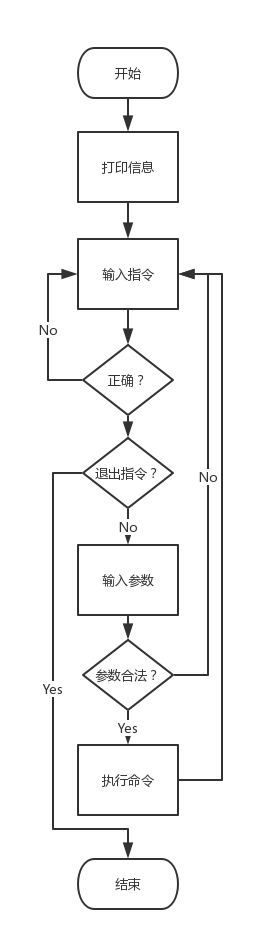
*cat FILE*  查看文件内容

*rename FILE TARGETNAME*  重命名文件

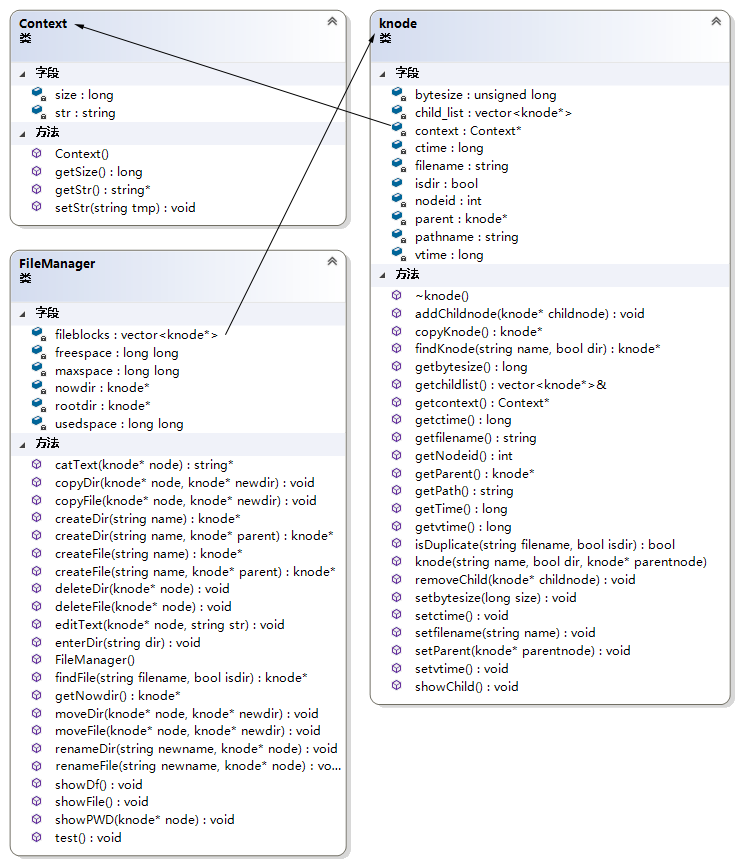
*renamed DIRECTORY TARGETNAME* 重命名目录

*df* 显示文件系统信息

**2.3 流程图**



**2.4 类图**



1. **代码分析**

**3.1 knode.h**

#include"Context.h"

using namespace std;

//inode的简化,单个文件的基本信息

class knode

{

private:

string filename;//文件名

string pathname;//绝对路径

int nodeid;//node编号，按照创建顺序指定

long ctime;//最后修改时间,也能表示创建时间

long vtime;//最后访问时间

unsigned long bytesize;//文件大小（字节）最大4GB

Context \*context;//文件内容

knode \*parent;//父节点的指针

vector<knode\*> child\_list;//子节点的向量指针

public:

//构造函数

knode(string name, bool dir, knode \*parentnode);

~knode();

//加入父节点，每次一个

void addChildnode(knode \*childnode);

//加入子节点，每次一个

void setParent(knode \*parentnode);

//移除子节点，每次一个

void removeChild(knode \*childnode);

bool isdir;//是否为目录

//显示目录下文件

void showChild();

//计算路径,返回路径string

string getPath();

//获取Unix时间戳

long getTime();

//目录下文件名是否重复

bool isDuplicate(string filename, bool isdir);

//检索子节点

knode\* findKnode(string name,bool dir);

//完全复制信息

knode\* copyKnode();

Context\* getcontext();

string getfilename();

long getvtime();

long getctime();

long getbytesize();

void setfilename(string name);

void setctime();

void setvtime();

void setbytesize(long size);

knode\* getParent();

int getNodeid();

vector<knode\*> &getchildlist();

};

**3.2 Context.h**

#include<cstdlib>

#include<string.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<sys/time.h>

#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

using namespace std;

class Context

{

private:

string str; //字符串信息，用于文本文件

long size; //占用大小

public:

Context();

void setStr(string tmp);

string\* getStr();

long getSize();

};

**3.3 FileManager.h**

#include "knode.h"

class FileManager

{

private:

long long maxspace;//最大空间

vector<knode\*> fileblocks;//记录文件信息

long long freespace;//剩余空间

long long usedspace;//已使用空间

knode \*nowdir;//当前目录

knode \*rootdir;//根目录knode

public:

//初始化

FileManager();

//删除文件

void deleteFile(knode \*node);

//删除文件

void deleteDir(knode \*node);

//创建文件

knode\* createFile(string name, knode \*parent);

//创建目录

knode\* createDir(string name, knode \*parent);

//创建文件在当前目录

knode\* createFile(string name);

//创建目录在当前目录

knode\* createDir(string name);

//重命名文件

void renameFile(string newname, knode \*node);

//重命名目录

void renameDir(string newname, knode \*node);

//移动文件

void moveFile(knode \*node,knode \*newdir);

//移动目录

void moveDir(knode \*node,knode \*newdir);

//复制文件

void copyFile(knode \*node, knode \*newdir);

//复制目录

void copyDir(knode \*node, knode \*newdir);

//显示当前目录下所有文件及目录

void showFile();

//进入目录

void enterDir(string dir);

//根据string解析knode

knode \*findFile(string filename,bool isdir);

//显示绝对路径

void showPWD(knode \*node);

//获取当前目录

knode\* getNowdir();

//文本内容修改

void editText(knode \*node,string str);

//文本内容获取

string \*catText(knode \*node);

//获取文件系统信息

void showDf();

void test();

};

**3.4 main.cpp**

using namespace std;

//交互界面

void StartMenu(FileManager \*filemanager)

{

string command;

string command1;

cout << "Welcome to LinuxFileManager!" << endl;

cout << "Use \"help\" to see what you can do." << endl;

while (1)

{

cout << "ubuntu@ubuntu:";

filemanager->showPWD(filemanager->getNowdir());

cout<<"~$ ";

cin >> command;

if (command == "help")

{

cout << "ls\t\t\t\tList information about the FILEs (the current directory by default)" << endl;

cout << "pwd\t\t\t\tShow absolute path" << endl;

cout << "cd DIRECTORY\t\t\tEnter CHILD-DIRECTORY(use ./ enter Parent DIRECTORY)" << endl;

cout << "mkdir DIRECTORY\t\tCreate the DIRECTORY(ies), if they do not already exist." << endl;

cout << "touch FILE\t\t\tUpdate the access and modification times of each FILE to the current time." << endl;

cout << "rm FILE\t\t\t\tRemove (unlink) the FILE(s)" << endl;

cout << "rmd DIRECTORY\t\t\tRemove directories and their contents recursively" << endl;

cout << "mv SOURCE TARGET\t\tMove SOURCE(s) to DIRECTORY" << endl;

cout << "mvd SOURCEDIRECTORY TARGET\tMove SOURCEDIRECTORY to DIRECTORY" << endl;

cout << "cp SOURCE TARGET\t\tCopy SOURCE to DIRECTORY" << endl;

cout << "cpd SOURCEDIRECTORY TARGET\tCopy SOURCEDIRECTORY to DIRECTORY" << endl;

cout << "vi FILE\t\t\t\tEDIT FILE" << endl;

cout << "cat FILE\t\t\tConcatenate FILE to standard output" << endl;

cout << "rename FILE TARGETNAME\t\tRename file" << endl;

cout << "renamed DIRECTORY TARGETNAME\tRename dictionary" << endl;

cout << "df\t\t\t\tShow information about the file system" << endl;

}

else if (command == "ls")

{

filemanager->showFile();

}

else if (command == "cd")

{

cin >> command;

filemanager->enterDir(command);

}

else if (command == "pwd")

{

filemanager->showPWD(filemanager->getNowdir());

cout << endl;

}

else if (command == "mkdir")

{

cin >> command;

filemanager->createDir(command);

}

else if (command == "touch")

{

cin >> command;

filemanager->createFile(command);

}

else if (command == "rm")

{

cin >> command;

filemanager->deleteFile(filemanager->findFile(command,false));

}

else if (command == "rmd")

{

cin >> command;

filemanager->deleteDir(filemanager->findFile(command, true));

}

else if (command == "mv")

{

cin >> command;

cin >> command1;

filemanager->moveFile(filemanager->findFile(command,false), filemanager->findFile(command1,true));

}

else if (command == "mvd")

{

cin >> command;

cin >> command1;

filemanager->moveDir(filemanager->findFile(command, true), filemanager->findFile(command1, true));

}

else if (command == "cp")

{

cin >> command;

cin >> command1;

filemanager->copyFile(filemanager->findFile(command, false), filemanager->findFile(command1, true));

}

else if (command == "cpd")

{

cin >> command;

cin >> command1;

filemanager->copyDir(filemanager->findFile(command, true), filemanager->findFile(command1, true));

}

else if (command == "vi")

{

cin >> command;

if (filemanager->findFile(command, false)!= NULL)

{

cout << "Enter your string here:";

cin >> command1;

filemanager->editText(filemanager->findFile(command, false), command1);

}

else

{

cout << "Can't find this file!";

}

}

else if (command == "cat")

{

cin >> command;

if (filemanager->findFile(command, false) != NULL)

{

cout << \*filemanager->catText(filemanager->findFile(command, false)) << endl;

}

else

{

cout << "Can't find this file!";

}

}

else if (command == "rename")

{

cin >> command;

cin >> command1;

filemanager->renameFile(command1, filemanager->findFile(command, false));

}

else if (command == "renamed")

{

cin >> command;

cin >> command1;

filemanager->renameFile(command1, filemanager->findFile(command,true));

}

else if (command == "df")

{

filemanager->showDf();

}

else if (command == "exit")

{

break;

}

else

{

cout << "Wrong command!Please try again.Use \"help\" to see all commands."<<endl;

}

}

}

int main() {

FileManager \*filemanager = new FileManager();

StartMenu(filemanager);

return 0;

}

**3.5 部分函数**

**3.5.1 创建文件函数**

创建文件函数需要一个文件名作为参数，默认父目录为当前目录，其他成员信息根据情况自动生成。

knode\* FileManager::createFile(string name)

{

//如果已经存在则不允许生成

if (nowdir->isDuplicate(name, false))

{

cout << "There is already a \"" << name << "\" in parent dictionary!" << endl;

}

else

{

knode \*tmp = new knode(name, false, nowdir);

//父目录子节点列表加入当前knode

nowdir->addChildnode(tmp);

//加入knode表

fileblocks.push\_back(tmp);

usedspace += tmp->getbytesize();

freespace = maxspace - usedspace;

//cout << "Create " << tmp->getfilename() << " completed" << endl;

return tmp;

}

return NULL;

}

**3.5.1 递归删除目录函数**

void FileManager::deleteDir(knode \*node)

{

if (node == NULL)

{

cout << "Can't find this dictionary!" << endl;

return;

}

if (node->isdir == false)

{

cout << "Can't delete file!Please use rm." << endl;

return;

}

//根目录不允许删除

if (node->getNodeid() == 0)

{

cout << "Can't delete root dictionary!" << endl;

return;

}

//当子目录不为空时递归删除所有子目录

for (vector<knode\*>::iterator it = node->getchildlist().begin(); it != node->getchildlist().end()&& !node->getchildlist().empty();++it)

{

if ((\*it)->isdir == true)

{

//为目录时递归调用

deleteDir(\*it);

}

else

{

//为文件时直接删除

deleteFile(\*it);

}

}

//取消与父目录的连接

node->getParent()->removeChild(node);

//将子目录文件信息移除出knode表

for (vector<knode\*>::iterator fbit = fileblocks.begin(); fbit != fileblocks.end();)

{

if (\*fbit == node)

{

fileblocks.erase(fbit);

cout << "Delete " << node->getfilename() << " completed" << endl;

break;

}

else

{

++fbit;

}

}

delete node;

return;

}

1. **实验结果**

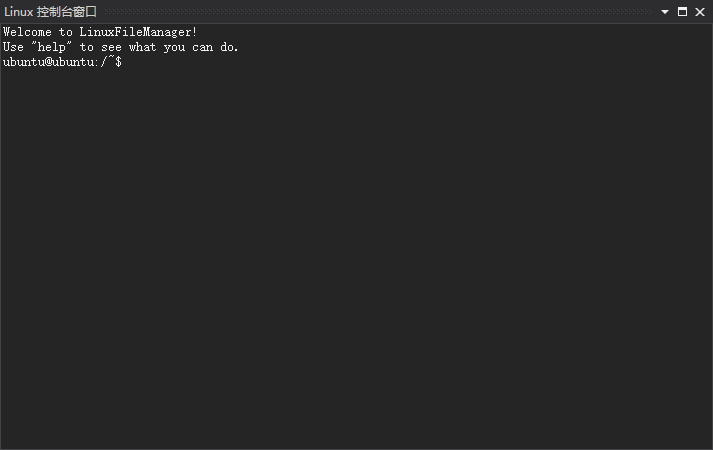


图4.1 开始界面

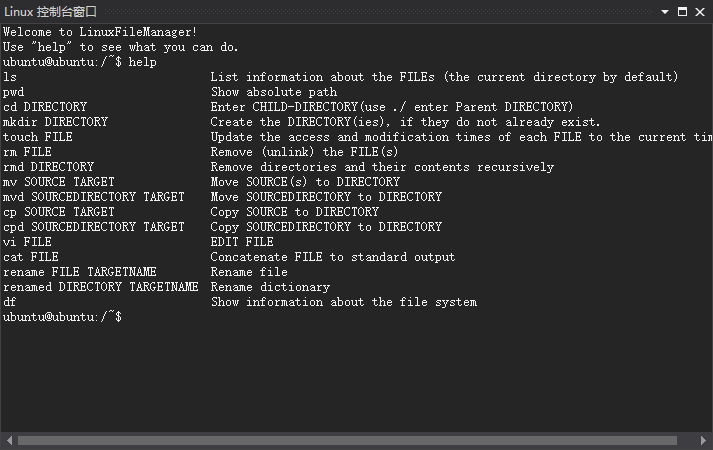


图4.2 help命令

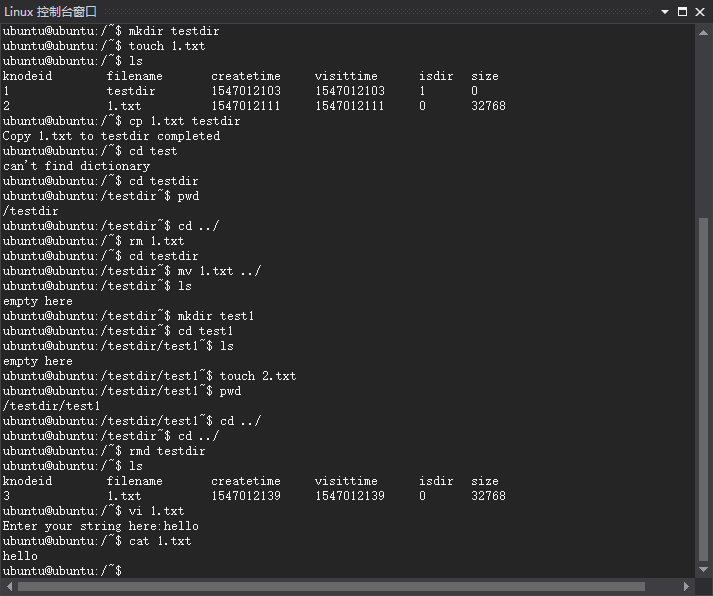


图4.3 各种命令组合

1. **实验总结**

**5.1对专业知识基本概念、基本理论和典型方法的理解。**

首先要理解文件系统，linux的文件系统与windows的不同，linux中所有数据都是文件，每个文件都有一个inode号，保存在一张inode表中，大大减少了文件系统的复杂度。所有的文件操作只需对inode表做出修改即可。

第二点，linux是纯命令系统，所有操作通过控制台输入命令后完成，所以用户可能输入不存在的文件名，需要对用户命令进行错误分析。

第三点，linux使用block存放数据，一个文件占用一个或多个block，由superblock记录当前磁盘大小，块数及使用情况。inode存放文件block占用信息。

**5.2怎么建立模型。**

本次模型建立仿照了linux文件系统，在linux文件系统中至少有inode类，superblock类，dentry类，总管理类。本次实验由于时间和精力有限，将inode类，dentry类 合并为一个knode类，将superblock类和管理类进行了精简合并。knode类存放单个文件的基本信息，FileManager用于管理每个文件。

**5.3如何利用基本原理解决复杂工程问题。**

在实验中有一个通过文件名查找knode信息的函数，所以要考虑到搜索的简易性和高效性，想了很多数据结构的设计方案，比如哈希表、数组、双向链表、动态数组等，哈希表在面对大量数据时效果不好，数组无法动态调整大小，链表编程复杂，动态数组是个不错的选择。最后决定使用双向链表加上动态数组的数据结构，节点之间通过双向链表连接，knode表通过动态数组存放，只需使用vector自带的搜索函数或自己使用迭代器搜索就解决了该问题。

**5.4具有实验方案设计的能力。**

本次方案从零开始完全自主设计，一开始就想好了knode类，然后就开始写了，写完之后发现有很多问题，包括文件管理如何实现等，之后便开始写FileManager类，同时管理类需要很多接口knode类没有提供，就边写管理类，遇到没提供的就再补充。在编程的过程中每写好一项功能就会进行一次简单测试，这为之后排除bug提供了巨大帮助，我觉得这是一个良好得习惯。

**5.5如何对环境和社会的可持续发展。**

对于后续得扩展性，我在开始编写得时候就考虑到了，文件管理和文件信息类相互独立，所以得内部私有成员都设置得读写接口，严格区分了文件和目录的操作函数，防止混淆。增加本类功能几乎不需要改动其他类，实现了独立性。