

**课程设计报告**

**课程名称： 操作系统课程设计**

**专业班级： 物联网1601**

**学 号： U201614887**

**姓 名： 林子涵**

**指导教师： 石柯**

**报告日期： 2018年9月8日**

**计算机科学与技术学院**

**目录**

[设计内容一 熟练掌握Linux编程环境 1](#_Toc524535609)

[**1.1目的** 1](#_Toc524535610)

[**1.2** **设计内容** 1](#_Toc524535611)

[**1.3环境及步骤** 1](#_Toc524535612)

[**1.4关键代码** 2](#_Toc524535613)

[**1.5调试记录及运行结果** 3](#_Toc524535614)

[设计内容二 增加系统调用 5](#_Toc524535615)

[**2.1目的** 5](#_Toc524535616)

[**2.2 设计内容** 5](#_Toc524535617)

[**2.3环境及步骤** 5](#_Toc524535618)

[**2.4关键代码** 6](#_Toc524535619)

[设计内容三 增加设备驱动程序 11](#_Toc524535620)

[**3.1目的** 11](#_Toc524535621)

[**3.2 设计内容** 11](#_Toc524535622)

[**3.3环境及步骤** 11](#_Toc524535623)

[**3.4关键代码** 13](#_Toc524535624)

[**3.5调试记录及运行结果** 14](#_Toc524535625)

[设计内容四 系统监控器 15](#_Toc524535626)

[**4.1目的** 15](#_Toc524535627)

[**4.2 设计内容** 15](#_Toc524535628)

[**4.3环境及步骤** 15](#_Toc524535631)

[**4.4关键代码** 17](#_Toc524535662)

[设计内容五 模拟文件系统 21](#_Toc524535663)

[**5.1目的** 21](#_Toc524535664)

[**5.2 设计内容** 21](#_Toc524535665)

[**5.3环境及步骤** 21](#_Toc524535666)

[**5.4关键代码** 22](#_Toc524535667)

[**5.5调试记录及运行结果** 24](#_Toc524535668)

[实验感想 29](#_Toc524535669)

[附录 30](#_Toc524535670)

# 设计内容一 熟练掌握Linux编程环境

## **1.1目的**

1）掌握Linux操作系统的使用方法；

2）了解Linux系统内核代码结构；

3）掌握实例操作系统的实现方法。

## **设计内容**

1）编写一个C程序，用read、write等系统调用实现文件拷贝功能。命令形式： copy <源文件名> <目标文件名>

2）编写一个C程序，使用基于文本的终端图形编程库(curses)或图形界面(QT/GTK），分窗口显示三个并发进程的运行(一个窗口实时显示当前时间，一个窗口实时监测CPU的利用率，一个窗口做1到100的累加求和，刷新周期均为1秒)。

## **1.3环境及步骤**

环境：Ubuntu16.04，内核版本：4.15.0

Qt版本：5.10.0

步骤：

对于第一个部分，没有了并行的要求，实际上就是最简单的文件读取操作，利用open系统调用打开作为命令行参数的文件，用read、write系统调用来实现逐字节的拷贝，利用循环每次从源文件中读出几个字节再拷贝如目标文件，以此实现最简单的文件拷贝。

对于第二个部分，这里需要用到课程实验时的fork( )程序以及QT的图形显示。

基本fork（）程序，实现三个并发进程的结构。主程序结构清晰，调用两次fork（）函数，在if else结构中判断当前属于哪个进程，分别启动三个相应的窗口即可。三个窗口的逻辑编写类似，都是在主窗口中添加一个label控件，每次更新时刷新label控件中显示的内容，这里记录时间的控件是QT自带的QTimer，将其设置为每一秒触发一次，利用QT的信号与槽机制，每次触发时调用更新函数，核心代码如下：connect(timer,SIGNAL(timeout()),this,SLOT(action()));

主程序逻辑框图如下：

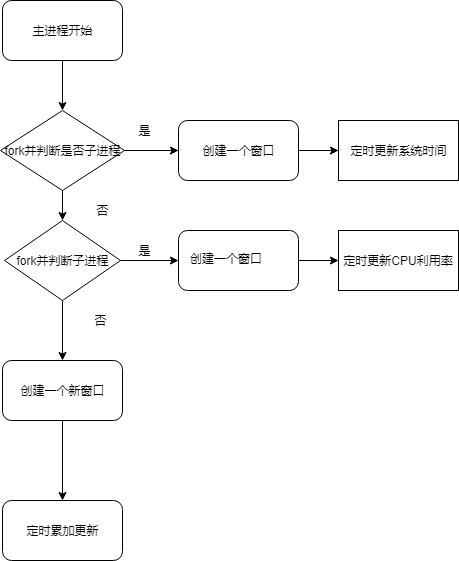


图1-1 多进程窗口现实流程图

其余代码见下面。

## **1.4关键代码**

第一部分：

int readfile = open(argv[1],O\_RDONLY,S\_IRUSR|S\_IWUSR);

int writefile = open(argv[2],O\_CREAT|O\_RDWR,S\_IRUSR|S\_IWUSR);

while(read(readfile,&get,sizeof(char))!=0){

if(write(writefile,&get,1)==-1){

printf("%c\n",get);

}

}

第二部分：

int p1,p2;

p1=fork();

if(p1==0){

QApplication a(argc, argv);

MainWindow1 w;

w.move(200,200);

w.show();

return a.exec();

}

else{

p2=fork();

if(p2==0){

QApplication b(argc, argv);

MainWindow2 x;

x.move(700,200);

x.show();

return b.exec();

}

else{

QApplication c(argc, argv);

MainWindow3 y;

y.move(500,400);

y.show();

return c.exec();

}

}

## **1.5调试记录及运行结果**

调试记录：

对于第一个部分我是使用实验代码中的并行文件复制改的，需要做的就是删除其中关于共享内存的部分以及fork创建进程的过程，因为程序过于简单，基本直接运行就可以通过。

第二个部分中fork创建进程部分是常见的，也是实验中出现过的，最关键的是qt的UI部分，因为对qt不是很熟，所以我查阅了相关资料，知道了要创建窗口首先要实例化QApplication然后再实例化自己写的窗口类，将这两步放到fork之后的ifelse语句中即可。下面的关键是如何实现更新了。因为要求1秒钟刷新一次，所以肯定要有一个timer，这里我用的是qt里面的控件QTimer，其可以与其他控件的信号-槽机制完美配合，至于信号-槽机制，其是用于对事件的响应的具体用法如下：connect(timer,SIGNAL(timeout()),this,SLOT(action()));，这样可以将两个函数对应的事件绑定起来，依次来实现定时更新。每个窗口内部只是一个简单的Text控件，来现实一个字符串，这个在查阅用法之后实现起来没有难度。

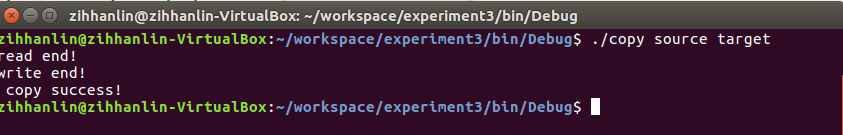


图1-2 文件拷贝运行结果

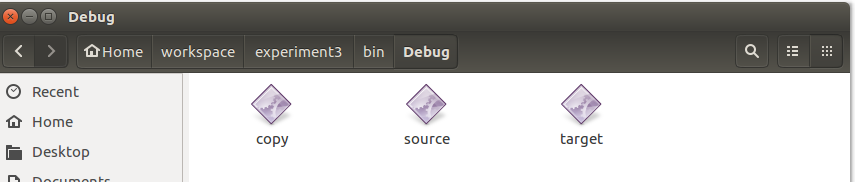


图1-3 文件拷贝运行结果（2）

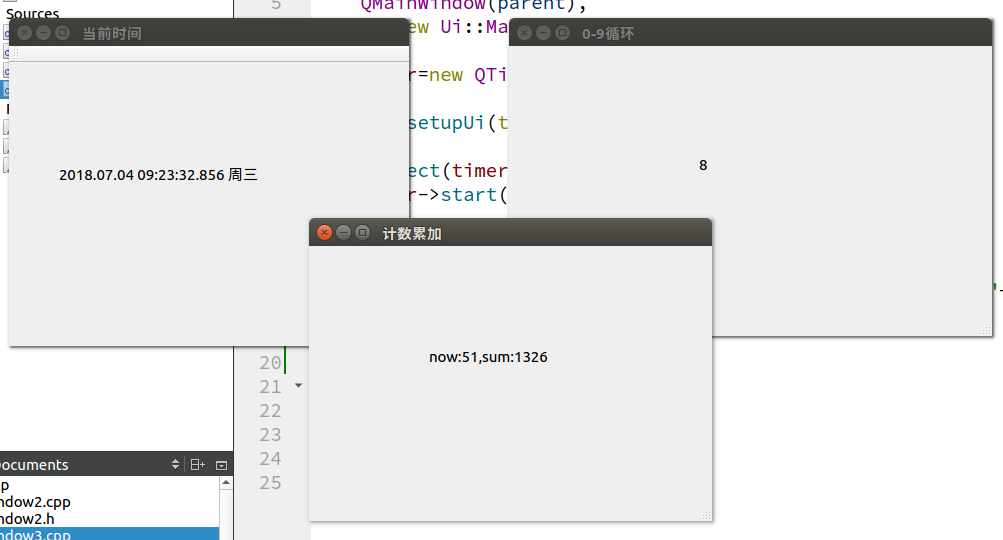


图1-4 三窗口运行结果

# 设计内容二 增加系统调用

## **2.1目的**

了解Linux系统内核代码结构；

## **2.2 设计内容**

1. 采用编译内核的方法，添加一个新的系统调用实现文件拷贝功能

2）编写一个应用程序，测试新加的系统调用

## **2.3环境及步骤**

环境：Ubuntu16.04 老内核：4.15.0新内核：4.9.11

步骤：

1. 前往**https://www.kernel.org/pub/linux/kernel/** 下载相应的内核版本，这里我下载的是linux-4.9.11，下载下来后找一个剩余空间大的磁盘，用CP命令，将压缩包拷贝过去。这里我下载的压缩包格式为：linux-4.9.11.tar.xz所以相应的解压命令为：xz -d linux4.9.11.tar.xz和tar xvf linux-4.9.11.tar。下面进入添加系统调用环节。
2. 在解压好的文件夹目录下进入命令行，使用命令 sudo gedit linux-4.9.11/kernel/sys.c打开sys.c文件，将新的系统调用函数复制到文件的末位。
3. 使用命令sudo gedit arch/x86/entry/syscalls/syscall\_64.tbl打开syscall\_64.tbl添加系统调用号，如下图所示：
4. 使用命令sudo gedit include/asm-generic/syscalls.h打开syscalls.h添加系统调用函数声明。
5. 安装ncurses库

sudo apt-get install libncurses5-dev

1. 现在开始编译内核文件，

首先生成内核配置文件：make mrnuconfig

弹出一个对话框，直接退出即可。

1. 完成内核配置映像：make bzImage
2. 编译内核模块：make modules
3. 安装并生成内核模块：make modules\_install
4. 安装新的系统 ：make install
5. 重启系统，选择新内核进入 uname -a命令查看当前版本。
6. 编写c语言程序，测试系统调用。

## **2.4关键代码**

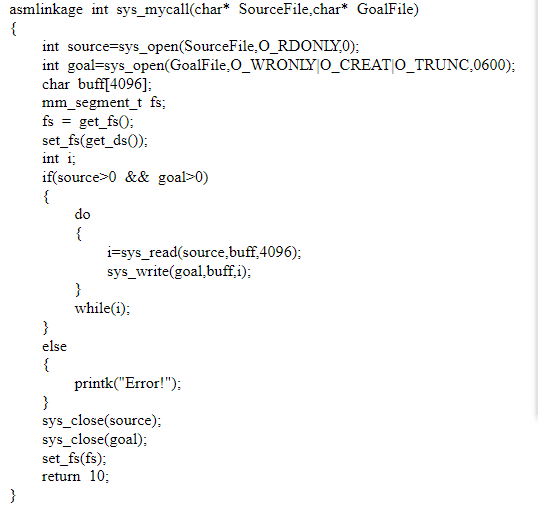


图2.1系统调用代码



图2.2 系统调用号



图2.3 系统调用声明

图片包含 屏幕截图

已生成高可信度的说明

图2.4 测试代码

**2.5调试记录及运行结果**

这个实验基本上没有设计部分吧，只需要按照流程走一遍掌握每一步的意义即可，其中编写系统调用的代码也是可以用实验一中的文件拷贝代码改动过来的，需要做的就是设置内核区域内存保护（ppt中有）以及修改系统调用为内核内部可认的sys型调用，然后编译安装过程都按照流程进行即可，这里如果想加快速度的话可以用 -j4来进行4核并行编译，每一步的执行结果见下面。

make mrnuconfig 结果如图：

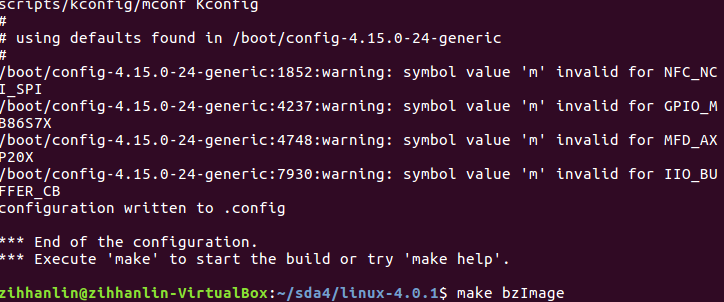


图2.5 编译结果

make bzImage 结果如下图：

图片包含 文字

已生成极高可信度的说明

图2.6 编译结果

make modules 结果如图：

图片包含 文字

已生成高可信度的说明

图2.7 编译结果

make modules\_install 结果如图：

图片包含 文字, 瓶子

已生成极高可信度的说明

图2.8 编译结果

make install 结果如图：

图片包含 屏幕截图, 文字

已生成高可信度的说明

图2.9 编译结果

uname -a命令查看当前版本如图：

图片包含 屏幕截图, 监视器, 室内, 屏幕

已生成高可信度的说明

图2.10系统版本号

测试程序运行结果如下：

图片包含 屏幕截图

已生成极高可信度的说明

图2.11测试结果

# 设计内容三 增加设备驱动程序

## **3.1目的**

了解Linux系统内核代码结构；

掌握模块化添加驱动程序方法

## **3.2 设计内容**

1. 采用模块方法，添加一个新的字符设备的驱动程序，实现打开/关闭、读/写等基本操作
2. 编写一个应用程序，测试添加的驱动程序

## **3.3环境及步骤**

环境：Ubuntu16.04 内核：4.15.0

Linux内核中的设备驱动程序是一组常驻内存的具有特权的共享库，是低级硬件处理例程。对用户程序而言，设备驱动程序隐藏了设备的具体细节，对各种不同设备提供了一致的接口，一般来说是把设备映射为一个特殊的设备文件，用户程序可以象对其它文件一样对此设备文件进行操作。

Linux支持3种设备：字符设备、块设备和网络设备。

设备由一个主设备号和一个次设备号标识。主设备号唯一标识了设备类型，即设备驱动程序类型，它是块设备表或字符设备表中设备表项的索引。次设备号仅由设备驱动程序解释,一般用于识别在若干可能的硬件设备中，I/O请求所涉及到的那个设备。

一个典型的驱动程序,大体上可以分为这么几个部分:

①注册设备：

在系统初启,或者模块加载时候,必须将设备登记到相应的设备数组,并返回设备的主设备号；

②定义功能函数：

对于每一个驱动函数来说，都有一些和此设备密切相关的功能函数。以最常用的块设备或者字符设备来说，都存在着诸如 open()、read()这一类的操作。当系统调用这些调用时，将自动的使用驱动函数中特定的模块。来实现具体的操作；

③卸载设备：

在不用这个设备时,可以将它卸载，主要是从/proc 中取消这个设备的特殊文件。

实验过程：

1. 编写驱动程序：

编写驱动程序的.c文件和Makefile文件，具体内容见附录。

1. 编译驱动程序

在驱动文件的目录下打开命令行，执行make命令，结果如图：

图片包含 瓶子, 文字

已生成高可信度的说明

图3.1编译驱动程序结果

1. 装载模块

insmod myDev.ko

1. 分配次设备号

每个文件都有两个设备号，第一个是主设备号，标识驱动程序，第二个是从设备号，标识使用同一个设备驱动程序的不同的硬件设备，比如有两个软盘，就可以用从设备号来区分他们.设备文件的的主设备号必须与设备驱动程序在登记时申请的主设备号一致，否则用户进程将无法访问到驱动程序.。

分配前必须要知道主设备号

cat  /proc/devices

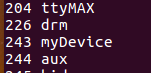


图2.16 查看设备号

分配从设备号

mknod /dev/myDevice c 243 0

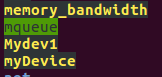


图2.17 分配从设备号

1. 测试驱动程序

编写测试程序，测试open、read和write函数是否可用。

## **3.4关键代码**

Makefile:

ifneq ($(KERNELRELEASE),)

#kbuild syntax.

mymodule-objs :=myDev.o //模块的文件组成

obj-m :=myDev.o //生成的模块文件名

else

PWD :=$(shell pwd)

KVER :=$(shell uname -r)

KDIR :=/lib/modules/$(KVER)/build

all:

$(MAKE) -C $(KDIR) M=$(PWD)

clean:

rm -f \*.cmd \*.o \*.mod \*.ko

endif

Test.c:

testdev = open("/dev/myDev1",O\_RDWR|O\_NONBLOCK);

if(testdec!=-1){

printf("Pleasee input a string:\n");

gets(get);

write(testdev,get,sizeof(get));

read(testdev,buf,sizeof(buf));

printf("\n The string of device is %s\n",buf);

close(testdev);

# 3.5调试记录及运行结果

这个部分同上面编译内核一样，有每一步的详细指导，唯一需要调试的应该就是编写驱动程序模块的C语言文件了，这个涉及到一些linux模块定义的标准，所以我上网查阅了一些资料和样例代码，说实话这里对于其具体的标准写法不是很清楚，只是照猫画虎的将字符串读写的功能加到驱动里面，最后经过调试安装还是成功完成了实验的功能。

测试open、read和write函数是否可用。结果如图：

图片包含 文字

已生成高可信度的说明

图2.18 测试结果

# 设计内容四 系统监控器

## **4.1目的**

了解/proc文件的组织结构

掌握对部分/proc文件中的内容的读取

将数据用图形化的方式展示出来

## **4.2 设计内容**

1）了解/proc文件的特点和使用方法

2）监控系统状态，显示系统部件的使用情况

3）用图形界面监控系统状态，包括CPU和内存利用率、所有进程信息等(可自己补充、添加其他功能)

## **4.3环境及步骤**

关于系统和进程的所有信息均在/proc文件系统下，我们只需按照文本文件的方式，读取相关的文件，即可实时获取当前信息，然后利用QT前端界面将信息显示出来，并定时刷新即可。

在本程序中用到的文件及文件内容格式如下：

**/proc/stat：**

在此文件中，我们只用到其第一行的内容，包含一个字符串和7个数字，具体含义如下定义：

user 从系统启动开始累计到当前时刻，用户态的CPU时间，   
nice 从系统启动开始累计到当前时刻nice值为负的进程所占用的CPU时间  
system 从系统启动开始累计到当前时刻，核心时间  
idle 从系统启动开始累计到当前时刻，除硬盘IO等待时间以外其它等待时间iowait 从系统启动开始累计到当前时刻，硬盘IO等待时间，  
irq 从系统启动开始累计到当前时刻，硬中断时间  
softirq 从系统启动开始累计到当前时刻，软中断时间

**/proc/meminfo：**

在此文件中，我们只用部分信息来计算内存利用率，用到4个

Total 所有可用的 RAM 大小，物理内存减去预留位和内核使用

Free 所有空间中的可用空间

Buffer 用来给块设备做缓存的大小（文件系统的 metadata, tracking in-flight pages）

Cached 文件的缓冲区大小

**/proc/$pid/stat:**

此文件内容相当复杂，这里我们仅使用其中的5个字符串信息，包括；

Name 进程名

Pid 进程编号

Ppid 父进程偏好

Rss 内存占用量

Priority 进程优先级

**/proc/sys/kernel/hostname：**

此文件仅包含一行内容，即系统的主机名，直接按照字符串进行读取即可。

**/proc/uptime：**

此文件包含一个浮点型数字，即系统自开始运行到现在的秒数。

**/proc/sys/kernel/ostype：**

此文件包含一个字符串，即系统的版本名。

**/proc/sys/kernel/osrelease：**

此文件包含一个字符串，即系统的版本号。

**/proc/cpuinfo：**

此文件内容同样复杂，在这里我们取其中的CPU型号这个字符串。

以上文件内容具体怎么取出请参考完整代码。

QT界面显示部分，实现细节如下：

程序的主界面是一个窗口，这里我们直接使用QMainWindow这个类，在窗口中一个主要的控件就是tabweight，其包含3个tab项分别显示需要的三个界面，即系统信息，进程信息和图表。在其下方是三个textBrowser，分别作为固定栏显示当前时间，CPU利用率和内存利用率。在第一个界面中，大部分的信息都是固定，这类信息我们只需要在开始时读一次文件显示即可，而随时间变化的内容，如当前系统运行时间，我们需要设置一个定时器，每一秒触发一次，同时设置相应的槽函数来更新显示内容。在第二个界面是进程信息，这里我用一个tabelWeight来显示所有信息，同时每5秒刷新一次，当点击结束进程按钮是，对应的进程会被系统命令kill直接杀死，同时更新显示。在第三个界面中是两个图表，分别显示CPU使用率变化曲线和内存使用率变化曲线，这里我使用的是一个QPixMap来作为画板，在上面进行绘制，信息由每秒读取的CPU和内存信息构成，将其保存在一个list中，每秒刷新重绘。

整体模块框架图如下：

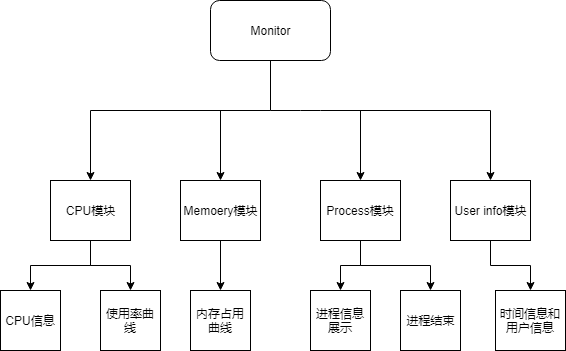


图4.1 模块结构图

## **4.4关键代码**

void MainWindow::UpdateCpuLine(){

int count=0;

QPixmap pix(600,160);

QPainter painter(&pix);

pix.fill(Qt::blue);

QPen pen0;

pen0.setColor(Qt::lightGray);

painter.setPen(pen0);

for(int i=1;i<4;i++) painter.drawLine(0,i\*40,600,i\*40);

QPen pen;

pen.setColor(Qt::red);

pen.setWidth(2);

painter.setPen(pen);

while(count<(cpulist.size()-1)){

painter.drawLine(5\*count,160-1.6\*(cpulist.value(count)),5\*count+5,160-1.6\*(cpulist.value(count+1)));

count++;

}

ui->cpulinelabel->setPixmap(pix);

}

void MainWindow::ProcessInfo(){

ui->tableWidget->setColumnCount(6);

QStringList headers;

headers<<"杩涚▼鍚?<<"pid"<<"ppid"<<"鍐呭瓨鍗犵敤/KB"<<"浼樺厛绾?<<"鎿嶄綔";

ui->tableWidget->setHorizontalHeaderLabels(headers);

ui->tableWidget->horizontalHeader()->setStretchLastSection(true);

int proNum=GetProcessInfo();

ui->tableWidget->setRowCount(proNum);

for(int i=0;i<proNum;i++){

QTableWidgetItem \*nameItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].name));

ui->tableWidget->setItem(i,0,nameItem);

nameItem->setTextAlignment(Qt::AlignCenter);

QTableWidgetItem \*pidItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].pid));

ui->tableWidget->setItem(i,1,pidItem);

pidItem->setTextAlignment(Qt::AlignCenter);

QTableWidgetItem \*ppidItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].ppid));

ui->tableWidget->setItem(i,2,ppidItem);

ppidItem->setTextAlignment(Qt::AlignCenter);

QTableWidgetItem \*rssItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].rss));

ui->tableWidget->setItem(i,3,rssItem);

rssItem->setTextAlignment(Qt::AlignCenter);

QTableWidgetItem \*priorityItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].priority));

ui->tableWidget->setItem(i,4,priorityItem);

priorityItem->setTextAlignment(Qt::AlignCenter);

QPushButton \*btn = new QPushButton;

btn->setText("缁撴潫杩涚▼");

btn->setStyleSheet("text-align:center;");

ui->tableWidget->setCellWidget(i,5,btn);

connect(btn,SIGNAL(clicked()),this,SLOT(RemovePro()));

}

}

**4.5调试记录及运行结果**

这个实验的重点还是在数据显示上，数据获取就是简单的读/proc下的各文本文件，前端显示的一些细节选择在上一小节中已经讲述过了，这里的调试过程主要为拖拽各控件，使其处于合适位置合适大小，测试各个控件对于事件的相应是否正确，具体细节不在赘述，这里说一点，就是QTableWeight控件中可以设置当前表的行数，当行不删除时进行内容的更新是不会产生闪烁或跳跃的，这一点qt已经帮我们实现好。

图片包含 屏幕截图

已生成极高可信度的说明

图4.2 系统监控器界面1

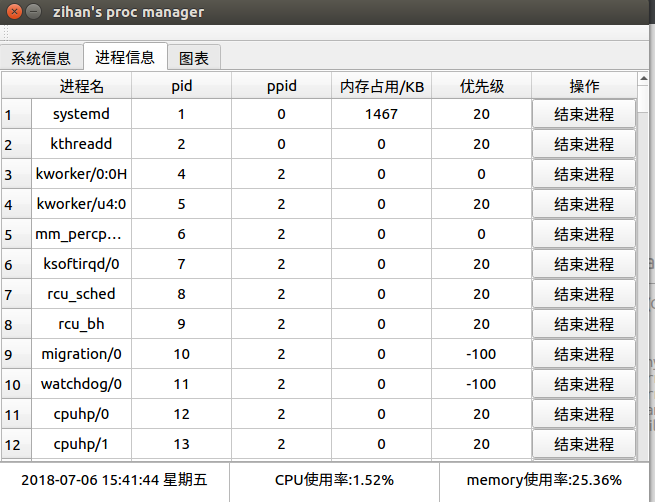


图4.3 系统监控器界面2

图片包含 屏幕截图

已生成极高可信度的说明

图4.4 系统监控器界面3

# 设计内容五 模拟文件系统

## **5.1目的**

了解Linux文件系统结构

掌握简单文件系统的实现方法

## **5.2 设计内容**

1. 设计、实现一个模拟的文件系统

方法：申请一个100MB的文件，模拟文件系统的存储空间就是该文件的存储空间，利用该文件的I结点的信息建立模拟文件系统的存储介质管理机制（表、链均可）

1. 包含文件/目录创建/删除，目录显示等基本功能(可自行扩充文件读/写、用户登录、权限控制、读写保护等其他功能)

## **5.3环境及步骤**

环境：代码在Windows 10 系统上Dev c++ 编写，迁移至Ubuntu 16.04上可正常运行。

步骤：

首先，文件系统最关键的就是文件之间的链接结构了，这里我选择的是最简单的一种基于bitmap的连续结构，先将内存中相应区域划分为大小相等的若干block，作为基本的块结构来存储文件数据，创建inode结构来代表文件或目录，每一个inode指向一个文件或一个目录，同时若指向目录则存储其下的子目录信息，如指向文件，则存储其相应的block位置，每个inode连续存放，其位置以及使用情况又对应的bitmap来表示。通过限制子目录个数和文件大小来限制inode中存储的数据项，其结构表示图如下：

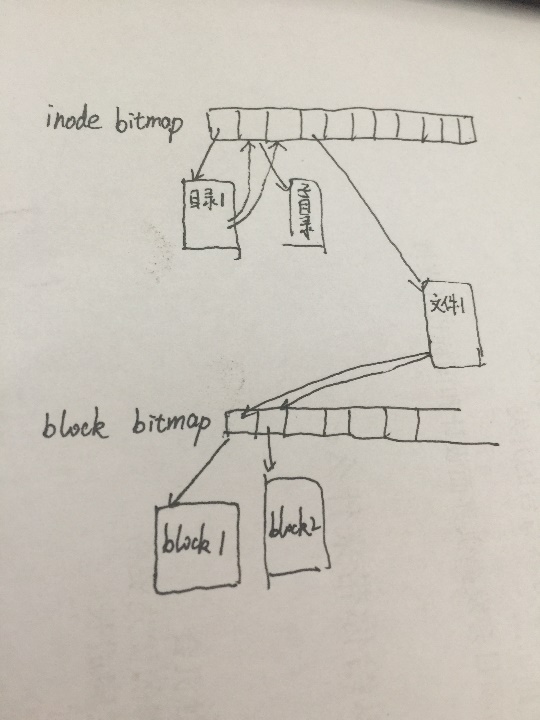


图5.1 文件系统基础结构

## **5.4关键代码**

I节点定义：

typedef struct Inode{

int isFileOrNot; //判断是文件还是目录,4byte;0为目录,1为文件

char name[26]; //文件或目录名，26byte

int block\_num; //所占用的磁盘块数，4byte

int block\_pos[7]; //占用的磁盘块地址，7\*4=28byte

int index; //在inode区中的编号，4byte

int parent; //父索引节点，4byte

int child\_num; //子节点个数，4byte

int child[Child\_Maxnum]; //子索引节点，6\*4=24byte

char ctime[26]; //文件创建时间,26byte

int role //文件创建者的角色

} inode;

创建文件部分：

case 3:{//create file

int i;

if(super\_block.free\_inode == 0 && super\_block.free\_block == 0){

printf("Not enough storage space!\n");

break;

}

if(iNodeList[curr].child\_num >= Child\_Maxnum){

printf("Subfiles is full!\n");

break;

}

for(i = 0; i < Inode\_Num; i++){

if(!inode\_bitmap[i])

{

char file\_name[28], text[Text\_Maxsize];

printf("Please input the name of file:");

scanf("%s", &file\_name);

getchar();

if(create\_inode(&file\_name, &iNodeList, i)){

iNodeList[i].isFileOrNot = 1;

iNodeList[i].index = i;

iNodeList[i].role = admin;

iNodeList[curr].child[iNodeList[curr].child\_num] = i;

iNodeList[curr].child\_num++;

super\_block.free\_inode--;

inode\_bitmap[i] = 1;

}

else{

break;

}

printf("Please input text: (End by $)\n");

char c=0;int cnt=0;

while(c!='$'){

scanf("%c",&c);

if(c=='$') break;

text[cnt++]=c;

}

text[cnt]='\0';

getchar();

printf("\n");

if(input(text, &blockList, &iNodeList,i))

printf("Creating Success!\n");

else

printf("Failed to create!\n");

break;

}

}

break;

}

## **5.5调试记录及运行结果**

调试记录：

主要调试部分包括两个方面：

1. 创建文件后相应文件内容是否能存入block中。

调试这一部分其实就类似于写一个自己的管理系统，因为不涉及到硬盘的读写，只使用内存来模拟，所以只要调试上面的功能函数是否可以完成键盘内容的识别，是否能正常结束，是否能原样输出，是否能修改，这几项都类似于C语言的管理系统，并不复杂，只要相应数据结构把握好，调试过程都很顺利，这里不再赘述。

1. 权限控制，普通用户对于管理员权限文件的读写限制。

因为要对于每个目录和文件的权限控制，就需要另外保存每个i节点创建时的用户权限，然后在相应读写操作时先进行权限判断，如果权限不足的话就拒绝操作，这个就相当于在整个功能层之上添加一个权限控制层，具体调试过程就是添加几个判断函数，详情可以参考附录中代码部分。

运行结果：

用户权限选择界面如下图：

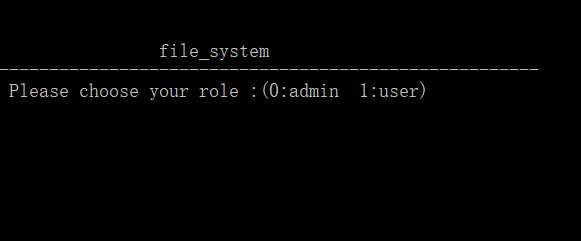


图5.2 角色选择

文本菜单主界面如下图：

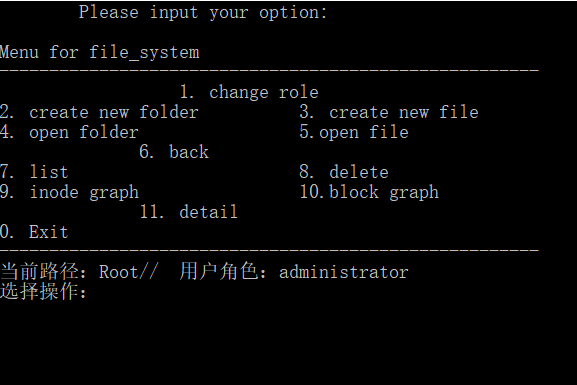


图5.3 主界面

创建目录结果如下图：

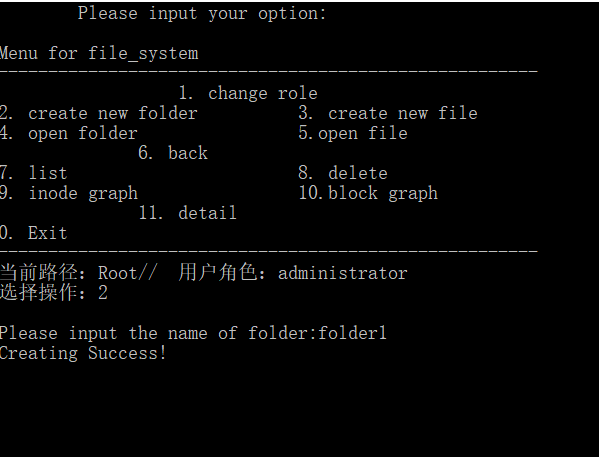


图5.4 创建目录

创建文件并写入字符串结果如下：

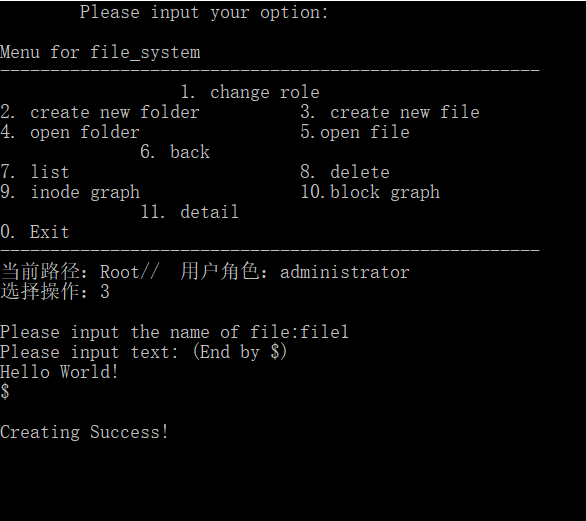


图5.5 创建文件

进入一层目录结果如下图：

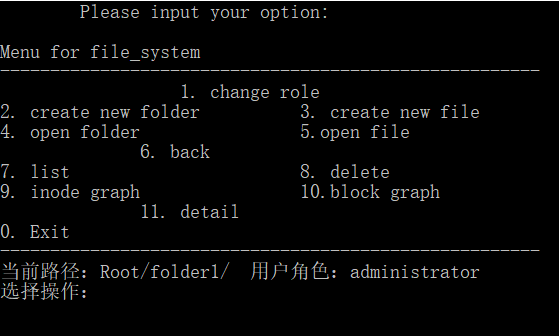


图5.6 进入子目录

查看文件内容，并修改如下图：

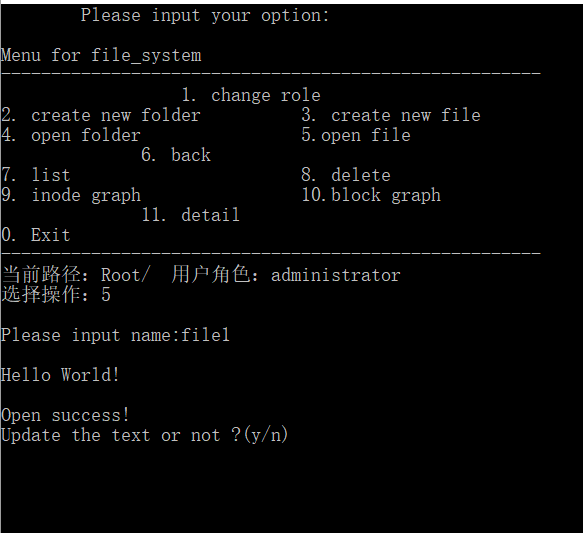


图5.7 查看文件

展示当前路径下所有目录和文件如下图：

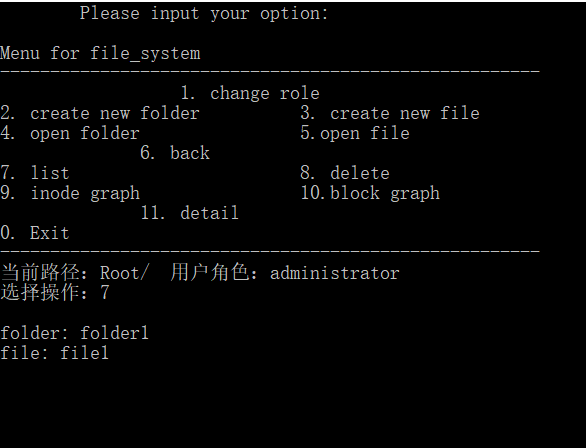


图5.8 类ls功能

展示一个文件的详细信息如下：

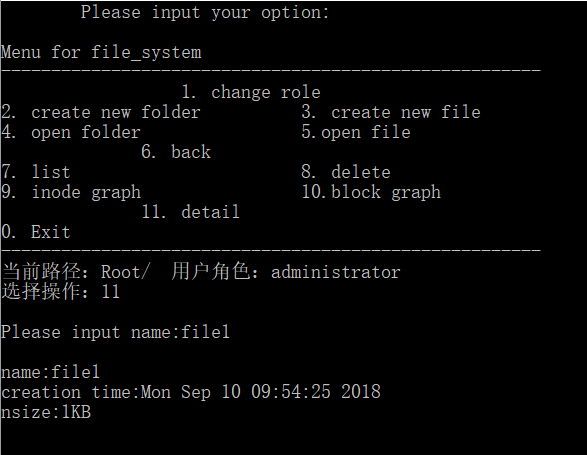


图5.9 文件信息

Inode位图存储结果展示如图：

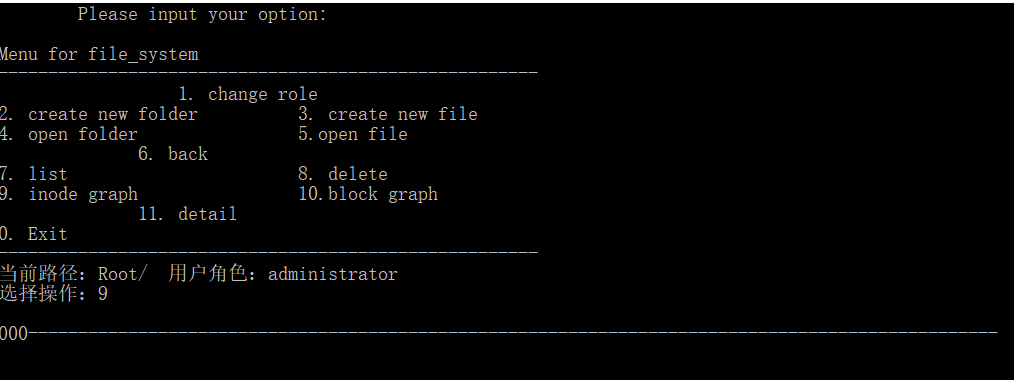


图5.10 inode存储

# 实验感想

这次的课程设计相比较与前面C语言和数据结构的可以说更加全面也更加费工夫了，暑假刚开始我就开始了课程设计的功能，第一个部分还算顺利，虽然之前没有接触过qt，但是网上查找相关资料后也做出了一个qt界面，虽然还不太美观，有很多可以改进的地方，但是基本的功能可以实现了。然后编译内核部分是最费力气的了，前后我花了一周多的时间吧，因为开了多核编译，速度还能接受，但是在切换内核后一直运行不起来，这个问题困扰了我很久，查阅了很多都没有发现类似的问题，包括手动改引导，改grub，最后都没有效果，最后通过询问老师，发现可能是内核版本差距过大导致的，具体的细节也无从考究了，这说明了linux系统灵活度很高，哪怕一个细节出问题都可能导致严重的后果，同时也给使用和开发人员带来了更多的可选择性，对于Linux的学习还有很多要做的。其余的几个部分在完成时没有遇到太大的问题，通过上网查找和问同学都可以解决，文件系统的整体结构比较简单，支持的功能也有限，但是让我体会到了linux文件系统的基本实现方式。

总体来说，这次课程设计收获很多，题目的设置也很合理。通过完成这些题目我进一步掌握了Linux系统的组织结构以及基于Linux编程的方法，也感谢问为我提供帮助的老师、各位同学以及博主，祝老师工作顺利！

# 附录

实验1，文件拷贝代码：

#include <stdio.h>

#include <stdlib.h>

#include<unistd.h>

#include<pthread.h>

#include<sys/sem.h>

#include<sys/shm.h>

#include<sys/stat.h>

#include <signal.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/types.h>

#include<fcntl.h>

#include <sys/wait.h>

char \*s;

int main(int argc,char \*argv[])

{

if(argc!=3){

printf("argc number fault!\n");

exit(0);

}

int readfile = open(argv[1],O\_RDONLY,S\_IRUSR|S\_IWUSR);

int writefile = open(argv[2],O\_CREAT|O\_RDWR,S\_IRUSR|S\_IWUSR);

if(readfile==-1||writefile==-1){

printf("file name error!\n");

exit(0);

}

char get;

while(read(readfile,&get,sizeof(char))!=0){

if(write(writefile,&get,1)==-1){

printf("%c\n",get);

}

}

printf("copy success!\n");

close(writefile);

close(readfile);

return 0;

}

分窗口显示并发进程信息：

main.cpp:

#include "mainwindow1.h"

#include "mainwindow2.h"

#include"mainwindow3.h"

#include <QApplication>

#include <sys/types.h>

#include <unistd.h>

int main(int argc, char \*argv[])

{

int p1,p2;

p1=fork();

if(p1==0){

QApplication a(argc, argv);

MainWindow1 w;

w.move(200,200);

w.show();

return a.exec();

}

else{

p2=fork();

if(p2==0){

QApplication b(argc, argv);

MainWindow2 x;

x.move(700,200);

x.show();

return b.exec();

}

else{

QApplication c(argc, argv);

MainWindow3 y;

y.move(500,400);

y.show();

return c.exec();

}

}

}

Mainwindow1.cpp:

#include "mainwindow1.h"

#include "ui\_mainwindow1.h"

#include<QTimer>

#include<QDateTime>

MainWindow1::MainWindow1(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow1)

{

timer=new QTimer();

ui->setupUi(this);

connect(timer,SIGNAL(timeout()),this,SLOT(action()));

timer->start(1000);

}

void MainWindow1::action(){

QDateTime current\_date\_time =QDateTime::currentDateTime();

QString current\_date =current\_date\_time.toString("yyyy.MM.dd hh:mm:ss.zzz ddd");

ui->label->setText(current\_date);

}

MainWindow1::~MainWindow1()

{

delete ui;

}

mainwindow2.cpp:

#include "mainwindow2.h"

#include "ui\_mainwindow2.h"

#include<QTimer>

MainWindow2::MainWindow2(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow2)

{

timer=new QTimer();

ui->setupUi(this);

connect(timer,SIGNAL(timeout()),this,SLOT(action()));

timer->start(1000);

}

void MainWindow2::action(){

ui->label->setText(QString::number(cnt%10, 10));

cnt++;

}

MainWindow2::~MainWindow2()

{

delete ui;

}

mainwindow3.cpp:

#include "mainwindow3.h"

#include "ui\_mainwindow3.h"

#include<QTimer>

MainWindow3::MainWindow3(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow3)

{

timer=new QTimer();

i=1;

ui->setupUi(this);

connect(timer,SIGNAL(timeout()),this,SLOT(action()));

timer->start(1000);

}

void MainWindow3::action(){

sum+=i;

ui->label->setText("now:"+QString::number(i, 10)+",sum:"+QString::number(sum, 10));

i++;

if(i==1001) i=0;

}

MainWindow3::~MainWindow3()

{

delete ui;

}

mainwindow1.h:

#ifndef MAINWINDOW1\_H

#define MAINWINDOW1\_H

#include <QMainWindow>

namespace Ui{

class MainWindow1;

}

class MainWindow1 : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow1(QWidget \*parent = 0);

~MainWindow1();

public slots:

void action();

private:

Ui::MainWindow1 \*ui;

QTimer \*timer ;

};

#endif // MAINWINDOW1\_H

mainwindow2.h:

#ifndef MAINWINDOW2\_H

#define MAINWINDOW2\_H

#include <QMainWindow>

namespace Ui{

class MainWindow2;

}

class MainWindow2 : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow2(QWidget \*parent = 0);

~MainWindow2();

public slots:

void action();

private:

Ui::MainWindow2 \*ui;

QTimer \*timer ;

int cnt;

};

#endif // MAINWINDOW2\_H

mainwindow3.h:

#ifndef MAINWINDOW3\_H

#define MAINWINDOW3\_H

#include <QMainWindow>

namespace Ui {

class MainWindow3;

}

class MainWindow3 : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow3(QWidget \*parent = 0);

~MainWindow3();

public slots:

void action();

private:

Ui::MainWindow3 \*ui;

QTimer \*timer ;

int i;

int sum;

};

#endif // MAINWINDOW3\_H

实验三内核添加设备驱动代码：

Makefile:

ifneq ($(KERNELRELEASE),)

#kbuild syntax.

mymodule-objs :=myDev.o //模块的文件组成

obj-m :=myDev.o //生成的模块文件名

else

PWD :=$(shell pwd)

KVER :=$(shell uname -r)

KDIR :=/lib/modules/$(KVER)/build

all:

$(MAKE) -C $(KDIR) M=$(PWD)

clean:

rm -f \*.cmd \*.o \*.mod \*.ko

endif

Mydev.c:

#include "linux/kernel.h"

#include "linux/module.h"

#include "linux/fs.h"

#include "linux/init.h"

#include "linux/types.h"

#include "linux/errno.h"

#include "linux/uaccess.h"

#include "linux/kdev\_t.h"

#define MAX\_SIZE 1024

static int my\_open(struct inode \*inode, struct file \*file);

static int my\_release(struct inode \*inode, struct file \*file);

static ssize\_t my\_read(struct file \*file, char \_\_user \*user, size\_t t, loff\_t \*f);

static ssize\_t my\_write(struct file \*file, const char \_\_user \*user, size\_t t, loff\_t \*f);

static char message[MAX\_SIZE] = "-------congratulations--------!";

static int device\_num = 0;//设备号

static int counter = 0;//计数用

static int mutex = 0;//互斥用

static char\* devName = "myDevice";//设备名

struct file\_operations pStruct =

{ open:my\_open, release:my\_release, read:my\_read, write:my\_write, };

int init\_module(void){

int ret;

ret = register\_chrdev(0, devName, &pStruct);

if (ret < 0){

printk("regist failure!\n");

return -1;

}

else{ device\_num = ret;

printk("the device has been registered!\n");

printk("the major number is %d.!\n", device\_num);

return device\_num;

}

}

void cleanup\_module(void){

unregister\_chrdev(device\_num, devName);

printk("unregister it success!\n");

}

static int my\_open(struct inode \*inode, struct file \*file)

{

if(mutex)

return -EBUSY;

mutex = 1;//上锁

try\_module\_get(THIS\_MODULE);

return 0;

}

static int my\_release(struct inode \*inode, struct file \*file)

{

printk("Device released!\n");

module\_put(THIS\_MODULE);

mutex = 0;//开锁

return 0;

}

static ssize\_t my\_read(struct file \*file, char \_\_user \*user, size\_t t, loff\_t \*f){

if(copy\_to\_user(user,message,sizeof(message))){

return -EFAULT;

}

return sizeof(message);

}

static ssize\_t my\_write(struct file \*file, const char \_\_user \*user, size\_t t, loff\_t \*f){

if(copy\_from\_user(message,user,sizeof(message))){

return -EFAULT;

}

return sizeof(message);

}

Test.c:

#include<stdio.h>

#include<stdlib.h>

#include<linux/fcntl.h>

#define MAX\_SIZE 1024

int main(void){

int testdev;

char buf[MAX\_SIZE];

char get[MAX\_SIZE];

testdev = open("/dev/myDev1",O\_RDWR|O\_NONBLOCK);

if(testdec!=-1){

printf("Pleasee input a string:\n");

gets(get);

write(testdev,get,sizeof(get));

read(testdev,buf,sizeof(buf));

printf("\n The string of device is %s\n",buf);

close(testdev);

}

return 0;

}

实验四系统监控器代码：

Mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~MainWindow();

QList<double> cpulist;

QList<double> memlist;

void HostName();

void BootTime();

void OsType();

void CpuType();

void addCpuPoint(double);

void addMemPoint(double);

void killProcess(int);

public slots:

void RunTime();

void RemovePro();

void NowTime();

void CpuRate();

void MemRate();

void ProcessInfo();

void UpdateCpuLine();

void UpdateMemLine();

private:

Ui::MainWindow \*ui;

QTimer \*timer;

};

#endif // MAINWINDOW\_H

Cpu.h:

#ifndef CPU\_H

#define CPU\_H

#include<stdio.h>

typedef struct CPU{

char name[20];//cpu name

unsigned user ;

unsigned nice ;

unsigned system ;

unsigned idle;

unsigned iowait;

unsigned irq;

unsigned softirq;

}cpuinfo;

void GetCpuInfo(cpuinfo \* s\_cpu){

FILE \*f;

char cpu[200];

f=fopen("/proc/stat","r");

fgets(cpu,sizeof(cpu),f);

sscanf(cpu,"%s %u %u %u %u %u %u %u",s\_cpu->name,&s\_cpu->user,&s\_cpu->nice,&s\_cpu->system,&s\_cpu->idle,&s\_cpu->iowait,&s\_cpu->irq,&s\_cpu->softirq);

fclose(f);

}

double CalcCpuRate(cpuinfo \*s\_cpu1,cpuinfo \*s\_cpu2){

int total1=s\_cpu1->user+s\_cpu1->nice+s\_cpu1->system+s\_cpu1->idle+s\_cpu1->iowait+s\_cpu1->irq+s\_cpu1->softirq;

int total2=s\_cpu2->user+s\_cpu2->nice+s\_cpu2->system+s\_cpu2->idle+s\_cpu2->iowait+s\_cpu2->irq+s\_cpu2->softirq;

if(total1!=total2){

return (double)(100.0-(s\_cpu2->idle-s\_cpu1->idle)\*100.0/(total2-total1));

}

else return 0.0;

}

#endif // CPU\_H

Sys.h:

#ifndef SYS\_H

#define SYS\_H

#include<stdio.h>

#include<stdlib.h>

#include<string>

#include<iostream>

#include<QString>

#endif // SYS\_H

std::string GetHostName(){

std::ifstream inputstream("/proc/sys/kernel/hostname");

std::string str;

inputstream>>str;

inputstream.close();

return str;

}

double GetRunTime(){

std::ifstream inputstream("/proc/uptime");

double runtime;

inputstream>>runtime;

inputstream.close();

return runtime;

}

std::string GetOsType(){

std::ifstream inputstream1("/proc/sys/kernel/ostype");

std::string osname;

inputstream1>>osname;

std::ifstream inputstream2("/proc/sys/kernel/osrelease");

std::string ostype;

inputstream2>>ostype;

inputstream1.close();

inputstream2.close();

return osname+" "+ostype;

}

std::string GetCpuType(){

std::ifstream inputstream("/proc/cpuinfo");

std::string temp,cputype;

for (int i=1;i<=16;i++)

inputstream>>temp;

for(int i=1;i<=6;i++){

inputstream>>temp;

cputype+=temp;

cputype+=" ";

}

inputstream.close();

return cputype;

}

Memory.h:

#ifndef MEMORY\_H

#define MEMORY\_H

#include<stdio.h>

#include<stdlib.h>

#include<fstream>

#include<iostream>

#include<string>

typedef struct MEM{

unsigned long total;

unsigned long free;

unsigned long buffers;

unsigned long cached;

}meminfo;

void GetMemInfo(meminfo \*mem){

std::string str;

unsigned long memtotal,memfree,memav,membuffer,memcached;

std::ifstream inputstream("/proc/meminfo");

inputstream>>str>>memtotal>>str;

inputstream>>str>>memfree>>str;

inputstream>>str>>memav>>str;

inputstream>>str>>membuffer>>str;

inputstream>>str>>memcached>>str;

mem->buffers=membuffer;

mem->cached=memcached;

mem->free=memfree;

mem->total=memtotal;

inputstream.close();

}

double CalcMemRate(meminfo \*mem){

return (double)((mem->total-mem->buffers-mem->cached-mem->free)\*100.0/mem->total);

}

#endif // MEMORY\_H

Process.h:

#ifndef PROCESS\_H

#define PROCESS\_H

#include<stdio.h>

#include<stdlib.h>

#include<string>

#include<unistd.h>

#include<dirent.h>

#include<iostream>

#endif // PROCESS\_H

using namespace std;

struct PROC\_INFO{

string name;

string pid;

string ppid;

string rss;

string priority;

}proinfo[16384];

void ReadPro(PROC\_INFO\* pinfo,const char \*pid){

string temp,pidname;

string s\_pid =pid;

std::ifstream inputstream("/proc/"+s\_pid+"/stat");

inputstream>>(pinfo->pid)>>pidname>>temp>>(pinfo->ppid)>>temp>>temp;

inputstream>>temp>>temp>>temp>>temp>>temp>>temp;

inputstream>>temp>>temp>>temp>>temp>>temp>>(pinfo->priority);

inputstream>>temp>>temp>>temp>>temp>>temp>>(pinfo->rss);

pinfo->name=pidname.substr(1,pidname.find(')')-1);

inputstream.close();

}

int GetProcessInfo(){

DIR \*dir;

struct dirent \*ptr;

int cnt=0;

if(!(dir=opendir("/proc")))

return 0;

while((ptr=readdir(dir))!=false){

if(ptr->d\_name[0]>='1'&&ptr->d\_name[0]<='9'){

ReadPro(&(proinfo[cnt]),ptr->d\_name);

cnt++;

}

}

closedir(dir);

return cnt;

}

Mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include<QTimer>

#include<QDateTime>

#include<QDebug>

#include<QPainter>

#include"cpu.h"

#include"memory.h"

#include"sys.h"

#include<stdio.h>

#include<process.h>

#include<stdlib.h>

#include<time.h>

#include<unistd.h>

#include<QPushButton>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

HostName();

BootTime();

OsType();

CpuType();

timer =new QTimer(this);

timer->start(1);

connect(timer,SIGNAL(timeout()),this,SLOT(RunTime()));

connect(timer,SIGNAL(timeout()),this,SLOT(CpuRate()));

connect(timer,SIGNAL(timeout()),this,SLOT(NowTime()));

connect(timer,SIGNAL(timeout()),this,SLOT(MemRate()));

QTimer\* timer1= new QTimer(this);

timer1->start(5);

connect(timer1,SIGNAL(timeout()),this,SLOT(ProcessInfo()));

}

void MainWindow::CpuRate(){

cpuinfo cpu1,cpu2;

double cpurate;

GetCpuInfo(&cpu1);

sleep(1);

GetCpuInfo(&cpu2);

cpurate=CalcCpuRate(&cpu1,&cpu2);

ui->textBrowser\_2->setText("CPU浣跨敤鐜?"+QString::number(cpurate,10,2)+"%");

ui->textBrowser\_2->setAlignment(Qt::AlignCenter);

addCpuPoint(cpurate);

}

void MainWindow::NowTime(){

QDateTime time = QDateTime::currentDateTime();

QString nowTime = time.toString("yyyy-MM-dd hh:mm:ss dddd");

ui->textBrowser->setText(nowTime);

ui->textBrowser->setAlignment(Qt::AlignCenter);

}

void MainWindow::MemRate(){

meminfo mem;

double memRate;

GetMemInfo(&mem);

memRate=CalcMemRate(&mem);

ui->textBrowser\_3->setText("memory浣跨敤鐜?"+QString::number(memRate,10,2)+"%");

ui->textBrowser\_3->setAlignment(Qt::AlignCenter);

addMemPoint(memRate);

}

void MainWindow::HostName(){

std::string hostname=GetHostName();

ui->Hostlabel->setText("涓绘満鍚? "+QString::fromStdString(hostname));

}

void MainWindow::RunTime(){

int rtime=(int)GetRunTime();

int days = (rtime)/(24\*3600);

int hours = (rtime)%(24\*3600)/3600;

int minutes = (rtime)%3600/60;

int second = (rtime)%60;

ui->runtimelabel->setText("绯荤粺杩愯鏃堕棿: "+QString::number(days,10)+"澶?+QString::number(hours,10)+"灏忔椂"+QString::number(minutes,10)+"鍒嗛挓"+QString::number(second,10)+"绉?);

}

void MainWindow::BootTime(){

int boottime=QDateTime::currentDateTime().toTime\_t()-(int)GetRunTime();

ui->Bootlabel->setText("绯荤粺鍚姩鏃堕棿: "+QDateTime::fromTime\_t(boottime).toString("yyyy-MM-dd hh:mm:ss.zzz"));

}

void MainWindow::OsType(){

std::string os= GetOsType();

ui->Oslabel->setText("绯荤粺璁剧疆: "+QString::fromStdString(os));

}

void MainWindow::CpuType(){

std::string cputype= GetCpuType();

ui->cpulabel->setText("cpu鍨嬪彿: "+QString::fromStdString(cputype));

}

void MainWindow::addCpuPoint(double cpurate){

int size =cpulist.size();

if(size>120) cpulist.pop\_front();

if(size==0)

for(int i=0;i<119;i++) cpulist.push\_back(0);

if(size>=1){

cpulist.push\_back(cpurate);

emit UpdateCpuLine();

}

else cpulist.push\_back(cpurate);

}

void MainWindow::UpdateCpuLine(){

int count=0;

QPixmap pix(600,160);

QPainter painter(&pix);

pix.fill(Qt::blue);

QPen pen0;

pen0.setColor(Qt::lightGray);

painter.setPen(pen0);

for(int i=1;i<4;i++) painter.drawLine(0,i\*40,600,i\*40);

QPen pen;

pen.setColor(Qt::red);

pen.setWidth(2);

painter.setPen(pen);

while(count<(cpulist.size()-1)){

painter.drawLine(5\*count,160-1.6\*(cpulist.value(count)),5\*count+5,160-1.6\*(cpulist.value(count+1)));

count++;

}

ui->cpulinelabel->setPixmap(pix);

}

void MainWindow::addMemPoint(double memrate){

int size =memlist.size();

if(size>120) memlist.pop\_front();

if(size==0)

for(int i=0;i<119;i++) memlist.push\_back(0);

if(size>=1){

memlist.push\_back(memrate);

emit UpdateMemLine();

}

else memlist.push\_back(memrate);

}

void MainWindow::UpdateMemLine(){

int count=0;

QPixmap pix(600,160);

QPainter painter(&pix);

pix.fill(Qt::blue);

QPen pen0;

pen0.setColor(Qt::lightGray);

painter.setPen(pen0);

for(int i=1;i<4;i++) painter.drawLine(0,i\*40,600,i\*40);

QPen pen;

pen.setColor(Qt::red);

pen.setWidth(2);

painter.setPen(pen);

while(count<(memlist.size()-1)){

painter.drawLine(5\*count,160-1.6\*(memlist.value(count)),5\*count+5,160-1.6\*(memlist.value(count+1)));

count++;

}

ui->memlinelabel\_2->setPixmap(pix);

}

void MainWindow::ProcessInfo(){

ui->tableWidget->setColumnCount(6);

QStringList headers;

headers<<"杩涚▼鍚?<<"pid"<<"ppid"<<"鍐呭瓨鍗犵敤/KB"<<"浼樺厛绾?<<"鎿嶄綔";

ui->tableWidget->setHorizontalHeaderLabels(headers);

ui->tableWidget->horizontalHeader()->setStretchLastSection(true);

int proNum=GetProcessInfo();

ui->tableWidget->setRowCount(proNum);

for(int i=0;i<proNum;i++){

QTableWidgetItem \*nameItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].name));

ui->tableWidget->setItem(i,0,nameItem);

nameItem->setTextAlignment(Qt::AlignCenter);

QTableWidgetItem \*pidItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].pid));

ui->tableWidget->setItem(i,1,pidItem);

pidItem->setTextAlignment(Qt::AlignCenter);

QTableWidgetItem \*ppidItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].ppid));

ui->tableWidget->setItem(i,2,ppidItem);

ppidItem->setTextAlignment(Qt::AlignCenter);

QTableWidgetItem \*rssItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].rss));

ui->tableWidget->setItem(i,3,rssItem);

rssItem->setTextAlignment(Qt::AlignCenter);

QTableWidgetItem \*priorityItem = new QTableWidgetItem(QString::fromStdString(proinfo[i].priority));

ui->tableWidget->setItem(i,4,priorityItem);

priorityItem->setTextAlignment(Qt::AlignCenter);

QPushButton \*btn = new QPushButton;

btn->setText("缁撴潫杩涚▼");

btn->setStyleSheet("text-align:center;");

ui->tableWidget->setCellWidget(i,5,btn);

connect(btn,SIGNAL(clicked()),this,SLOT(RemovePro()));

}

}

void MainWindow::RemovePro(){

QPushButton \*pushBtn =dynamic\_cast<QPushButton\*>(this->sender());

if(pushBtn ==0) return ;

int x=pushBtn->frameGeometry().x();

int y=pushBtn->frameGeometry().y();

QModelIndex index =ui->tableWidget->indexAt(QPoint(x,y));

int row=index.row();

int colum =index.column();

qDebug()<<QObject::tr("row:%1,colum:%2").arg(row).arg(colum);

killProcess(row);

}

void MainWindow::killProcess(int i){

QString command;

QString id=QString::fromStdString(proinfo[i]. pid);

command = QString("kill %1").arg(id);

QByteArray ba= command.toLatin1();

system(ba.data());

ProcessInfo();

}

MainWindow::~MainWindow()

{

delete ui;

}

Main.cpp:

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.move(200,200);

//w.resize(650,478);

w.setFixedSize(650,478);

w.show();

return a.exec();

}

实验五文件系统代码：

#include<windows.h>

#include<string.h>

#include<stdio.h>

#include<stdbool.h>

#include<time.h>

#include<stdlib.h>

#define Disk\_Size 1023

#define Inode\_Num 100

#define Block\_Maxnum 300

#define Text\_Maxsize 3000

#define Child\_Maxnum 6

int inode\_bitmap[Inode\_Num]; //Inode位图

int block\_bitmap[Block\_Maxnum]; //Block位图

struct super\_block{

int free\_inode; //空闲Inode数目

int free\_block; //空闲Block数目

} super\_block;

typedef struct Inode{

int isFileOrNot; //判断是文件还是目录,4byte;0为目录,1为文件

char name[26]; //文件或目录名，26byte

int block\_num; //所占用的磁盘块数，4byte

int block\_pos[7]; //占用的磁盘块地址，7\*4=28byte

int index; //在inode区中的编号，4byte

int parent; //父索引节点，4byte

int child\_num; //子节点个数，4byte

int child[Child\_Maxnum]; //子索引节点，6\*4=24byte

char ctime[26]; //文件创建时间,26byte

int role //文件创建者的角色

} inode;

typedef struct block{

char content[Disk\_Size+1];

} block;

int curr; //当前索引值

char current[512]; //当前路径

int current\_num[128]; //当前索引值路径

int level=1; //当前索引深度

time\_t t;

int admin; //当前用户权限

int block\_used=0;

int inode\_used=0;

bool init (inode \*\*iNodeList);

bool create\_inode(char \*name, inode \*\*iNodeList, int seq);

bool input(char \*text, block \*\*blockList,inode \*\*iNodeList, int seq);

void destroy(char \*name,inode \*\*iNodeList);

void destroy\_file(int num, inode \*\*iNodeList);

void remove\_array(int \*a, int amount, int num);

void destroy\_folder(int num, inode \*\*iNodeList); //当前路径

void path\_update(inode \*iNodeList);

int open(inode \*iNodeList, char \*name);

void show(inode \*iNodeList,block \*blockList,int n);

void back(inode \*iNodeList);

void menu(void);

int main(){

inode \*iNodeList = NULL;

block \*blockList=NULL;

int option;

init(&iNodeList);

printf("\n\n");

printf(" file\_system \n");

printf("------------------------------------------------------\n");

printf(" Please choose your role :(0:admin 1:user)\n");

scanf("%d",&admin);

system("cls");

do{

printf(" Please input your option:");

menu();

printf("当前路径：%s ",current);

if(admin) printf("用户角色：user\n");

else printf("用户角色：administrator\n");

printf("选择操作：");

scanf("%d", &option);

getchar();

printf("\n");

switch(option){

case 0:

break;

case 1:

printf(" Please choose your role :(0:admin 1:user)\n");

scanf("%d",&admin);

break;

case 2:{

int i;

if(super\_block.free\_inode == 0 ){

printf("Not enough storage space!\n");

break;

}

if(iNodeList[curr].child\_num >= Child\_Maxnum){

printf("Subfiles is full!\n");

break;

}

for(i = 0; i < Inode\_Num; i++){

if(!inode\_bitmap[i] ){

char folder\_name[26], text[Text\_Maxsize];

printf("Please input the name of folder:");

scanf("%s", &folder\_name);

getchar();

if(create\_inode(&folder\_name, &iNodeList, i)){

iNodeList[i].isFileOrNot = 0;

iNodeList[i].index = i;

iNodeList[i].role = admin;

iNodeList[curr].child[iNodeList[curr].child\_num] = i;

iNodeList[curr].child\_num++;

iNodeList[i].child\_num = 0;

super\_block.free\_inode--;

inode\_bitmap[i] = 1;

printf("Creating Success!\n");

break;

}

else

break;

}

}

break;

}

case 3:{//create file

int i;

if(super\_block.free\_inode == 0 && super\_block.free\_block == 0){

printf("Not enough storage space!\n");

break;

}

if(iNodeList[curr].child\_num >= Child\_Maxnum){

printf("Subfiles is full!\n");

break;

}

for(i = 0; i < Inode\_Num; i++){

if(!inode\_bitmap[i])

{

char file\_name[28], text[Text\_Maxsize];

printf("Please input the name of file:");

scanf("%s", &file\_name);

getchar();

if(create\_inode(&file\_name, &iNodeList, i)){

iNodeList[i].isFileOrNot = 1;

iNodeList[i].index = i;

iNodeList[i].role = admin;

iNodeList[curr].child[iNodeList[curr].child\_num] = i;

iNodeList[curr].child\_num++;

super\_block.free\_inode--;

inode\_bitmap[i] = 1;

}

else{

break;

}

printf("Please input text: (End by $)\n");

char c=0;int cnt=0;

while(c!='$'){

scanf("%c",&c);

if(c=='$') break;

text[cnt++]=c;

}

text[cnt]='\0';

getchar();

printf("\n");

if(input(text, &blockList, &iNodeList,i))

printf("Creating Success!\n");

else

printf("Failed to create!\n");

break;

}

}

break;

}

case 4:{//打开文件夹

char name[28],text[3000];

int n;

printf("Please input name:");

scanf("%s",name);

getchar();

printf("\n");

n=open(iNodeList,name);

if(n==-1||iNodeList[n].isFileOrNot==1){

printf("folder does not exist!\n");

break;

}

if(iNodeList[n].role<admin){

printf("permission denied\n");

break;

}

show(iNodeList,blockList,n);

printf("Open success!\n");

break;

}

case 5:{//打开文件

char name[28],text[Text\_Maxsize];;

int n;

printf("Please input name:");

scanf("%s",name);

getchar();

printf("\n");

n=open(iNodeList,name);

if(n==-1||iNodeList[n].isFileOrNot==0){

printf("File does not exist!\n");

break;

}

show(iNodeList,blockList,n);

printf("Open success!\n");

if(iNodeList[n].role<admin){

printf("Not enough privileges to write!\n");

break;

}

printf("Update the text or not ?(y/n)");

char s[2];

scanf("%s",s);

if(s[0]=='y'||s[0]=='Y'){

printf("Please input text: (End by $)\n");

char c=0;int cnt=0;

while(c!='$'){

scanf("%c",&c);

if(c=='$') break;

text[cnt++]=c;

}

text[cnt]='\0';

getchar();

printf("\n");

if(input(text, &blockList, &iNodeList,n))

printf("Update Success!\n");

else

printf("Failed to Update!\n");

break;

}

else getchar();

break;

}

case 6://返回上一级

back(iNodeList);

break;

case 7:

list(iNodeList);

break;

case 8:{//删除

char name[28];

printf("Please input name:");

scanf("%s",name);

getchar();

destroy(name,&iNodeList);

printf("Delete success!\n");

break;

}

case 9:{

int i;

for(i=0; i<Inode\_Num; i++){

if(!inode\_bitmap[i])

printf("-");

else

printf("0");

}

break;

}

case 10:{

int i;

for(i=0; i<Block\_Maxnum; i++){

if(!block\_bitmap[i])

printf("-");

else

printf("0");

}

break;

}

case 11:{

char name[28];

int n;

printf("Please input name:");

scanf("%s",name);

getchar();

printf("\n");

n=open(iNodeList,name);

if(n==-1){

printf("File does not exist!\n");

break;

}

if(!iNodeList[n].isFileOrNot)

printf("name:%s\ncreation time:%s\n",iNodeList[n].name,iNodeList[n].ctime);

else

printf("name:%s\ncreation time:%snsize:%dKB\n",iNodeList[n].name,iNodeList[n].ctime,iNodeList[n].block\_num);

break;

}

}

getchar();

system("cls");

}

while(option != 0);

printf("\n--------------------Welcome again!--------------------\n");

}

/\*初始化整个文件系统\*/

bool init (inode \*\*iNodeList){

super\_block.free\_inode = Inode\_Num;

super\_block.free\_block = Block\_Maxnum;

int i;

for(i = 0; i < Inode\_Num; i++)

inode\_bitmap[i] = 0;

for(i = 0; i < Block\_Maxnum; i++)

block\_bitmap[i] = 0;

inode\_bitmap[0] = 1;

curr = 0;

strcpy(current, "Root//");

current\_num[0] = 0;

\*iNodeList=(inode \*)malloc(sizeof(inode));

(\*iNodeList)->isFileOrNot = 0;

strcpy((\*iNodeList)->name, "Root");

(\*iNodeList)->index = curr;

(\*iNodeList)->parent = curr;

(\*iNodeList)->child\_num=0;

time(&t);

strcpy((\*iNodeList)->ctime,ctime(&t));

return true;

}

/\*在当前路径下新建一个文件夹或文件，名称与已有的不能重复\*/

bool create\_inode(char \*name, inode \*\*iNodeList, int seq){

int i;

if(strlen(name) > 28){

printf("File name is too long!\n");

return false;

}

for(i = 0; i < ((\*iNodeList)[curr].child\_num); i++){

if(!strcmp(name, (\*iNodeList)[(\*iNodeList)[curr].child[i]].name)){

printf("iNodeListhe same file name!\n");

return false;

}

}

if(seq>=inode\_used)

{

\*iNodeList=(inode \*)realloc(\*iNodeList,(seq+1)\*sizeof(inode));

inode\_used=seq;

}

strcpy((\*iNodeList)[seq].name , name);

(\*iNodeList)[seq].parent = curr;

time(&t);

strcpy((\*iNodeList)[seq].ctime,ctime(&t));

return true;

}

/\*为一个已经创建的文件写入内容，注意这里一个文件最多占用7个空闲块\*/

bool input(char \*text, block \*\*blockList, inode \*\*iNodeList,int seq){

if(strlen(text) >= 7 \* Disk\_Size)

{

printf("File size is too large!\n");

return false;

}

int m = strlen(text) / Disk\_Size ;

int n = strlen(text) % Disk\_Size ;

(\*iNodeList)[seq].block\_num = m + 1;

int j, w = 0;

for(j = 0; j <= m ; j++){

if(super\_block.free\_block == 0){

printf("Not enough storage space!\n");

return false;

}

do{

if(!block\_bitmap[w]){

if(w>=block\_used){

\*blockList=(block \*)realloc(\*blockList,(w+1)\*sizeof(block));

block\_used=w;

}

if(j < m){

strncpy((\*blockList+w)->content, text + j \* (Disk\_Size), Disk\_Size );

((\*blockList+w)->content)[Disk\_Size]='\0';

}

else if(j == m){

strncpy((\*blockList+w)->content, text + j \* (Disk\_Size) , n );

((\*blockList+w)->content)[n]='\0';

}

(\*iNodeList)[seq].block\_pos[j] = w;

super\_block.free\_block--;

block\_bitmap[w] = 1;

break;

}

w++;

}

while(w < Block\_Maxnum);

}

return true;

}

/\*以名字删除一个文件或文件夹\*/

void destroy(char \*name,inode \*\*iNodeList){

int i=(\*iNodeList+curr)->child\_num;

while(i>0){

if(!strcmp(name,(\*iNodeList)[(\*iNodeList)[curr].child[i-1]].name)){

if(!(\*iNodeList)[(\*iNodeList)[curr].child[i-1]].isFileOrNot)

destroy\_folder((\*iNodeList)[curr].child[i-1],iNodeList);

else

destroy\_file((\*iNodeList)[curr].child[i-1],iNodeList);

return;

}

i--;

}

printf("File not exist!\n");

}

/\*删除一个文件

\*/

void destroy\_file(int num, inode \*\*iNodeList){

inode\_bitmap[num] = 0;

int i = (\*iNodeList)[num].block\_num - 1;

do{

block\_bitmap[(\*iNodeList)[num].block\_pos[i]] = 0;

i--;

}while(i > 0);

remove\_array((\*iNodeList)[(\*iNodeList)[num].parent].child, (\*iNodeList)[(\*iNodeList)[num].parent].child\_num, num);

(\*iNodeList)[(\*iNodeList)[num].parent].child\_num--;

}

/\*在父i节点的孩子列表中删除num这个节点\*/

void remove\_array(int \*a, int amount, int num){

int i;

for(i = 0; i <= amount - 1; i++){

if(a[i] == num){

do{

a[i] = a[i + 1];

i++;

}

while(i <= amount - 2);

}

}

}

/\*递归删除一个文件夹及其所有子文件

\*/

void destroy\_folder(int num, inode \*\*iNodeList){

inode\_bitmap[num] = 0;

(\*iNodeList)[(\*iNodeList)[num].parent].child\_num--;

remove\_array((\*iNodeList)[(\*iNodeList)[num].parent].child, (\*iNodeList)[(\*iNodeList)[num].parent].child\_num, num);

if((\*iNodeList)[num].isFileOrNot == 1){

destroy\_file(num, (\*iNodeList));

}

int i = (\*iNodeList)[num].child\_num - 1;

while(i>=0){

destroy\_folder((\*iNodeList)[num].child[i], iNodeList);

i--;

}

}

/\*更新当前路径

\*/

void path\_update(inode \*iNodeList){

int i = 0;

memset(current, 0, sizeof (char) \* 512);

do{

strcat(current, iNodeList[current\_num[i]].name);

strcat(current, "/");

i++;

}

while(i<level);

}

/\*以名字打开一个文件或目录，返回i节点编号

若此名字不存在返回-1

\*/

int open(inode \*iNodeList, char \*name){

int i, n;

for(i = 0; i < iNodeList[curr].child\_num; i++){

if(!strcmp(iNodeList[iNodeList[curr].child[i]].name, name)){

n = iNodeList[curr].child[i];

return n;

}

}

return -1;

}

/\*若是一个目录则进入此目录，若是一个文件则打开显示该文件内容。

n是要操作的i节点编号

\*/

void show(inode \*iNodeList,block \*blockList,int n){

int i;

if(iNodeList[n].isFileOrNot==0){

curr=n;

current\_num[level]=n;

++level;

path\_update(iNodeList);

}

if(iNodeList[n].isFileOrNot==1)

for(i=0; i<iNodeList[n].block\_num; i++)

printf("%s\n",blockList[iNodeList[n].block\_pos[i]].content);

}

/\*返回上一级目录

\*/

void back(inode \*iNodeList){

curr=iNodeList[curr].parent;

level--;

path\_update(iNodeList);

}

/\*列出当前目录下所有的文件夹和文件名

\*/

void list(inode \*iNodeList){

char file[1000]=" ";

char folder[1000]=" ";

int i=iNodeList[curr].child\_num;

if(i==0){

printf("Empty!\n");

return;

}

i--;

do{

if(iNodeList[iNodeList[curr].child[i]].isFileOrNot){

strcat(file,iNodeList[iNodeList[curr].child[i]].name);

strcat(file,"\t");

}

else{

strcat(folder,iNodeList[iNodeList[curr].child[i]].name);

strcat(folder,"\t");

}

i--;

}

while(i>=0);

printf("folder:%s\nfile:%s",folder,file);

}

/\*显示功能菜单文件

\*/

void menu(){

printf("\n\n");

printf("Menu for file\_system \n");

printf("------------------------------------------------------\n");

printf(" 1. change role \n");

printf("2. create new folder 3. create new file\n");

printf("4. open folder 5.open file\n");

printf(" 6. back \n");

printf("7. list 8. delete\n");

printf("9. inode graph 10.block graph\n");

printf(" 11. detail\n");

printf("0. Exit\n");

printf("------------------------------------------------------\n");

}