



# 华中科技大学

## 操作系统原理课程设计报告

姓 名：刘 美

学 院：计算机科学与技术学院

专 业：计算机科学与技术

班 级：CS1806

学 号：U201814788

指导教师：周 正 勇

分数	
教师签名	

2021 年 04 月 09 日

# 目 录

<b>1 实验一 Linux 用户界面设计.....</b>	<b>1</b>
1.1 实验目的.....	1
1.2 实验内容.....	1
1.3 实验设计.....	1
1.3.1 开发环境.....	1
1.3.2 实验设计.....	1
1.4 实验调试.....	2
1.4.1 实验步骤.....	2
1.4.2 实验调试与心得.....	6
附录 实验代码.....	7
<b>2 实验二 添加系统功能调用.....</b>	<b>20</b>
2.1 实验目的.....	20
2.2 实验内容.....	20
2.3 实验设计.....	20
2.3.1 开发环境.....	20
2.3.2 实验设计.....	20
2.4 实验调试.....	21
2.4.1 实验步骤.....	21
2.4.2 实验调试及心得.....	23
附录 实验代码.....	24
<b>3 实验三 添加字符设备驱动.....</b>	<b>25</b>
3.1 实验目的.....	25
3.2 实验内容.....	25
3.3 实验设计.....	25
3.3.1 开发环境.....	25
3.3.2 实验设计.....	25
3.4 实验调试.....	26
3.4.1 实验步骤.....	26
3.4.2 实验调试及心得.....	27
附录 实验代码.....	28
<b>4 实验四 使用 Qt 实现系统监视器.....</b>	<b>32</b>

4.1 实验目的.....	32
4.2 实验内容.....	32
4.3 实验设计.....	32
4.3.1 开发环境.....	32
4.3.2 实验设计.....	32
4.4 实验调试.....	33
4.4.1 实验步骤.....	33
4.4.2 实验调试及心得.....	35
附录 实验代码.....	38
<b>5 实验五 模拟文件系统.....</b>	<b>61</b>
5.1 实验目的.....	61
5.2 实验内容.....	61
5.3 实验设计.....	61
5.3.1 开发环境.....	61
5.3.2 实验设计.....	61
5.4 实验调试.....	66
5.4.1 实验步骤.....	66
5.4.2 实验调试及心得.....	67
附录 实验代码.....	69

# 1 实验一 Linux 用户界面设计

## 1.1 实验目的

掌握 Linux 操作系统的使用方法；  
了解 Linux 系统内核代码结构；  
掌握实例操作系统的实现方法。

## 1.2 实验内容

编一个 C 程序，其内容为实现文件拷贝的功能。基本要求：使用系统调用 open/read/write。

编一个 C 程序，其内容为分窗口同时显示三个并发进程的运行结果。要求用到 Linux 下的图形库。(Qt) 基本要求：三个独立子进程，各自窗口显示；

## 1.3 实验设计

### 1.3.1 开发环境

内核：Linux version 5.8.0-45-generic  
编译器：gcc 9.3.0  
操作系统版本：Ubuntu 20.04  
内存：8G  
硬盘：100G  
虚拟机：VMware

### 1.3.2 实验设计

任务 1：第一部分的实验要求实现利用系统函数 open, write 等函数实现文件的拷贝。首先需要建立一个有内容的文本文件，使用 open 函数打开，并根据函数的返回值做出异常处理。然后使用 stst 结构体获得文件的大小，并设计一个同等大小的 char 型缓冲区数组，最后再使用 write 函数实现写入。

任务 2: 第二部分的实验是基于 Qt 实现三个进程的誊抄。实验需要设计两个共享缓冲区 s 和 t, 四个信号灯, 分别代表两个缓冲区的空、满数目, 三个进程分别为 get, copy, put。get 将文件的内容复制到 s 缓冲区, copy 将 s 缓冲区的数据复制到 t 缓冲区, 最后由 put 将 t 缓冲区的内容复制到指定文件。缓冲区的大小设置为 100 字节, 三个窗口分别显示对应的进程正在第几次读取缓冲区。将三个进程封装为三个类, 每隔一秒触发一次其对应的进程誊抄函数, 最后设计一个主函数实现三个进程的并发执行。

## 1.4 实验调试

### 1.4.1 实验步骤

#### 1. 任务一

(1) 检查参数输入是否合法, 正确输入为: ./执行文件 输入文件路径 输出文件路径。

```
if(argc<3){
    printf("Please Input More Args!\n");
    return -1;
}
```

(2) 使用 open 函数打开两个文件, 如果文件打开失败需要做出异常处理。

```
int fsrc=open(srcfile,O_RDONLY);
if(fsrc == -1){
    perror("read error");
    exit(1);
}
int fdes=open(desfile,O_CREAT|O_WRONLY,0777);
if(fdes==-1){
    perror("write error");
    exit(1);
}
```

(3) 获得输入文件的大小并设置缓冲区。

```
struct stat statbuf;
stat(srcfile,&statbuf);
int filesize=statbuf.st_size;
char buf[filesize];
```

- (4) 使用 read 和 write 函数实现读写

```
int count=read(fsrc,buf,filesize);
if(count==-1){
    perror("Read error! The file is empty!");
    exit(1);
}
if(write(fdes,buf,filesize)==-1){
    perror("Write Error!");
    exit(1);
}
```

- (5) 关闭文件

```
close(fsrc);
close(fdes);
```

- (6) linux 终端下使用 gcc -o main main.c 生成可执行的二进制文件

- (7) 创建源文件 input.txt 和 output.txt

- (8) 执行 ./main input.txt output.txt

- (9) 返回文件夹查看 output.txt 和 input.txt 文件内容是否相同

## 2. 任务二

- (1) 在 Qt 中设计出三个窗口的样式，并分别修改所对应的窗口的对象名为三个进程的类型：Get，Copy，Put。

- (2) 设计 main.h 头文件，定义实现进程和共享缓冲区需要用到的头文件、结构体和 P,V 函数。

- (3) 在 main.cpp 中使用 fork() 函数创建三个进程，并在对应的进程下执行窗口的建立和显示。

- (4) 在 main.cpp 中，初始化信号灯和两个共享缓冲区，以供其他文件调用。

### ❖ 共享缓冲区

```
int shmid_s=0;
int shmid_t=0;
key_t key_s=ftok("./",68);
key_t key_tt=ftok("../",68);
if((shmid_s=shmget(key_s,sizeof(BUF),IPC_CREAT|0666))== -1){
    puts("shm_s creat failed\n");
    exit(1);
}
if((shmid_t=shmget(key_tt,sizeof(BUF),IPC_CREAT|0666))== -1){
```

```

        puts("shm_t creat failed!\n");
        exit(1);
    }
❖ 信号灯
    int sem_id;
    key_t key_sem=ftok("./",66);
    if((sem_id=semget(key_sem,4,IPC_CREAT|0666))== -1){
        puts("sem created failed\n");
        exit(1);
    }
    union semun arg;
    //s 信号灯
    arg.val=0;//sfull
    semctl(sem_id,0,SETVAL,arg);
    arg.val=1;//sempty
    semctl(sem_id,1,SETVAL,arg);
    //t 信号灯
    arg.val=0;//tfull
    semctl(sem_id,2,SETVAL,arg);
    arg.val=1;//tempty
    semctl(sem_id,3,SETVAL,arg);
    printf("sem finished\n");

```

(5) 三个进程分别对应三个类，并分别写在三个独立文件中实现。其进程的誊抄分别对应函数: Getbuf(), Copybuf(), Putbuf()。

```

❖ Getbuf(){
    if(flag==0){
        P(semid,1);
        memset(bufdata->buf,'\0',LENGTH);
        bufdata->size=bufdata->state=0;
        count++;
        QString getstr="Get: ";
        ui->label->setText(getstr.append(QString::number(count)));
        int len=fread(bufdata->buf,sizeof
(char),LENGTH,writefile);
        filesize +=len;
    }
}

```

```

        printf("len %d, filesize: %d\n",len,filesize);
        if(filesize==0){
            bufdata->state=1;
            bufdata->size=len;
            flag=1;
            fclose(writefile);
            V(semid,0);
        }
        else{
            bufdata->size=LENGTH;
            bufdata->state=0;
            V(semid,0);
        }
    }
}

❖ Copybuf(){
    if(flag==0){
        P(semid,0);    //sfull -1
        P(semid,3);    //tempty -1
        count++;
        QString getstr="Copy: ";
        ui->label->setText(getstr.append(QString::number(count)));
        memcpy(bufdata_t->buf,bufdata_s->buf,bufdata_s->size);
        bufdata_t->size=bufdata_s->size;
        bufdata_t->state=bufdata_s->state;
        if(bufdata_t->state==1) //判断结束
            flag=1;
        V(semid,1);    //sempty +1
        V(semid,2);    //tfull +1
    }
}

❖ Putbuf(){
    if(flag==0){
        P(semid,2);

```



```

        count++;
        QString putstr = "PUT: ";
        ui->label->setText(putstr.append(QString::number(count)));
        fwrite(bufdata->buf,sizeof(char),bufdata->size,readfile);
        if(bufdata->state==1){
            fclose(readfile);
            flag=1;
        }
        V(semid,3);
    }
}

```

(6) 在 Qt 下，点击运行，直到三个窗口的显示的数字不在改变，意味着文件的誊抄结束，关闭三个窗口，比较输出文件和输入文件的内容。

## 1.4.2 实验调试与心得

### 1. 任务一

执行命令，并观察两个文件的内容。



```

lm@lm-virtual-machine:~/桌面/OS/lab1/lab1_1$ ./lab1_1 input.txt output.txt

```

图 1-1 执行指令



图 1-2 比较文件大小

比较两个文件的输出内容和文件大小，发现一致，符合实验预期。

### 2. 任务二

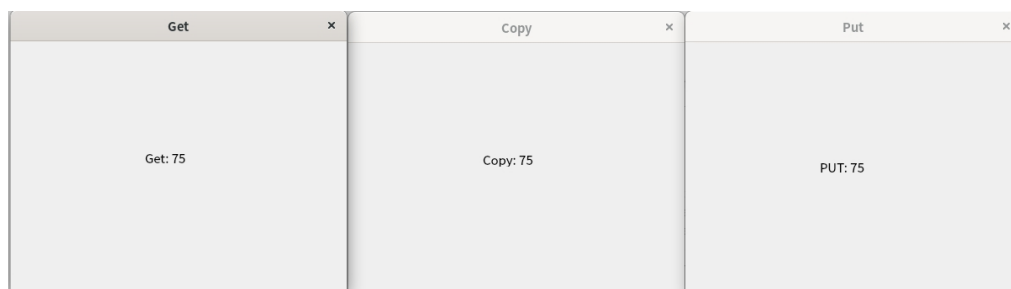


图 1-3 三个进程的窗口显示

文件大小为 7487 字节，缓冲区大小为 100 字节，需要读取 75 次缓冲区，观察实验结果和预期一致。

### 3. 实验心得

本次实验分为两个部分。第一个实验实现文件的拷贝，实验思路简单，检查完后再发现自己在进行拷贝的时候一次性拷贝了所有的内容，但其实针对于比较大的文件这种做法不太合适，应该设定指定大小的缓冲数组，进行循环复制。

第二个实验实现三个进程的并发誊抄。誊抄部分的代码实现并不困难，主要是对 Qt 的使用。

在实验中遇到过许多关于 Qt 的问题，比如使用 ui->自定义控件名，编译报错无法识别，是因为没有修改对应的窗口的对象名为自己设定的类名。其次，在 QtCeator 中有自己的函数库，有些函数在 linux 下可以使用但在 Qt 下却会报错比如 waitpid() 函数，这需要查看 Qt 文档并转换思路。通过这次实验,我对 Qt 的图形设计和函数使用有了初步的了解。

## 附录 实验代码

#### ❖ 任务一：

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>

int main (int argc, char* argv[]){
    //the true format is the srcfile path, the des path
    if(argc<3){
        //the args inputed is less
        printf("Please Input More Args!\n");
        return -1;
    }
    const char* srcfile=argv[1];
    const char* desfile=argv[2];
    int fsrc=open(srcfile,O_RDONLY);
    if(fsrc == -1){
        perror("read error");
        exit(1);
    }
    int fdes=open(desfile,O_CREAT|O_WRONLY,0777);
```

```

    if(fdes==-1){
        perror("write error");
        exit(1);
    }
    struct stat statbuf;
    stat(srcfile,&statbuf);
    int filesize=statbuf.st_size;    //获得要读取的文件的大小
    printf("filesize=%d\n",filesize);

    char buf[filesize];
    int count=read(fsrc,buf,filesize);
    if(count==-1){
        perror("Read error! The file is empty!");
        exit(1);
    }
    if(write(fdes,buf,filesize)==-1){
        perror("Write Error!");
        exit(1);
    }
    close(fsrc);
    close(fdes);
}

```

## ❖ 任务二

### ❖ main.h

```

#ifndef MAIN_H
#define MAIN_H
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#define LENGTH 100
#define TIME 100 //设定界面的触发时间，每 1s 更新一次
union semun {
    int val; /* value for SETVAL */
    struct semid_ds *buf; /* buffer for IPC_STAT, IPC_SET */
    unsigned short *array; /* array for GETALL, SETALL */
    struct seminfo *__buf; /* buffer for IPC_INFO */
}

```

```

        void * __pad;
    };

    static void P(int semid,int index){
        struct sembuf sem;
        sem.sem_num=index;        //要操作的信号灯的编号置为 index
        sem.sem_op=-1;            //执行-1 操作
        sem.sem_flg=0;
        semop(semid,&sem,1);
        return ;
    }

```

```

    static void V(int semid,int index){
        struct sembuf sem;
        sem.sem_num=index;
        sem.sem_op=1;
        sem.sem_flg=0;
        semop(semid,&sem,1);
        return ;
    }

```

```

typedef struct BUFF{        //定义缓冲区块结构
    int size;    //当前缓冲区的大小
    int state;    //判断是否为最后一次读取
    char buf[LENGTH];    //缓冲区内容

}BUF;

```

```

#endif // MAIN_H

```

#### ❖ main.cpp

```

#include "mainwindow.h"
#include <QApplication>
#include "main.h"
#include "get.h"
#include "put.h"
#include "copy.h"

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

```

```

    printf("main\n");

```

```

    //实现三个进程的誊抄

```

//设置两个缓冲区 s 和 t, get 将文件读到缓冲区 s 中, copy 将数据从 s 中读到 t 中, put 将 t 中的数据读到输出文件中

```

        int shmid_s=0;

```

```

        int shmid_t=0;

```

```

key_t key_s=ftok("./",68);
key_t key_tt=ftok("./",68);
if((shmid_s=shmget(key_s,sizeof(BUF),IPC_CREAT|0666))== -1){
    puts("shm_s creat failed\n");
    exit(1);
}
printf("shmget\n");
BUF * buff = (BUF *)shmat(shmid_s, NULL, 0);
buff->size = 0;
buff->state=0;
if((shmid_t=shmget(key_tt,sizeof(BUF),IPC_CREAT|0666))== -1){
    puts("shm_t creat failed!\n");
    exit(1);
}
buff = (BUF *)shmat(shmid_t, NULL, 0);
buff->size = 0;
buff->state=0;
printf("shm finished\n");

//创建信号灯集
int sem_id;
key_t key_sem=ftok("./",66);
if((sem_id=semget(key_sem,4,IPC_CREAT|0666))== -1){
    puts("sem created failed\n");
    exit(1);
}
union semun arg;
//s 信号灯
arg.val=0;//sfull
semctl(sem_id,0,SETVAL,arg);
arg.val=1;//sempty
semctl(sem_id,1,SETVAL,arg);
//t 信号灯
arg.val=0;//tfull
semctl(sem_id,2,SETVAL,arg);
arg.val=1;//tempty
semctl(sem_id,3,SETVAL,arg);
printf("sem finished\n");

/*创建子进程*/
static pid_t p1;
if((p1=fork())==0){ //the first
    printf("pid1 get create\n");
    QApplication a(argc, argv); //定义并创建应用程序

```

```

    QString title="Get";
    Get w; //定义并创建窗口
    w.show(); //显示窗口
    a.exec(); //应用程序运行
    exit(0);
}
static pid_t p2;
if((p2=fork())==0){ //the second
    printf("pid2 copy create\n");
    QApplication a(argc, argv); //定义并创建应用程序
    QString title="Copy";
    Copy w; //定义并创建窗口
    w.show(); //显示窗口
    a.exec(); //应用程序运行
    exit(0);
}

static pid_t p3;
if((p3=fork())==0){ //the third
    printf("pid3 put create\n");
    QApplication a(argc, argv); //定义并创建应用程序
    QString title="Put";
    Put w; //定义并创建窗口
    w.show(); //显示窗口
    a.exec(); //应用程序运行
    exit(0);
}

printf("main create\n");
QApplication a(argc, argv); //定义并创建应用程序 //the father
/*删除信号灯*/
semctl(sem_id,0,IPC_RMID,arg);
semctl(sem_id,1,IPC_RMID,arg);
semctl(sem_id,2,IPC_RMID,arg);
semctl(sem_id,3,IPC_RMID,arg);

/*删除共享内存*/
shmctl(shmid_s,IPC_RMID,0);
shmctl(shmid_t,IPC_RMID,0);
return a.exec();
}

```

❖ get.h

```
#ifndef GET_H
```

```

#define GET_H

#include "main.h"
#include <QWidget>
QT_BEGIN_NAMESPACE
namespace Ui { class Get; } //用于描述界面组件
QT_END_NAMESPACE
class Get: public QWidget //实现一个窗体类
{
    Q_OBJECT

public:
    explicit Get(QWidget *parent = nullptr);
    ~Get();

public slots:
    void GetBuf(); //获取缓冲区内容函数

private:
    Ui::Get *ui; //指针 ui 是指向可视化设计的界面，后面会看到要访问界面上的
    //在定义实现誊抄进程的成员变量
    int count=0;
    int flag=0;
    FILE* writefile;
    int filesize;
    int shmid=0; //在初始化时使用方括号清零，就可以不用用到 memset 函数
    int semid;
    key_t key;
    BUF* bufdata={0};
};
#endif // GET_H

```

#### ❖ get.cpp

```

#include "get.h"
#include "ui_get.h"
#include <QTimer>
//类的成员函数
Get::Get(QWidget *parent):
    QWidget(parent),
    ui(new Ui::Get)
{
    ui->setupUi(this);
    ui->setupUi(this);
}

```

```

/*获取信号灯*/
key=ftok("./",66);
semid=semget(key,4,0666);

/*打开原文件*/
writefile=fopen("/home/lm/untitled/input.txt","rb");
if(writefile==NULL){
    printf("No File\n");
    exit(0);
}
struct stat statbuf;
stat("/home/lm/untitled/input.txt",&statbuf);
filesize=statbuf.st_size;    //获得要读取的文件的大小
printf("filesize:%d\n",filesize);

/*获取共享内存*/

key=ftok("./",68);
shmid=shmget(key,sizeof(BUF),0666);
bufdata=(BUF*)shmat(shmid,0,0);

/*start 后每秒调用一次 GetBuf*/
QTimer *timer = new QTimer(this);
connect(timer,SIGNAL(timeout()),this,SLOT(GetBuf()));
timer->start(TIME);

}

void Get::GetBuf(){
    printf("get flag:%d\n",flag);
    if(flag==0){        //确保在读完最后一个缓冲区之后不再继续读
        P(semid,1);
        printf("Get %d\n",count);
        memset(bufdata->buf,'\0',LENGTH);    //清零，防止后面数据不完全覆盖导致 len 有问题
        bufdata->size=bufdata->state=0;
        count++;
        QString getstr="Get: ";
        ui->label->setText(getstr.append(QString::number(count)));
        int len=fread(bufdata->buf,sizeof(char),LENGTH,writefile);
        filesize -=len;
        printf("len %d, filesize: %d\n",len,filesize);
        if(filesize==0){
            //当在读完当前缓冲区内容后文件剩余大小为 0，则当前缓冲区是最后一个被读的

```

缓冲区



```

        bufdata->state=1;
        bufdata->size=len;
        flag=1;
        fclose(writefile);
        V(semid,0);
    }
    else{
        bufdata->size=LENGTH;
        bufdata->state=0;

        V(semid,0);
    }
}
}

```

```

Get::~~Get(){
    if(ui)
        delete ui;
}

```

❖ copy.h

```

#ifndef COPY_H
#define COPY_H

#include "main.h"
#include <QWidget>
QT_BEGIN_NAMESPACE
namespace Ui { class Copy; } //用于描述界面组件
QT_END_NAMESPACE
class Copy: public QWidget //实现一个窗体类
{
    Q_OBJECT

public:
    explicit Copy(QWidget *parent = nullptr);
    ~Copy();

public slots:
    void CopyBuf(); //获取缓冲区内容函数

private:
    Ui::Copy *ui; //指针 ui 是指向可视化设计的界面，后面会看到要访问界面上的
    //在定义实现拷贝进程的成员变量
    int count=0;
}

```

的组件，都需要通过这个指针 ui。

```

int flag=0;
int shmids=0; //在初始化时使用方括号清零，就可以不用用到 memset 函数
int shmids_t=0;
int semid;
key_t key,key_s,key_tt;
BUF* bufdata_s={0};
BUF* bufdata_t={0};
};

```

```

#endif // COPY_H

```

❖ copy.cpp

```

#include "copy.h"
#include "ui_copy.h"
#include <QTimer>
//类的成员函数
Copy::Copy(QWidget *parent):
    QWidget(parent),
    ui(new Ui::Copy)
{
    ui->setupUi(this);
    ui->setupUi(this);

    /*获取信号灯*/
    key=ftok("./",66);
    semid=semget(key,4,0666);

    /*获取共享内存*/
    key_s=ftok("./",68);
    key_tt=ftok("./",68);
    shmids=shmget(key_s,sizeof (BUF),0666);
    shmids_t=shmget(key_tt,sizeof (BUF),0666);
    bufdata_s=(BUF*)shmat(shmids,0,0);
    bufdata_t=(BUF*)shmat(shmids_t,0,0);

    /*start 后每秒调用一次 CopyBuf*/
    QTimer *timer = new QTimer(this);
    connect(timer,SIGNAL(timeout()),this,SLOT(CopyBuf()));
    timer->start(TIME);
}

void Copy::CopyBuf(){
    if(flag==0){
        P(semid,0);    //sfull -1
        P(semid,3);    //tempty -1
    }
}

```

```

        count++;
        QString getstr="Copy: ";
        ui->label->setText(getstr.append(QString::number(count)));
        memcpy(bufdata_t->buf,bufdata_s->buf,bufdata_s->size);//将缓冲区 s 中的内容复制到缓
缓冲区 t 中

```

```

        bufdata_t->size=bufdata_s->size;
        bufdata_t->state=bufdata_s->state;
        if(bufdata_t->state==1) //判断结束
            flag=1;
        V(semid,1);    //sempty +1
        V(semid,2);    //tfull +1
    }
}

```

```

Copy::~Copy(){
    if(ui)
        delete ui;
}

```

❖ put.h

```

#ifndef PUT_H
#define PUT_H
#include "main.h"
#include <QWidget>
QT_BEGIN_NAMESPACE
namespace Ui { class Put; } //用于描述界面组件
QT_END_NAMESPACE
class Put: public QWidget    //实现一个窗体类
{
    Q_OBJECT

```

public:

```

    explicit Put(QWidget *parent = nullptr);
    ~Put();

```

public slots:

```

    void PutBuf(); //获取缓冲区内容函数

```

private:

```

    Ui::Put *ui; //指针 ui 是指向可视化设计的界面，后面会看到要访问界面上的

```

组件，都需要通过这个指针 ui。

```

    //在定义实现誊抄进程的成员变量

```

```

    int count=0;

```

```

    int flag=0;

```

```

    FILE* readfile;

```

```

int filesize;
int shmid=0; //在初始化时使用方括号清零，就可以不用用到 memset 函数
int semid;
key_t key;
BUF* bufdata={0};
};

```

```

#endif // PUT_H

```

❖ put.cpp

```

#include "put.h"
#include "ui_put.h"
#include<QTimer>
Put::Put(QWidget *parent):
    QWidget(parent),
    ui(new Ui::Put)
{
    ui->setupUi(this);
    ui->setupUi(this);
    /*获取信号灯*/
    key=ftok("./",66);
    semid=semget(key,4,0666);

    /*打开原文件*/
    readfile=fopen("/home/lm/untitled/output.txt","wb");
    if(readfile==NULL){
        printf("Open Output.txt failed\n");
        exit(0);
    }

    /*获取共享内存*/
    key=ftok("./",68); //应该是 t 缓冲区对应的文件
    shmid=shmget(key,sizeof (BUF),0666);
    bufdata=(BUF*)shmat(shmid,0,0);

    /*start 后每秒调用一次 GetBuf*/
    QTimer *timer = new QTimer(this);
    connect(timer,SIGNAL(timeout()),this,SLOT(PutBuf()));
    timer->start(TIME);
}

void Put::PutBuf(){
    if(flag==0){
        P(semid,2);
    }
}

```

```

        count++;
        QString putstr = "PUT: ";
        ui->label->setText(putstr.append(QString::number(count)));
        fwrite(bufdata->buf,sizeof(char),bufdata->size,readfile);
        if(bufdata->state==1){
            fclose(readfile);
            flag=1;
        }
        V(semid,3);
    }
}

```

```

Put::~Put(){
    if(ui)
        delete ui;
}

```

❖ mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H

```

❖ mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)

```

```
        , ui(new Ui::MainWindow)
    {
        ui->setupUi(this);
    }

MainWindow::~MainWindow()
{
    delete ui;
}
```

## 2 实验二 添加系统功能调用

### 2.1 实验目的

通过本次实验掌握系统调用的过程，了解 Linux 编译内核的方法，了解并掌握添加系统调用的方法及其使用。

### 2.2 实验内容

- (1) 内核编译、生成，用新内核启动；
- (2) 新增系统调用实现：文件拷贝或 P、V 操作。

### 2.3 实验设计

#### 2.3.1 开发环境

原内核：Linux version 5.8.0-45-generic

新内核：Linux 4.4.258

编译器：gcc 9.3.0

操作系统版本：Ubuntu 20.04

#### 2.3.2 实验设计

本实验在新安装的内核下，利用自己写入的系统调用函数实现文件拷贝。首先需要在原内核下安装新内核 Linux-4.4.258。然后编写实现文件拷贝的内核函数，并将其函数和声明以及自定义的系统调用号写入到对应的文件中。最后重启 Ubuntu，选择新内核。在新内核下，编写一个测试文件测试系统功能调用的正确性。

## 2.4 实验调试

### 2.4.1 实验步骤

#### 1. 安装新内核 Linux-4.4.258

- (1) 在本机上下载 Linux-4.4.258.tar.gz。下载完毕后将其拖至虚拟机桌面上。  
使用 `sudo mv /home/lm/桌面/linux-4.4.258.tar.gz /usr/src/` 命令，将其放到 /usr/src/ 文件夹中。
- (2) 在 /usr/src 目录下，使用 `tar -zxvf /usr/src/linux-4.4.258.tar.gz` 解压。
- (3) 预先安装库文件：`sudo apt-get install build-essential kernel-package libncurses5-dev bison flex libssl-dev`。

#### 2. 编写新系统调用函数，并将其写入相应的文件中

- (1) 执行命令 `cd /usr/src/linux-4.4.258/kernel`，然后使用 `gedit sys.c` 命令打开该 c 文件，在文本末尾添加自己的系统调用函数，保存并退出。

```
❖ asmlinkage int sys_mycall(char * source_file, char * target_file){
    int source = sys_open(source_file, O_RDONLY, 0);
    int target = sys_open(target_file, O_WRONLY|O_CREAT,
S_IRUSR|S_IWUSR);
    char buf[4096];
    mm_segment_t fs;
    fs = get_fs();
    set_fs(get_ds());
    int i;
    if(source>0 && target >0){
        while(i){
            i = sys_read(source, buf, 4096);
            sys_write(target, buf, i);
        }
    }else
        printk("ERROR");
    sys_close(source);
    sys_close(target);
    set_fs(fs);
    return 0;
}
```



(2) 执行命令 `cd /usr/src/linux-4.4.258/include/linux`, 使用 `gedit syscalls.h`, 打开并在末尾添加自己的函数的声明 `asmlinkage int sys_mycall(char * source_file, char * target_file);`, 保存并退出。

(3) 执行命令 `cd /usr/src/linux-4.4.258/arch/x86/entry/syscalls`, 使用 `gedit syscall_64.tbl` 打开该文件在末尾添加自己的系统调用号: `333 64 mycall sys_mycall`); 保存并退出。

### 3. 重新编译内核

(1) 打开新内核所在文件: `cd /usr/src/linux-4.4.258`

(2) 清楚之前的编译 (如果是第一次执行不需要此步骤): `make clean`

(3) `make menuconfig`, 当弹出界面后可直接 `Exit`, 也可以进入 `SetUp` 修改 `localversion` 添加名字: `mykernel`, 然后 `Save` 再 `Exit`。

(4) 创建镜像: `make bzImage -j4`

(5) 编译模块: `make modules -j4`

(6) 安装模块: `make modules_install -j4`

(7) 安装: `make install`

### 4. 修改 grub 文件

执行指令 `sudo update-grub`, 然后使用 `gedit grub` 命令, 修改 `GRUB_TIMEOUT=10`, 使得再重启后可以通过按下 `Shift` 键选择内核版本。

### 5. 编写测试程序

❖ `test.c`

```
#include <stdio.h>
```

```
#include <linux/kernel.h>
```

```
#include <sys/syscall.h>
```

```
#include <unistd.h>
```

```
int main(int argc, char * argv[]){
```

```
    if(argc != 3){
```

```
        printf("parameter is not right.\n");
```

```
        return 0;
```

```
    }
```

```
    printf("Success to copy, and the code is %ld\n", syscall(335, argv[1],  
argv[2]));
```

```
    return 0;
```

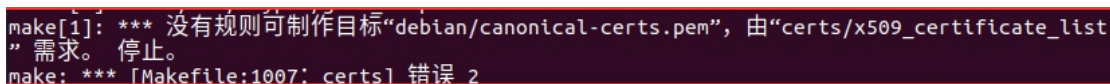
```
}
```

## 2.4.2 实验调试及心得

本次实验虽然没有什么逻辑上的难度，但是过程却十分繁琐，且很容易出错。最好的实现是安装好了新内核之后，不添加新系统调用直接编译一次看所安装的新内核版本是否符合预期，然后再重新添加新的系统调用并重新编译安装。编译安装非常耗时，我的虚拟机性能较好，全程下来顺利的话大约需要一个小时，所以每一步都需要谨慎。在本次实验中我的新内核版本和旧内核版本相差有点大，导致我的新内核启动后，会卡在开机的图形界面，所以最后新内核都是在文字版上完成的。

在本次实验编译构成中，遇到很多报错，需要针对报错信息进行修改。比如执行 `make bzImage -j4` 时报出如图 2.1 的错误。

查阅资料后，得到解决方案：在 `Linux-4.4.258` 目录下打开 `.config` 文件，修改 `CONFIG_SYSTEM_TRUSTED_KEYS="debian/canonical-certs.pem"` 为 `CONFIG_SYSTEM_TRUSTED_KEYS=""`



```
make[1]: *** 没有规则可制作目标“debian/canonical-certs.pem”，由“certs/x509_certificate_list”需求。 停止。
make: *** [Makefile:1007: certs] 错误 2
```

图 2-1 编译报错

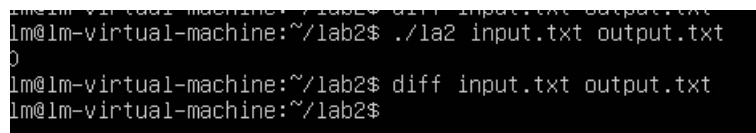
实验最后实现结果截图如 2-2 所示



```
GNU GRUB version 2.04

Ubuntu, Linux 5.8.0-45-generic
Ubuntu, with Linux 5.8.0-45-generic (recovery mode)
Ubuntu, Linux 5.4.0-58-generic
Ubuntu, with Linux 5.4.0-58-generic (recovery mode)
*Ubuntu, Linux 4.4.258myKernel
Ubuntu, with Linux 4.4.258myKernel (recovery mode)
```

图 2-2 选择新内核版本



```
lm@lm-virtual-machine:~/lab2$ ./la2 input.txt output.txt
0
lm@lm-virtual-machine:~/lab2$ diff input.txt output.txt
lm@lm-virtual-machine:~/lab2$
```

图 2-3 测试程序运行结果

由上述截图我们得知调用系统函数拷贝文件的实验结果符合预期。

## 附录 实验代码

### ❖ 内核函数

```
asm linkage int sys_mycall(char * source_file, char * target_file){
    int source = sys_open(source_file, O_RDONLY, 0);
    int target = sys_open(target_file, O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR);
    char buf[4096];
    mm_segment_t fs;
    fs = get_fs();
    set_fs(get_ds());
    int i;
    if(source>0 && target >0){
        while(i){
            i = sys_read(source, buf, 4096);
            sys_write(target, buf, i);
        }
    }else
        printk("ERROR");
    sys_close(source);
    sys_close(target);
    set_fs(fs);
    return 0;
}
```

### ❖ 测试函数

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

int main(int argc, char * argv[]){
    if(argc != 3){
        printf("parameter is not right.\n");
        return 0;
    }
    printf("Success to copy, and the code is %ld\n", syscall(335, argv[1], argv[2]));

    return 0;
}
```

## 3 实验三 添加字符设备驱动

### 3.1 实验目的

通过本次实验，掌握增加设备驱动程序的方法。

### 3.2 实验内容

通过模块方法，增加一个新的字符设备驱动程序，其功能可以简单,基于内核缓冲区。

基本要求：演示实现字符设备读、写；

### 3.3 实验设计

#### 3.3.1 开发环境

内核：Linux version 5.8.0-45-generic

编译器：gcc 9.3.0

操作系统版本：Ubuntu 20.04

内存：8G

硬盘：100G

虚拟机：VMware

#### 3.3.2 实验设计

本次实验需要编写一个设备驱动程序，需要实现

(1) 编写一个自己的设备驱动函数，能够实现把数据从内核传送到硬件和从硬件读取数据，初始化和释放设备等功能。1.在设备驱动程序中有一个很重要的结构 `file_operations`，该结构的每个域都对应着一个系统调用。我们在编写新的字符设备驱动程序时，就是对该结构体中的子函数进行重新定义。为了完成数据读写，设备初始化和释放，以及最重要的注册设备获取设备的主设备号和主校设备，需要分别定义 `mydriver_open()`, `mydriver_release()`, `mydriver_write()`, `mydriver_read()`, `__init_my_init()`, `__exit_my_exit()` 函数并填充到 `file_operations` 结构体中

- (2) 编写 Makefile 文件完成编译工作。
- (3) 编写实现将设备文件生成对应设备文件名的指令 sh 文件。
- (4) 编写测试函数，测试设备驱动程序的正确性。

## 3.4 实验调试

### 3.4.1 实验步骤

#### 1. 编写字符设备驱动程序。

(1) 分别设计 `mydriver_open()`, `mydriver_release()`, `mydriver_write()`, `mydriver_read()`, `__init_my_init()`, `__exit_my_exit()` 函数。定义全局变量 `open_nr` 代表当前需要使用该设备的进程数目，在 `mydriver_open()` 函数中，如果当前 `open_nr` 值为 0，则自加一，否则该进程应该被挂起，同时输出错误警告。在 `mydriver_release()` 中，每当一个进程结束使用设备，`open_nr` 减一。`mydriver_read` 和 `mydriver_write` 函数分别使用系统的 `copy_to_user()` 和 `copy_from_user()` 函数，并借用一个 1024 字节大小的 `buf` 数组实现数据的复制，并且在读写失败时返回 -1，否则返回读取的数据大小。`__init_my_init()` 和 `__exit_my_exit()` 函数分别使用系统的 `register_chrdev()` 和 `unregister_chrdev()` 函数注册和注销设别，并在注册时动态获得系统分配的设备号。

(2) 编写好实现的函数之后，填充 `file_operations` 结构体。将自己的注册注销函数作为参数传入系统的注册注销函数中。

```
❖ struct file_operations mydriver_fops = {  
    .read=mydriver_read,  
    .write=mydriver_write,  
    .open=mydriver_open,  
    .release=mydriver_release,  
};  
  
❖ module_init(mydriver_init);  
    module_exit(mydriver_exit);
```

2. 编写 Makefile 文件。按照 PPT 提供的 Makefile 文件，只需要将 `obj-m` 的值修改为自己的调用函数名字。

```
obj-m :=mydriver5.o
```

3. 编写装载模块需要用到的指令的 sh 文件。包括：

(1) 编译：make

(2) 装载模块：insmod mydriver5.c

(3) 打开系统设备文件，查看自己的新设备的设备号：cat /proc/devices | grep mydriver5

(4) 查看后发现系统设备号为 238，为系统字符驱动文件中添加我的系统号：mknod /dev/mydriver5 c 238 0

(5) 为新的设备驱动程序赋予可读，可写，可执行：sudo chmod 777 /dev/mydriver5

4. 编写自己的测试程序，实现打开设备，通过 read()函数显示设备缓冲的原始内容，然后从键盘获取输入，通过 write()函数该输入写入到设备缓冲区中，再通过 read()函数将设备缓冲中的内容输出显示到屏幕中。

### 3.4.2 实验调试及心得

1. 编写完系统驱动函数后，运行./load.sh 文件执行编译指令。显示系统调用号：

```
root@lm-virtual-machine:/home/lm/OS_LAB3_TEST4# cat /proc/devices | grep mydriver5
238 mydriver5
```

图 3-1 新字符驱动设备设备号

2. ./test 测试设备程序

```
root@lm-virtual-machine:/home/lm/OS_LAB3_TEST4# ./test
Input the string:
1
Before innput, the message in mydriver is:
After input, the message changed to: 1
root@lm-virtual-machine:/home/lm/OS_LAB3_TEST4# ./test
Input the string:
123456789
Before innput, the message in mydriver is: 1
After input, the message changed to: 123456789
root@lm-virtual-machine:/home/lm/OS_LAB3_TEST4# ./test
Input the string:
123
Before innput, the message in mydriver is: 123456789
After input, the message changed to: 123
root@lm-virtual-machine:/home/lm/OS_LAB3_TEST4#
```

图 3-2 测试新字符系统驱动设备程序结果

由图 3-2 可以看到，该系统设备第一次执行前，内容为空；能够实现字符键盘输入显示器输出，并在设备缓冲中留下缓存；无论输入字符串的长短，缓冲数据没有错误，实验结果符合预期。

### 3. 心得

本次实验难点主要在于如何实现模块方法添加一个新的设备。在本次实验中遇到的主要问题是：

设置的缓冲区大小明明是 1024 字节，但是测试时，发现最多只能处理 8 个字节。

原因在于，我的程序中 `mydriver_read()` 函数和 `mydriver_write()` 函数读取的长度分别是 `sizeof(buf)` 和 `sizeof(buffer)`，两个的参数都是一个字符串指针，而使用 `sizeof()` 函数指针的大小为 8 字节，所以无论字符串有多大都只能最多输入 8。

事实上，这两个函数都自带了读取的长度参数 `size_t count`，可直接使用 `count` 形参，而不需要自己获取。

通过本次实验我了解了添加新的设备主要就是定义自己的设备函数并填充 `file_operations` 结构体。并且我对于如何编译安装一个新系统驱动设备的过程有了更深的理解。

## 附录 实验代码

#### ❖ 系统驱动函数--mydriver.c

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/uaccess.h>
#include <linux/version.h>
#include <linux/slab.h>

#if CONFIG_MODVERSIONS == 1
#define MODVERSIONS
#endif

#define DEVICE_NUM 0 //device number randomly

int device_num = 0; //save the device num after being created successfully
char* buffer;
int open_nr = 0; //the pid num

//function declaration
int mydriver_open(struct inode* inode, struct file* filp);
int mydriver_release(struct inode* inode, struct file* filp);
ssize_t mydriver_read(struct file* file, char __user *buf, size_t count, loff_t *f_pos);
```

```

ssize_t mydriver_write(struct file* file, const char __user* buf, size_t count, loff_t *f_pos);

//fill the entrances of struct file_operations
struct file_operations mydriver_fops = {
    .read=mydriver_read,
    .write=mydriver_write,
    .open=mydriver_open,
    .release=mydriver_release,
};

//open
int mydriver_open(struct inode* inode, struct file* filp){
    printk("\nThe main device is %d, and the slave device
is %d\n",MAJOR(inode->i_rdev),MINOR(inode->i_rdev));
    if(open_nr==0) //the first pid
    {
        open_nr++; //the num of pid which is usinf the dev
        try_module_get(THIS_MODULE);//add 1 for the current module
        return 0;
    }
    else{
        printk(KERN_ALERT"There has already another process open the devive\n");//guaqi
        return -1;
    }
}

//read
ssize_t mydriver_read(struct file* file, char __user *buf, size_t count, loff_t *f_pos){
    if(copy_to_user(buf,buffer,count))
        return -1;
    return count;
}

//write
ssize_t mydriver_write(struct file *file, const char __user *buf, size_t count, loff_t *f_pos){
    if(copy_from_user(buffer,buf,count))
        return -1;
    return count;
}

int mydriver_release(struct inode *inode, struct file* filp){
    open_nr--;
    printk("The device is released\n");
}

```



```

        module_put(THIS_MODULE);
        return 0;
    }

//init
static int __init mydriver_init(void){
    printk(KERN_ALERT "Init the device\n");
    buffer = (char *) kmalloc(1024, GFP_KERNEL);
    int result;
    result=register_chrdev(0,"mydriver5",&mydriver_fops);
    if(result<0){
        printk(KERN_WARNING "mydriver:register failed\n");
        return -1;
    }
    else{
        printk("mydriver: register successfully\n");
        device_num=result;
        return 0;
    }
}

//exit
static void __exit mydriver_exit(void){
    printk(KERN_ALERT "Unregister...\n");
    unregister_chrdev(device_num,"mydriver5");
    kfree(buffer);
    printk("Unregister successfully\n");
}

module_init (mydriver_init);
module_exit (mydriver_exit);

MODULE_LICENSE("GPL");

```

❖ 测试函数--test.c

```

#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(void){
    char in[1024],out[1024];

```

```

memset(in,0,sizeof(in));
memset(out,0,sizeof(out));
printf("Input the string:\n");
gets(in);

int fd=open("/dev/mydriver5",O_RDWR,S_IRUSR|S_IWUSR);//open the device
if(fd>0){
    read(fd,&out,sizeof(out));
    printf("Before innput, the message in mydriver is: %s\n",out);

    write(fd,in,sizeof(in));//put the input to device
    memset(out,0,sizeof(out));//must memset
    read(fd,out,sizeof(out));//get the buf from the device and print
    printf("After input, the message changed to: %s\n",out);
    sleep(1);
}
else{
    printf("Failed\n");
    return -1;
}
close(fd);
return 0;
}

```

❖ Makefile

```

ifneq ($(KERNELRELEASE),)
    #kbuild syntax.
    obj-m :=mydriver5.o
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 *.a
endif

```

❖ load.sh

```

#!/bin/bash

make
insmod mydriver5.ko
cat /proc/devices | grep mydriver5
mknod /dev/mydriver5 c 238 0
chmod 777 /dev/mydriver5

```

## 4 实验四 使用 Qt 实现系统监视器

### 4.1 实验目的

通过本次实验学会使用 Qt 实现系统监视器，了解/proc 文件的特点和使用方法。

### 4.2 实验内容

- (1) 监控系统进程运行的情况
- (2) 用图形界面实现系统资源的监控，能欧显示系统的基本信息、CPU 等

### 4.3 实验设计

#### 4.3.1 开发环境

内核：Linux version 5.8.0-45-generic

编译器：gcc 9.3.0

操作系统版本：Ubuntu 20.04

图形界面工具：Qt-5.12

#### 4.3.2 实验设计

该实验实现的图形界面被分为四个部分和一个公共的状态栏部分。

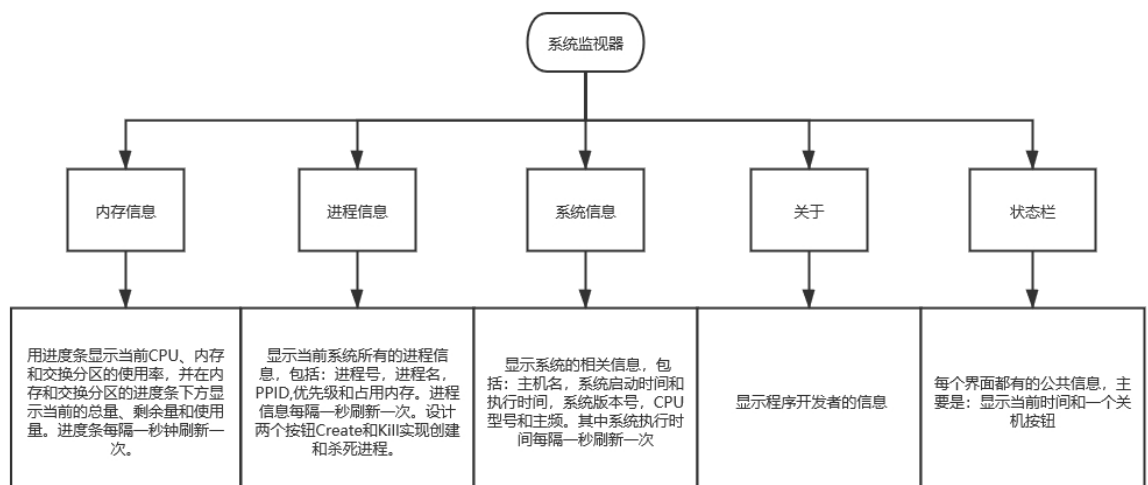


图 4-1 系统监视器总体设计图

## 4.4 实验调试

### 4.4.1 实验步骤

1. 使用 Qt 插件分别设计四个图形界面。

(1) 四个界面统一显示在一个主窗口中，使用一个 `QTabWidget` 类实现点击切换不同界面。主窗口还要设计一个 `QLabel` 用于显示当前时间，一个 `QPushButton` 实现关机按钮，显示为“Shutdown”。

(2) 系统界面需要设计三个进度条，对应的类为：`QProgressBar`。在设置若干标签 `QLabel` 用于显示当前内存和交换分区的使用量、剩余内存量和总内存量：`label_RAM_USED`, `label_RAM_FREE`, `label_RAM_TOTAL`, `label_SWAP_USED`, `label_SWAP_FREE`, `label_SWAP_TOTAL`。并插入固定的 `QLabel` 显示文字信息：“CPU”，“内存”，“交换分区”。

(3) 进程信息的界面设计显示进程信息的部分需要插入一个 `QListWidget` 实现一个列表效果的窗口用于显示多个进程的信息，同时添加两个 `QPushButton`，并设计显示为 Create 和 Kill。

(4) 系统信息的界面只需要用到标签 `QLabel`，一部分是固定的文字标签显示提示信息：“主机名”，“系统启动时间”，“系统执行时间”，“CPU 型号”，“CPU 主频”，然后在对应的文字标签后面设置 `QLabel` 显示具体信息（显示的情况根据程序给出）。

(5) 关于界面也是全为 `QLabel`，并都为固定的文字，实现固定为自己的信息：姓名，邮箱，学校，联系电话。

2. 实现获取四个界面相关信息的代码，并封装在函数 `show_tab_INFO(int index)` 中。

(1) 根据传入的参数 `index` 得知当前正在处于哪个界面。

(2) 如果 `index==0`，代表为系统信息界面。

❖ 打开文件“/proc/meminfo”文件，通过查看本地文件我们得知每一项信息都是以行为单位显示列出，我们需要通过循环读取一行信息和比较，获取我们需要的信息，并使用 `mid()` 和 `trimmed()` 函数获得信息并规范格式。在这里我们需要用到的信息有：`MemTotal`, `MemFree`, `SwapTotal`, `SwapFree`。获得这些信息后可以根据总量减去剩余量得到使用量，利用使用量\*100/总量得到内

存和交换分区的使用率。

- ❖ 打开文件“/proc/stat”，在该文件中，给出了 CPU,CPU0, CPU1, CPU2 的时间使用信息，每一项时间以空格隔开，有效项为前 7 项，每一个 CPU 编号之间以回车换行隔开。第一行 CPU 给出的是所有 CPU 对应时间之和。CPU 的使用率为（CPU 总时间-空闲时间）/CPU 总时间，此处的总的空闲时间对应第一行的第四项 idle--从系统启动开始累积到当前时刻除 IO 等待时间以外其他等待时间。打开文件后，只需要执行一次读取行的操作，使用 section()函数获取每一项时间的信息，循环七次累加得到 CPU 总时间，而在第 4 次循环得到空闲时间，再利用公式可以得到 CPU 的使用率。

- ❖ 最后再将上面得到的数据填充到对应的图形界面控件中。

(3) 如果 index==1，代表为进程信息界面。进程信息对应的文件与其进程号有关，一个进程对应一个在 /proc 下有一个进程文件，在该进程文件下的 /stat 文件存储了进程的有关信息。

- ❖ 首先创建一个 QListWidget 对象 title，并设置列表头信息
- ❖ 打开 /proc 目录，获取目录项的信息，每一个目录项对应一个进程项，使用 mid()函数由目录项获得进程号。再以“/proc/进程号/stat”的方式打开进程信息文件。
- ❖ 在该文件中我们需要获取的信息和对应的行分别为:进程名——第 1 行，父进程号 PPID——第 4 行，动态优先级——第 18 行，内存空间大小——第 23 行。
- ❖ 创建 QListWidget 对象 item，将进程号 PID，进程名，父进程号，优先级和内存以和列表头信息相同的格式写入列表中。
- ❖ 对创建进程，可以使用 Qt 自带的函数完成。首先新建一个 QProcess 的类 pro，然后设置一个对话框 QInputDialog 使用其 getText 函数获取输入要创建的进程名，最后使用 pro->start(输入的进程名)即可直接创建一个进程。
- ❖ 对于杀死进程，使用 shell 命令将指定的进程杀死。首先设置一个对话框 QInputDialog 并用其 getText()函数获得要杀死的进程的进程号。然后打开这个进程号指定的进程文件，依次获取该进程的信息，再设置一个消息确认对话框 QMessageBox，对话框的主体信息就是获得的进程信息，并设置“Yes”，“No”，“Cancel”三个按钮，只有当调用该对话框的 question()函数返回的是“Yes”选择后，才使用 system(命令)的函数执行“kill -9 %1”的强制杀死

进程的命令。

(4) 如果 `idnex==2` 代表为系统信息。

- ❖ 打开“`/proc/sys/kernel/hostname`”文件获取主机名
- ❖ 打开“`/proc/sys/kernel/ostype`”文件，读取第一行获取操作系统内核名“Linux”，打开“`/proc/sys/kernel/osrelease`”文件读取第一行的信息获得内核版本号，将两者拼接得到系统版本
- ❖ 打开文件“`/proc/cpuinfo`”，通过循环读取每一行，寻找比较得到需要的信息，包括：“`model name`”，“`cpu MHz`”。
- ❖ 调用函数 `get_boot_time()`，获得系统执行时间。在该函数中，使用系统定义的结构体 `struct sysinfo`，利用其中的成员变量 `uptime` 获得启动到现在经过的时间，即系统启动时间，再利用 `time()` 获得当前的系统时间，两者相减就可以得到系统的执行时间。规定执行时间只显示小时、分钟和秒
- ❖ 将上面得到的数据填充到对应的图形控件

### 3. 公共状态栏显示函数

- (1) 公共栏需要显示当前的时间，对应函数 `show_common_timeInfo()`。  
该函数的实现可以使用 Qt 自带的类 `QTime` 实现，时间的年月日，时分秒均存储在该类的 `currentDateTime()` 函数中，对应使用“`yyyy`”，“`M`”，“`d`”，“`h`”，“`m`”，“`s`”格式输出就可以分别得到。
- (2) 关机按钮和杀死进程类似，使用 shell 命令完成，指令为“`shutdown -h 1`”，执行该命令后系统将会在 1 分钟后自动关机。

### 4. 使用 `connect()` 函数触发各种函数

- (1) 用信号 `timeouter()` 触发公共状态栏显示的函数 `show_common_timeInfo()`
- (2) 用信号 `timeouter()` 触发界面刷新函数 `timer_update_currentTabInfo()`，在该函数中，当前的页面号 `index`，调用 `show_tab_Info()` 函数刷新界面，需要刷新的界面包括：系统信息，进程信息和内存信息界面。
- (3) 点击按钮，使用 `clicked()` 信号触发对应的按钮函数：创建、杀死进程，关机。

## 4.4.2 实验调试及心得

### 1. 实验运行结果如下：

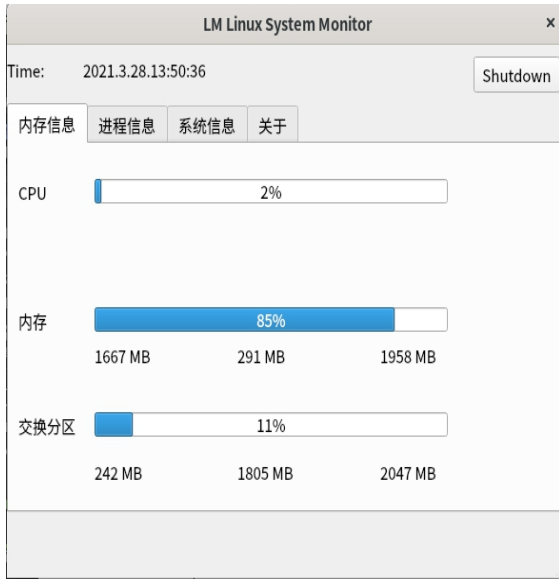


图 4-2 内存信息界面



图 4-3 进程信息界面

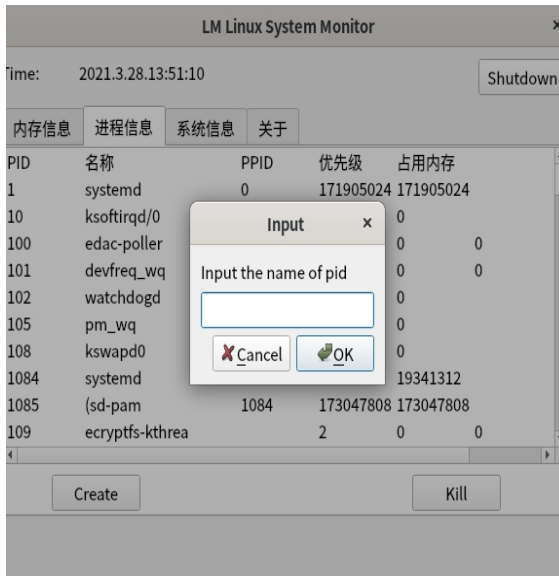


图 4-4 创建进程



图 4-5 杀死进程

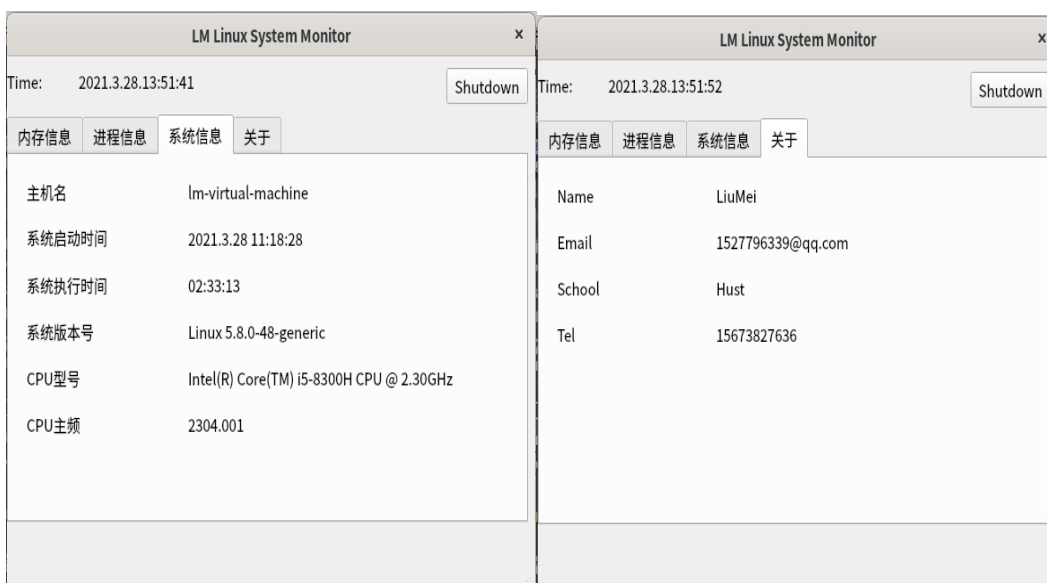


图 4-6 系统信息

图 4-7 开发者信息

## 2. 心得

本次实验最终实现了以下功能：

- (1) 显示主机名
- (2) 显示系统启动时间
- (3) 显示系统到目前为止持续运行的时间；
- (4) 显示系统版本号；
- (5) 显示 CPU 型号和主频大小；
- (6) 通过 pid 查询一个进程并显示该进程的详细信息，在弹出的对话框中点击 Yes 杀死该进程，点击其他就取消；
- (7) 显示系统所有进程的一些信息：pid，ppid，占用内存大小优先级，进程名；
- (8) cpu 使用率使用率的进度条显示；
- (9) 内存和交换分区使用率的进度条显示
- (10) 数字显示当前内存和交换分区的内存使用具体情况；
- (11) 显示当前时间和；
- (12) 通过输入进程名创建一个新的进程；
- (13) 实现了关机功能。

由于 cpu 使用率和内存使用情况已经在界面中详细给出，就不在重复将其放在状态栏中显示。

本次实验其实思维上难度不大，主要是考察对 Qt 的使用，一方面是 Qt 自带的函数和类，另一方面就是图像界面的设计，要能够了解常用的图形控件的作用。通过本次实验我对 Qt 实现较为复杂的窗口界面有了深刻的理解和体验。



## 附录 实验代码

```
❖ mainwindow.h

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
//定义资源和事件响应函数
#include <QMainWindow>
#include <QTimer>
#include <QFile>
#include <QDir>
#include <QMessageBox>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    QTimer* timer;//计时器
    //int index;

private slots:
    void show_tab_Info(int index);//显示窗口信息
    //void show_tab3Info();//显示开发者信息
    void show_comm_timeInfo();//状态栏显示当前时间
    void timer_update_currentTabInfo();//更新状态信息
    void create_pid();//创建进程
    void kill_pid();//杀死进程
    void shutdown();//按键关机
    void on_tabWidget_INFO_currentChanged();//点击 tab 界面触发调用对应的窗口函数
    void get_boot_time();//获得系统运行时间，因为需要刷新所以单独作为函数
};

#endif // MAINWINDOW_H

❖ mainwindow.cpp

#include "mainwindow.h"
#include "ui_mainwindow.h"
```

```

#include <QListWidget>
#include <QListWidgetItem>
#include <QStringList>
#include <QDateTime>
#include <QProcess>
#include <QInputDialog>
#include <sys/sysinfo.h>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    timer=new QTimer(this);
    connect(timer,SIGNAL(timeout()),this,SLOT(show_comm_timeInfo()));//定期更新系统时间
    connect(timer,SIGNAL(timeout()),this,SLOT(timer_update_currentTabInfo()));

connect(ui->tabWidget_INFO,SIGNAL(currentChanged()),this,SLOT(on_tabWidget_INFO_currentChange
d()));//选择不同界面触发不同函数显示对应窗口，currentChanged 监测界面切换
    connect(ui->pushButton_shutdown,SIGNAL(clicked()),this,SLOT(shutdown()));
    connect(ui->CreateButton,SIGNAL(clicked()),this,SLOT(create_pid()));
    connect(ui->KillButton,SIGNAL(clicked()),this,SLOT(kill_pid()));
    timer->start(1000);
}

void MainWindow::timer_update_currentTabInfo(){
    //更新信息
    int index=ui->tabWidget_INFO->currentIndex();
    //printf("index:%d\n",index);
    //定时器刷新内存 tab 页面，用于进度条动态显示以及系统信息页面需要刷新运行时长
    if(index==0||index==2||index==1)
        show_tab_Info(index);
}

void MainWindow::show_comm_timeInfo(){
    //状态栏显示当前时间
    QDateTime time;
    ui->label_time->setText(time.currentDateTime().toString("yyyy")+":"+\\

time.currentDateTime().toString("M")+":"+\\

time.currentDateTime().toString("d")+":"+\\

time.currentDateTime().toString("h")+":"+\\

```

```

time.currentDateTime().toString("m")+":"+s+"\n");

time.currentDateTime().toString("s"));
}

void MainWindow::show_tab_Info(int index){
    QFile tmpfile;
    QString tmpstr;
    int pos;
    if(index==0){
        //内存信息
        tmpfile.setFileName("/proc/meminfo");
        if(!tmpfile.open(QIODevice::ReadOnly)){
            QMessageBox::warning(this,tr("warning"),tr("
            open
            /proc/meminfo
            error"),QMessageBox::Yes);
            return ;
        }
        QString memTotal,memFree,memUsed,swapTotal,swapFree,swapUsed;//内存/交换分区总/
        生于/使用
        int nmt,nmf,nmu,nst,nsf,nsu;

        while(1){
            tmpstr=tmpfile.readLine();//读取一行的信息
            pos=tmpstr.indexOf("MemTotal");//查找位置
            if(pos!=-1){
                //打开文件，发现格式如 MemTotal:      2005388 kB
                //查找大小所在位置，trimmed 删除多余空格，转换为数字并且转换为 MB 单
                位
                memTotal=tmpstr.mid(pos+10,tmpstr.length()-13).trimmed();//-13 去掉 “ kB” 三
                个字符
                nmt=memTotal.toInt()/1024;

            }
            else if(pos=tmpstr.indexOf("MemFree"),pos!=-1){
                memFree=tmpstr.mid(pos+9,tmpstr.length()-12).trimmed();
                nmf=memFree.toInt()/1024;
            }
            else if(pos=tmpstr.indexOf("SwapTotal"),pos!=-1){
                swapTotal=tmpstr.mid(pos+11,tmpstr.length()-14).trimmed();
                nst=swapTotal.toInt()/1024;
            }
            else if(pos=tmpstr.indexOf("SwapFree"),pos!=-1){
                swapFree=tmpstr.mid(pos+10,tmpstr.length()-13).trimmed();
                nsf=swapFree.toInt()/1024;
                break;//文件中 swapfree 信息是最后的，所以循环从这里退出
            }
        }
    }
}

```

```

    }
}

//注意该文件中并没有显示已使用的信息，需要自己计算
nmu=nmt-nmf;
nsu=nst-nsf;
memUsed=QString::number(nmu,10);//转换为 QString 用于 ui 显示（十进制）
swapUsed=QString::number(nsu,10);
memTotal=QString::number(nmt,10);
memFree=QString::number(nmf,10);
swapTotal=QString::number(nst,10);
swapFree=QString::number(nsf,10);

ui->label_RAM_Used->setText(memUsed+" MB");
ui->label_RAM_Free->setText(memFree+" MB");
ui->label_RAM_Total->setText(memTotal+" MB");
ui->label_SWAP_Used->setText(swapUsed+" MB");
ui->label_SWAP_Free->setText(swapFree+" MB");
ui->label_SWAP_Total->setText(swapTotal+" MB");

ui->progressBar_RAM->setValue(nmu*100/nmt);//使用率
ui->progressBar_SWAP->setValue(nsu*100/nst);
tmpfile.close();

//stat 文件中，CPU，CPU0，CPU1，CPU2，CPU3 每行参数的解释为：
// user 从系统启动开始累计到当前时刻，用户态的 CPU 时间，不包含 nice 值为负进程
// 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 等待时间以外其他等待时间
// 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
//steal(通常缩写为 st)，代表当系统运行在虚拟机中的时候，被其他虚拟机占用的 CPU
时间

//guest(通常缩写为 guest)，代表通过虚拟化运行其他操作系统的时间，也就是运行虚
拟机的 CPU 时间

//guest_nice(通常缩写为 gnice)，代表以低优先级运行虚拟机的时间
// CPU 时间等于以上时间之和
// intr 给出中断信息
//第一行 CPU 是所有有后缀 CPU 时间之和
//CPU 的使用率=1-空闲时间(idle)/CPU 总时间
int cputotal=0,cpufree=0;//CPU 总时间和空闲时间
tmpfile.setFileName("/proc/stat");
if(!tmpfile.open(QIODevice::ReadOnly)){
    QMessageBox::warning(this,tr("warning"),tr("The stat open failed"));
}

```

```

        return ;
    }
    tmpstr=tmpfile.readLine();// “cpu” 和后面的数字之间有两个空格
    for(int i=0;i<7;i++){
        int t=tmpstr.section(" ",i+2,i+2).toInt();//获得时间
        cputotal += t;
        if(i==3)
            cpufree =t;
    }
    tmpfile.close();
    ui->progressBar_CPU->setValue((cputotal-cpufree)*100/cputotal);//注意两次 CPU 总时间
    由于精度的问题可能为 0
}
else if(index==1){
    //显示进程信息
    //进程信息
    //在/proc/pid/stat 中，一行参数解释为：
    //pid 应用程序或命令的名字 任务状态 ppid 第 18，动态优先级 第 19，静态优先级
    //在/proc/pid/stam 中，按行解释为：
    //(pages 为单位),任务虚拟地址空间的大小 应用程序正在使用的物理内存大小
    ui->ListWidget_process->clear();
    QDir qd("/proc");
    QStringList qlist=qd.entryList();
    QString qs=qlist.join("\n");
    QString id_of_pro;
    bool ok;
    int find_start=3;
    int a,b,propid;
    //int totalPronum;//进程总数
    QString proName;//进程名
    QString proPpid;//父进程号
    QString proPri;//进程优先级
    QString proMem;//占用内存
    QListWidgetItem* title=new QListWidgetItem("PID\t"+QString::fromUtf8("名称")+"\t\t"
        +"\tPPID\t"
        +QString::fromUtf8("优先级")+"\t\t"
        +QString::fromUtf8("占用内存")
        ,ui->ListWidget_process );

    //循环读取进程
    while(1){
        a=qs.indexOf("\n",find_start);
        b=qs.indexOf("\n",a+1);
        find_start=b;
        id_of_pro=qs.mid(a+1,b-a-1);

```

```

        //totalPronum++;
        propid=id_of_pro.toInt(&ok,10);//获取进程号
        if(!ok)
            break;
        //打开进程号对应的进程状态文件
        tmpfile.setFileName("/proc/"+id_of_pro+"/stat");
        if(!tmpfile.open(QIODevice::ReadOnly)){
            QMessageBox::warning(this,tr("warning"),tr("The      pid      file      open
failed"),QMessageBox::Yes);
            return ;
        }
        tmpstr=tmpfile.readLine();
        a=tmpstr.indexOf("(");
        b=tmpstr.indexOf(")");//应用程序名以：（名字）的形式给出
        proName=tmpstr.mid(a+1,b-a-1);
        proName.trimmed();//删除两端的空格
        proPpid=tmpstr.section(" ",3,3);//父进程 pid
        proPri=tmpstr.section(" ",17,17);//动态优先级
        proMem=tmpstr.section(" ",22,22);//虚拟地址空间的大小？？？不是实际物理空
间的大小,23

        QListWidgetItem* item=new QListWidgetItem(id_of_pro+"\t"
                                                    +proName+"\t\t"
                                                    +proPpid+"\t"
                                                    +proMem+"\t"
                                                    +proMem,
                                                    ui->ListWidget_process);//这种表
述有时候无法和表头对齐，无语

        tmpfile.close();
    }
    //QString tmp=QString::number(totalPronum,10);
    //ui->label_pNum->setText(tmp);

}
else if(index==2){
    //系统信息
    //需要显示：主机名，系统启动时间，到目前为止运行的时间，系统版本号，CPU 型号
和主频大小
    // 分 别 对 应 文 件 ： /proc/ sys/kernel/hostname      uptime      uptime
sys/kernel/osrelease sys/kernel/osrelease  cpuinfo
    ui->ListWidget_process->clear();

    tmpfile.setFileName("/proc/sys/kernel/hostname");
    if (!tmpfile.open(QIODevice::ReadOnly) )

```

```

        {
            QMessageBox::warning(this, tr("warning"), tr("The /proc/sys/kernel/hostname open
failed!"), QMessageBox::Yes);
            return ;
        }
        QString hostname=tmpfile.readLine();//主机名
        ui->label_hostname->setText(hostname);
        tmpfile.close();

        get_boot_time();
        struct sysinfo info;
        sysinfo(&info);
        struct tm* ptm=NULLptr;
        ptm=gmtime(&info.uptime);//gmtime:转换时间格式, uptime:启动到目前经过的时间
        char buf[30];
        sprintf(buf,"%02d:%02d:%02d",ptm->tm_hour,ptm->tm_min,ptm->tm_sec);
        ui->label_runtime->setText(QString(buf));

        // /proc/sys/kernel/ostype 中记录系统名 (Linux), osrelease 中记录版本号
        tmpfile.setFileName("/proc/sys/kernel/ostype");
        if (!tmpfile.open(QIODevice::ReadOnly) )
        {
            QMessageBox::warning(this, tr("warning"), tr("The /proc/sys/kernel/ostype open
failed!"), QMessageBox::Yes);
            return ;
        }
        tmpstr=tmpfile.readLine();//第一行就是
        tmpfile.close();
        tmpfile.setFileName("/proc/sys/kernel/osrelease");
        if (!tmpfile.open(QIODevice::ReadOnly) )
        {
            QMessageBox::warning(this, tr("warning"), tr("The /proc/sys/kernel/release open
failed!"), QMessageBox::Yes);
            return ;
        }
        QString tmpstr2=tmpfile.readLine();
        tmpstr=tmpstr.trimmed();
        tmpstr.append(" ");
        tmpstr.append(tmpstr2);
        ui->label_System_type->setText(tmpstr);
        tmpfile.close();

        tmpfile.setFileName("/proc/cpuinfo");
        if (!tmpfile.open(QIODevice::ReadOnly) )

```

```

        {
            QMessageBox::warning(this, tr("warning"), tr("The /proc/cpuinfo open failed!"),
QMessageBox::Yes);
            return ;
        }
        while(1){
            //查找 CPU 型号
            tmpstr=tmpfile.readLine();
            pos=tmpstr.indexOf("model name");
            if(pos!=-1){
                QString* cpuname=new QString(tmpstr.mid(pos+13,tmpstr.length()-13));
                ui->label_CPU_type->setText(*cpuname);
            }
            else if(pos=tmpstr.indexOf("cpu MHz"),pos!=-1){
                QString* cpufreq=new QString(tmpstr.mid(pos+11,tmpstr.length()-11));
                ui->label_CPU_freq->setText(*cpufreq);
                break;
            }
        }
        tmpfile.close();
    }
}

void MainWindow::on_tabWidget_INFO_currentChanged(){
    int index=ui->tabWidget_INFO->currentIndex();
    show_tab_Info(index);
}

void MainWindow::get_boot_time(){
    struct sysinfo info;
    time_t curtime=0,boottime=0;
    struct tm* pom=NULLptr;
    sysinfo(&info);
    time(&curtime);//获得当前时间返回到 curtime，以秒为单位
    if(curtime>info.uptime)
        boottime=curtime-info.uptime;
    else
        boottime=info.uptime-curtime;
    pom=localtime(&boottime);
    char boot_time_buf[30];

    sprintf(boot_time_buf,"%d.%d.%d %02d:%02d:%02d",pom->tm_year+1900,pom->tm_mon+1,pom->tm_m
day,pom->tm_hour,pom->tm_min,pom->tm_sec);
    ui->label_boottime->setText(QString(boot_time_buf));
}

```



```

    }

    void MainWindow::create_pid(){
        //点击按钮创建进程
        QProcess* pro=new QProcess;
        QString newproc;
        bool ok;
        newproc=QInputDialog::getText(this,"Input      ", "Input      the      name      of
pid",QLineEdit::Normal,"",&ok);//设置输入对话框获取进程名
        pro->start(newproc);//start 以子进程方式打开外部程序，外部进程的父进程为主程序，随主程
序退出而退出
    }

    void MainWindow::kill_pid(){
        QFile tmpfile;
        QString tmpstr;
        //点击 kill 按键杀死一个进程
        //QProcess pro;
        QString input_pid;
        bool ok=false;
        input_pid=QInputDialog::getText(this,"Input      ", "Input      the      pid      of      process",
QLineEdit::Normal,"",&ok);//设置输入对话框获取进程名
        if(ok){
            tmpfile.setFileName("/proc/" + input_pid + "/stat");
            if(!tmpfile.open(QIODevice::ReadOnly)){
                QMessageBox::warning(this,tr("warning"),tr("The      pid      file      open
failed"),QMessageBox::Yes);
                return ;
            }
            tmpstr=tmpfile.readLine();
            int a=tmpstr.indexOf("(");
            int b=tmpstr.indexOf(")");//应用程序名以：（名字）的形式给出
            QString proName=tmpstr.mid(a+1,b-a-1);
            proName.trimmed();//删除两端的空格
            QString proPpid=tmpstr.section(" ",3,3);//父进程 pid
            QString proPri=tmpstr.section(" ",17,17);//动态优先级
            QString proMem=tmpstr.section(" ",22,22);//虚拟地址空间的大小
            tmpfile.close();
            int
result=QMessageBox::question(this,"Kill",input_pid+"\t"+proName+"\t"+proPpid+"\t"+proPri+"\t"+proMe
m+"\t",QMessageBox::Yes|QMessageBox::No |QMessageBox::Cancel,QMessageBox::NoButton);
            if(result==QMessageBox::Yes){
                QString cmd=QString("kill -9 %1").arg(input_pid);//强制杀死进程
                system(cmd.toLocal8Bit().data());
            }
        }
    }

```

```

    }
    //pro->start(newproc);
}
void MainWindow::shutdown(){
    QProcess pro;    //通过 QProcess 类来执行第三方案序
    QString cmd = QString("shutdown -h 1");//linux 下执行 1 分钟后自动关机命令
    pro.start(cmd);    //执行命令 cmd
    pro.waitForStarted();
    pro.waitForFinished();
    close();    //关闭上位机
}
MainWindow::~MainWindow()
{
    delete ui;
}

```

#### ❖ main.cpp

```

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("LM Linux System Monitor");
    w.show();
    return a.exec();
}

```

#### ❖ mainwindow.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
    <class>MainWindow</class>
    <widget class="QMainWindow" name="MainWindow">
        <property name="geometry">
            <rect>
                <x>0</x>
                <y>0</y>
                <width>589</width>
                <height>457</height>
            </rect>
        </property>
        <property name="windowTitle">

```

```

    <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
    <widget class="QTabWidget" name="tabWidget_INFO">
      <property name="geometry">
        <rect>
          <x>0</x>
          <y>50</y>
          <width>581</width>
          <height>351</height>
        </rect>
      </property>
      <property name="currentIndex">
        <number>2</number>
      </property>
      <widget class="QWidget" name="tab">
        <attribute name="title">
          <string>内存信息</string>
        </attribute>
        <widget class="QLabel" name="label_RAM_Used">
          <property name="geometry">
            <rect>
              <x>90</x>
              <y>170</y>
              <width>68</width>
              <height>23</height>
            </rect>
          </property>
          <property name="text">
            <string/>
          </property>
        </widget>
        <widget class="QProgressBar" name="progressBar_CPU">
          <property name="geometry">
            <rect>
              <x>90</x>
              <y>30</y>
              <width>371</width>
              <height>21</height>
            </rect>
          </property>
          <property name="value">
            <number>24</number>
          </property>
        </widget>
      </widget>
    </widget>
  </widget>

```

```

</widget>
<widget class="QProgressBar" name="progressBar_RAM">
  <property name="geometry">
    <rect>
      <x>90</x>
      <y>140</y>
      <width>371</width>
      <height>21</height>
    </rect>
  </property>
  <property name="value">
    <number>24</number>
  </property>
</widget>
<widget class="QProgressBar" name="progressBar_SWAP">
  <property name="geometry">
    <rect>
      <x>90</x>
      <y>230</y>
      <width>371</width>
      <height>21</height>
    </rect>
  </property>
  <property name="value">
    <number>24</number>
  </property>
</widget>
<widget class="QLabel" name="label_RAM_Free">
  <property name="geometry">
    <rect>
      <x>240</x>
      <y>170</y>
      <width>68</width>
      <height>23</height>
    </rect>
  </property>
  <property name="text">
    <string/>
  </property>
</widget>
<widget class="QLabel" name="label_RAM_Total">
  <property name="geometry">
    <rect>
      <x>390</x>

```

```

        <y>170</y>
        <width>68</width>
        <height>23</height>
    </rect>
</property>
<property name="text">
    <string/>
</property>
</widget>
<widget class="QLabel" name="label_SWAP_Used">
    <property name="geometry">
        <rect>
            <x>90</x>
            <y>270</y>
            <width>68</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string/>
    </property>
</widget>
<widget class="QLabel" name="label_SWAP_Free">
    <property name="geometry">
        <rect>
            <x>240</x>
            <y>270</y>
            <width>68</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string/>
    </property>
</widget>
<widget class="QLabel" name="label_SWAP_Total">
    <property name="geometry">
        <rect>
            <x>390</x>
            <y>270</y>
            <width>68</width>
            <height>23</height>
        </rect>
    </property>

```

```

    <property name="text">
      <string/>
    </property>
  </widget>
  <widget class="QLabel" name="label">
    <property name="geometry">
      <rect>
        <x>10</x>
        <y>30</y>
        <width>68</width>
        <height>23</height>
      </rect>
    </property>
    <property name="text">
      <string>CPU</string>
    </property>
  </widget>
  <widget class="QLabel" name="label_2">
    <property name="geometry">
      <rect>
        <x>10</x>
        <y>140</y>
        <width>68</width>
        <height>23</height>
      </rect>
    </property>
    <property name="text">
      <string>内存</string>
    </property>
  </widget>
  <widget class="QLabel" name="label_3">
    <property name="geometry">
      <rect>
        <x>10</x>
        <y>230</y>
        <width>68</width>
        <height>23</height>
      </rect>
    </property>
    <property name="text">
      <string>交换分区</string>
    </property>
  </widget>
</widget>

```

```

<widget class="QWidget" name="tab_3">
  <attribute name="title">
    <string>进程信息</string>
  </attribute>
  <widget class="QListWidget" name="ListWidget_process">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>581</width>
        <height>271</height>
      </rect>
    </property>
  </widget>
  <widget class="QPushButton" name="CreateButton">
    <property name="geometry">
      <rect>
        <x>50</x>
        <y>280</y>
        <width>91</width>
        <height>31</height>
      </rect>
    </property>
    <property name="text">
      <string>Create</string>
    </property>
  </widget>
  <widget class="QPushButton" name="KillButton">
    <property name="geometry">
      <rect>
        <x>420</x>
        <y>280</y>
        <width>91</width>
        <height>31</height>
      </rect>
    </property>
    <property name="text">
      <string>Kill</string>
    </property>
  </widget>
</widget>
<widget class="QWidget" name="tab_2">
  <attribute name="title">
    <string>系统信息</string>

```

```

</attribute>
<widget class="QLabel" name="label_4">
  <property name="geometry">
    <rect>
      <x>20</x>
      <y>20</y>
      <width>68</width>
      <height>23</height>
    </rect>
  </property>
  <property name="text">
    <string>主机名</string>
  </property>
</widget>
<widget class="QLabel" name="label_5">
  <property name="geometry">
    <rect>
      <x>20</x>
      <y>60</y>
      <width>101</width>
      <height>21</height>
    </rect>
  </property>
  <property name="text">
    <string>系统启动时间</string>
  </property>
</widget>
<widget class="QLabel" name="label_6">
  <property name="geometry">
    <rect>
      <x>20</x>
      <y>100</y>
      <width>91</width>
      <height>23</height>
    </rect>
  </property>
  <property name="text">
    <string>系统执行时间</string>
  </property>
</widget>
<widget class="QLabel" name="label_7">
  <property name="geometry">
    <rect>
      <x>20</x>

```



```

        <y>140</y>
        <width>81</width>
        <height>23</height>
    </rect>
</property>
<property name="text">
    <string>系统版本号</string>
</property>
</widget>
<widget class="QLabel" name="label_8">
    <property name="geometry">
        <rect>
            <x>20</x>
            <y>180</y>
            <width>68</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string>CPU 型号</string>
    </property>
</widget>
<widget class="QLabel" name="label_9">
    <property name="geometry">
        <rect>
            <x>20</x>
            <y>220</y>
            <width>68</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string>CPU 主频</string>
    </property>
</widget>
<widget class="QLabel" name="label_hostname">
    <property name="geometry">
        <rect>
            <x>200</x>
            <y>20</y>
            <width>301</width>
            <height>23</height>
        </rect>
    </property>

```

```

    <property name="text">
      <string/>
    </property>
  </widget>
  <widget class="QLabel" name="label_boottime">
    <property name="geometry">
      <rect>
        <x>200</x>
        <y>60</y>
        <width>331</width>
        <height>23</height>
      </rect>
    </property>
    <property name="text">
      <string/>
    </property>
  </widget>
  <widget class="QLabel" name="label_runtime">
    <property name="geometry">
      <rect>
        <x>200</x>
        <y>100</y>
        <width>331</width>
        <height>23</height>
      </rect>
    </property>
    <property name="text">
      <string/>
    </property>
  </widget>
  <widget class="QLabel" name="label_System_type">
    <property name="geometry">
      <rect>
        <x>200</x>
        <y>140</y>
        <width>341</width>
        <height>23</height>
      </rect>
    </property>
    <property name="text">
      <string/>
    </property>
  </widget>
  <widget class="QLabel" name="label_CPU_type">

```

```

<property name="geometry">
  <rect>
    <x>200</x>
    <y>180</y>
    <width>361</width>
    <height>23</height>
  </rect>
</property>
<property name="text">
  <string/>
</property>
</widget>
<widget class="QLabel" name="label_CPU_freq">
  <property name="geometry">
    <rect>
      <x>200</x>
      <y>220</y>
      <width>351</width>
      <height>23</height>
    </rect>
  </property>
  <property name="text">
    <string/>
  </property>
</widget>
</widget>
<widget class="QWidget" name="tab_4">
  <attribute name="title">
    <string>关于</string>
  </attribute>
  <widget class="QLabel" name="label_10">
    <property name="geometry">
      <rect>
        <x>20</x>
        <y>20</y>
        <width>68</width>
        <height>23</height>
      </rect>
    </property>
    <property name="text">
      <string>Name</string>
    </property>
  </widget>
  <widget class="QLabel" name="label_11">

```

```

<property name="geometry">
  <rect>
    <x>20</x>
    <y>60</y>
    <width>68</width>
    <height>23</height>
  </rect>
</property>
<property name="text">
  <string>Email</string>
</property>
</widget>
<widget class="QLabel" name="label_12">
  <property name="geometry">
    <rect>
      <x>20</x>
      <y>100</y>
      <width>68</width>
      <height>23</height>
    </rect>
  </property>
  <property name="text">
    <string>School</string>
  </property>
</widget>
<widget class="QLabel" name="label_13">
  <property name="geometry">
    <rect>
      <x>20</x>
      <y>140</y>
      <width>68</width>
      <height>23</height>
    </rect>
  </property>
  <property name="text">
    <string>Tel</string>
  </property>
</widget>
<widget class="QLabel" name="label_14">
  <property name="geometry">
    <rect>
      <x>200</x>
      <y>20</y>
      <width>68</width>

```

```

        <height>23</height>
    </rect>
</property>
<property name="text">
    <string>LiuMei</string>
</property>
</widget>
<widget class="QLabel" name="label_15">
    <property name="geometry">
        <rect>
            <x>200</x>
            <y>60</y>
            <width>151</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string>1527796339@qq.com</string>
    </property>
</widget>
<widget class="QLabel" name="label_16">
    <property name="geometry">
        <rect>
            <x>200</x>
            <y>100</y>
            <width>68</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string>Hust</string>
    </property>
</widget>
<widget class="QLabel" name="label_17">
    <property name="geometry">
        <rect>
            <x>200</x>
            <y>140</y>
            <width>101</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string>15673827636</string>
    </property>
</widget>

```

```

        </property>
    </widget>
</widget>
</widget>
<widget class="QLabel" name="label_18">
    <property name="geometry">
        <rect>
            <x>0</x>
            <y>10</y>
            <width>68</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string>Time:</string>
    </property>
</widget>
<widget class="QLabel" name="label_time">
    <property name="geometry">
        <rect>
            <x>80</x>
            <y>10</y>
            <width>261</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string/>
    </property>
</widget>
<widget class="QPushButton" name="pushButton_shutdown">
    <property name="geometry">
        <rect>
            <x>490</x>
            <y>10</y>
            <width>91</width>
            <height>31</height>
        </rect>
    </property>
    <property name="text">
        <string>Shutdown</string>
    </property>
</widget>
</widget>

```

```
<widget class="QMenuBar" name="menubar">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>589</width>
      <height>28</height>
    </rect>
  </property>
</widget>
<widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```

## 5 实验五 模拟文件系统

### 5.1 实验目的

- (1) 基于一个大文件，设计并实现一个模拟的文件系统；
- (2) 格式化，建立文件系统管理数据结构；
- (3) 基本操作，实现文件、目录相关操作

### 5.2 实验内容

- (1) 确定文件目录项的结构，空白块的管理（每个块=连续的N个文件字节）；
- (2) 扩充系统调用命令实现文件的操作(字符界面或图形界面：open、close、read、write、cp、rm等,字符界面演示；
- (3) 选择支持：多用户、树形目录（mkdir,cd...），图形界面。

### 5.3 实验设计

#### 5.3.1 开发环境

内核：Linux version 5.8.0-45-generic  
编译器：gcc 9.3.0  
操作系统版本：Ubuntu 20.04  
内存：8G  
硬盘：100G  
虚拟机：VMware

#### 5.3.2 实验设计

##### 1. 文件目录项和块的设计

在本实验中，模拟磁盘的大小为 64MB，磁盘每一个块的大小为 1KB。第 0 块放的是用户，但本实验中并没有实现多用户的功能，所以此块其实没有用到。第 1 块为超级块，采用位示图的方法管理 inode 和普通块。第 2-1025 块为 inode 结点块，之后的块就是普通的存放文件和目录内容的块。这样实现的是一片连续的空间。



## 2. 实现的指令功能

本次实验最终实现 `mkdir`, `touch`, `cd`, `cp`, `vim`, `rm`, `rmdir` 七条指令。

### (1) 创建目录/文件——`mkdir/touch`

- ① 检查当前的目录的数量是否超过了规定的最大目录数
- ② 检查待创建的目录名是否在当前目录下重名
- ③ 检查目前所有的目录是否已经占满了一个磁盘块，如果是跳转到④，否则执行⑤
- ④ 当前使用的块的数量加一，调用 `alloc_block()`函数分配一个新的块
- ⑤ 检查当前需要的块的数量是否大于剩余块的数量，如果是直接输出报错语句并返回-1.否则执行⑥
- ⑥ 分配一个 `inode` 结点
- ⑦ 如果传入的参数 `type` 为 0，代表创建目录文件，调用 `init_dir_inode()` 函数，否则代表创建普通数据文件，应该调用 `init_file_inode()`函数进行初始化。
- ⑧ 更新当前结点和当前目录的内容

### (2) 打开目录——`cd`

- ① 检查输入的目录是否存在
- ② 打开输入的目录获取其 `i` 结点信息
- ③ 修改全局变量当前 `i` 结点号值为新打开的 `i` 结点号的值
- ④ 修改终端显示的路径名

### (3) 复制文件——`cp`

- ① 检查输入的源文件是否存在，如果不存在，输出错误信息并退出，否则执行②
- ② 获得源文件对应的 `inode` 结点信息
- ③ 将源文件的所有存储在磁盘块中的内容复制到缓冲区 `my_tmp_disk` 中
- ④ 检查如果目的是复制到其他目录下，执行⑤，否则执行⑩
- ⑤ 检查目的目录是否存在，如果不存在，输出错误信息并退出，否则执行⑥
- ⑥ 获取目的目录的 `inode` 结点信息，调用 `cd_dir()`函数打开目的目录
- ⑦ 检查目的目录下是否已经存在与源文件同名的文件，如果存在，输出错误信息并退出，否则执行⑧
- ⑧ 在目的目录下创建新的目录项并保存

- ⑨ 再次转回到原来的目录下，结束
- ⑩ 在当前目录下直接创建新的目录项并保存，结束。

#### (4) 编辑文件——vim

- ① 检查输入的文件是否存在，如果不存在，输出错误信息，然后退出，否则执行②
- ② 获取要打开文件对应的 `inode` 结点的信息
- ③ 将该文件对应的磁盘内的所有内容复制到缓冲文件 `my_tmp_disk` 中
- ④ 创建一个进程，使用 `vim` 命令，进入缓冲文件中进行修改
- ⑤ 将缓冲区文件的内容重新写回到磁盘中

#### (5) 删除目录/文件——`rmdir/rm`

- ① 检查输入的合法性：不能为不存在的目录或文件，不能为“.”和“..”目录
- ② 获取对应的 `inode` 结点的信息
- ③ 如果该结点的成员变量 `type` 为 0，表示删除目录，执行④，否则执行⑤
- ④ 进入要删除的文件目录，如果在该目录下含有文件，则该文件不能删除，输出错误信息并退出。否则执行⑤
- ⑤ 调用 `free_inode()`函数回收掉对应的 `i` 结点
- ⑥ 修改当前目录数组的内容
- ⑦ 如果当前要删除的文件所在的块只有它本身，则调用 `free_block()`函数将块一起回收掉。

#### (6) 查看文件内容——`cat`

- ① 检查输入的合法性：输入文件是否存在
- ② 获取输入文件的 `inode` 结点信息
- ③ 将文件的磁盘内容复制到缓冲区文件中
- ④ 输出缓冲区文件的内容

#### (7) 显示当前目录的目录项——`ls`

- ① 循环获取当前目录数组中的每一项目录信息
- ② 按照当前信息为目录还是普通数据文件以不同格式输出显示

### 3. 数据结构体定义

#### (1) 超级块 `super_block`

```
typedef struct super_block {
    int inode_map[INODENUM]; // i 节点示意图
    int block_map[BLOCKNUM];
```

```

    int inode_free_num;//空闲 i 节点数
    int block_free_num;
}super_block;

```

超级块采用位示图的方法，i 结点号或者磁盘块号作为下标，如果对应的值为 1 代表当前 i 结点或者普通块正在被使用，否则为空闲块。同时还记录了两者的空闲数量用于判断异常。

## (2) 普通 i 结点

```

typedef struct inode {
    //普通 i 节点
    int block_used[FILEBLKMAX];//存放该文件占用的块号
    int block_used_num;
    int size;//文件大小
    time_t creat_time;//创建时间
    time_t modify_time;//最后一次修改的时间
    int type;//为 0 代表目录，为 1 代表文件
}inode;

```

普通 i 结点的定义是在实际 unix i 结点的基础上简化和修改的。该结构体使用一个连续数组存储当前文件所用到的磁盘块的块号，并设置一个变量存储该文件所占用的块的数量。其余基本信息还包括：文件创建和最后一次修改的时间，文件大小以及文件的类型。

## (3) 目录项结构

```

typedef struct directory {
    char name[NAMEMAX];
    int inode_id;//由文件名对应一个 i 节点号，由此得到文件信息
}directory;

```

目录项的结构定义比较简单，只需要知道目录项的名字以及该文件名对应的 i 结点号。该 i 结点号要么对应一个目录文件要么对应一个普通的数据文件。目录结构图如 5-1 所示。

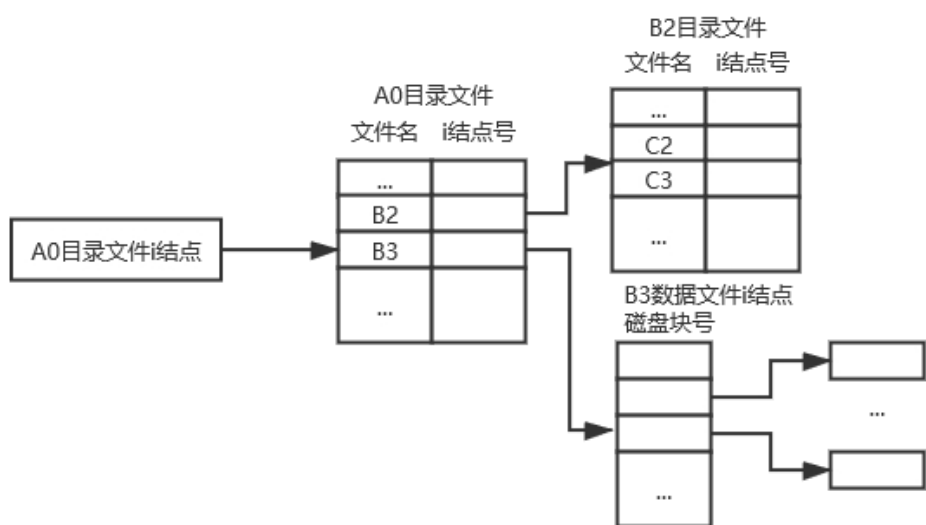


图 5-1 目录结构示意图

#### 4. 重要函数的设计

在本实验中，需要尤其注意 i 结点的分配和释放以及初值设定。

##### (1) 创建 i 结点——int alloc\_inode()

- ❖ 参数：该函数如果成功分配 i 结点将返回分配的 i 结点的结点号，否则返回-1。
- ❖ 函数设计：每次创建一个 i 结点，需要用 for 循环遍历超级块的位示图，并找到下标最小的对应的数值为 0 的空闲块。然后将该空闲块置为 1 表示已经被使用，同时超级块中的空闲 i 节点数要减一。

##### (2) 释放 i 结点——int free\_indoe(int ino)

- ❖ 参数：ino 代表要回收的 i 结点的结点号，该函数成功释放返回 1，否则返回-1
- ❖ 函数设计：i 结点的回收并不是真的要 free 结点占用的空间，而是只需要修改对应的超级块中的内容即可。考虑到释放 i 结点的实际使用，在释放时不仅要回收 i 结点空间，还要回收 i 结点对应的文件所对应的磁盘块的空间。首先通过 fseek()函数从头开始找到磁盘中获得要回收的 i 结点的位置，然后通过 fread()函数读取该 i 结点的全部信息，再通过该 i 结点的 block\_used[]数组，遍历所有使用到的磁盘块，依次调用 free\_block()函数将其回收。在回收完所有的磁盘块后，再修改超级块中位示图中下标为 ino 的值为 0 表示为空闲，同是空闲 inode 节点数加一。

(3) 创建磁盘块——`int alloc_block()`

- ❖ 参数：函数分配成功返回成功分配的磁盘块的块号，否则返回-1
- ❖ 函数设计：首先需要进行异常判断，如果当前的空闲磁盘块的数量小于等于 0，则输出“磁盘已满”的错误信息，并退出。否则，与分配 inode 结点类似，循环遍历超级块的磁盘块位示图，找到第一个值为 0 的下标，并将其值置为 1 表示已使用，然后退出循环返回这个下标值。

(4) 释放磁盘块——`int free_block(int bno)`

- ❖ 参数：bno 表示要释放的磁盘块的块号，成功释放返回 1
- ❖ 函数设计：块的释放和 inode 释放的思想相同，只需要修改超级块中的相关信息，将块号 bno 对应的位示图的值置为 0 表示空闲，且空闲的磁盘块的总数加一即表示完成释放。

5. 格式化磁盘函数——`int fmt_disk()`

磁盘在创建好之后，运行程序执行设计的指令之前需要先格式化磁盘。

该函数涉及到的操作包括：

- (1) 设置管理块结构体中各变量的初始值为初始状态值
- (2) 设置当前的 i 结点号和当前使用的 i 结点的数量为初始值
- (3) 设置当前的目录数量为 2，即“.”和“..”两个目录，表示在 root 目录下创建的两个目录。

## 5.4 实验调试

### 5.4.1 实验步骤

1. 在虚拟机终端下编译源文件 `gcc -o main main.c`
2. 使用 `dd` 指令创建一个磁盘文件
3. 执行二进制文件 `./main`
4. 进入文件系统，输入 `fmt` 格式化磁盘
5. 测试设计的指令

5.4.2 实验调试及心得

表 5-1 测试结果

指令编号	测试指令	指令功能	测试结果
1	mkdir test	创建新目录	通过
2	touch data1.txt	创建新文件	通过
3	ls	列出当前目录	通过
4	vim data1.txt	编辑文件内容	通过
5	cat data1.txt	显示文件内容	通过
6	cp data2.txt test/	复制文件到其他目录	通过
7	cd test	打开其他目录	通过
8	rm data1.txt	移除文件	通过
9	rmdir test	移除目录	通过

1. 测试 1、2、3

```
root@localhost: / >mkdir test
root@localhost: / >touch data1.txt
root@localhost: / >ls
.      ..      test  data1.txt
```

图 5-2 测试 1&2&3 结果

2. 测试 4、5

```
root@localhost: / >vim data1.txt
root@localhost: / >cat data1.txt
123
```

图 5-3 测试 4&5 结果

3. 测试 6、7

```
root@localhost: / >vim data2.txt
root@localhost: / >cp data2.txt test/
when create new directory, ls:
.      ..
after creat new directory, ls:
.      ..      data2.txt
after close_file(cur_inode_id, srcfile), ls:
.      ..      data2.txt

root@localhost: / >cd test
root@localhost: /test >ls
.      ..
root@localhost: /test >cd ..
root@localhost: / >ls
.      ..      test  data.txt
root@localhost: / >
```

图 5-4 测试 6&7 结果

4. 测试 8

```

root@localhost: / >ls
.      ..      data1.txt
root@localhost: / >rm data1.txt
root@localhost: / >ls
.      ..

```

图 5-5 测试 8 结果

## 5. 测试 9

```

root@localhost: / >mkdir test
root@localhost: / >ls
.      ..      test
root@localhost: / >rmdir test
root@localhost: / >ls
.      ..      -

```

图 5-9 测试 9 结果

由上述截图可知，本次实验实现的指令功能符合预期

## 6. 心得

本次实验是所有五个实验中最难的实验。做这个实验之前首先需要想清楚设计的结构是什么样的。之前尝试用链表来写，但是最后发现虽然可以实现指令功能但是没有涉及到块的管理所以放弃。后来参考了网上很多资料，才清楚如果要添加块的管理的话需要做哪些工作——超级块、块的初始化和释放、如何得知哪些块是空闲的等。需要提前想好整个代码的框架才能好好地实现。

由于时间比较匆忙，本次实验的代码还是存在不少的 bug 和不足之处。比如对用户的一些异常输入没法完美的进行解释，对于文件系统的指令也只是完成了部分，像 `ls -l` 指令由于我并没有在该程序中对文件进行权限管理所以就没有实现。希望能够在以后的学习中，再对这次的代码进行完善。

通过本次试验，我学到了很多知识，给老师检查时也学到了一些内容。比如本次实验的磁盘文件的创建其实是误打误撞形成的，后来老师指正应该使用 `dd` 指令创建新的磁盘文件。同时，在检验实验功能的时候，我只考虑到了终端上能够显示出来的，但是老师确还考虑到了检查磁盘数据区的正确性以及涉及到较大内存的时候，在定义的一些结构体内的成员时，要考虑到数据的对齐来尽可能减少数据结构体占用的空间。看来在这方面还是有很大的学习和提升空间。

除此以外在测试代码的时候我以为我的返回上一级目录的功能实现失败，但是检查完之后去调试发现是当时输入指令错误，检查时输入的是 `cd ../`，但实际是 `cd ..` 就可以返回上一级。

## 附录 实验代码

```
❖ filesystem.h

#pragma once

#ifndef FILESYSTEM
#define FILESYSTEM

#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <wait.h>

#define BLOCKSIZE 1024
#define INODENUM 1024
#define BLOCKNUM (1024*64)
#define FILEBLKMAX (1024*4) //文件最大为 4M
#define NAMEMAX 20 //命名最大长度
#define PWDMAX 10

#define INODESIZE sizeof(inode)
#define SUPERPOS sizeof(usr) //第 0 块为管理块，1-1024 为普通块
#define INODEPOS sizeof(super_block)
#define BLOCKPOS (INODEPOS+INODESIZE*INODENUM)//1025 i 节点之后就是 1KB 的普通块
#define DIRMAXINBLK (BLOCKSIZE / sizeof(directory)) //每一个块能存储的目录项的数目
#define DIRMAXNUM (FILEBLKMAX*DIRMAXINBLK) //每一个目录下的最大文件大小

typedef struct super_block {
    //管理块，使用位示图来管理 i 节点和空闲块
    int inode_map[INODENUM]; //i 节点位示图
    int block_map[BLOCKNUM];
    int inode_free_num; //空闲 i 节点数
    int block_free_num;
} super_block;

typedef struct inode {
    //普通 i 节点
    int block_used[FILEBLKMAX]; //存放该文件占用的块号
```



```

    int block_used_num;
    int size;//文件大小
    int mode;//文件权限
    time_t creat_time;//创建时间
    time_t modify_time;//最后一次修改的时间
    int type;//为 0 代表目录，为 1 代表文件
} inode;

typedef struct directory {
    //目录项结构
    char name[NAMEMAX];
    int inode_id;//由文件名对应一个 i 节点号，由此得到文件信息
} directory;

typedef struct usr {
    char username[NAMEMAX];
    char usrpwd[PWDMAX];
} usr;

extern FILE* disk;//使用文件模拟磁盘
extern super_block super;
extern int cur_inode_id;
extern inode cur_inode;
extern directory cur_dir_content[DIRMAXNUM];//保存当前目录下目录
extern int cur_dir_num;
extern int cur_user_id;

extern char path[128];

/*Func of Inode and Block*/
int alloc_inode();
int free_inode(int ino);
int free_block(int bno);
int init_dir_inode(int new_ino, int ino);
int init_file_node(int new_ino);
int alloc_block();

/*Func of Directory*/
int creat_dir(int ino, char* name, int type);//因为创建目录和创建文件方法相似，放在一起
int rm_dir(int ino, char* name);
int cd_dir(int ino, char* name);
int ls_dir();
int open_dir(int ino);
int close_dir(int ino);

```

```

/*Func of File*/
int cat_file();
int cp_file(int ino, char* srcfile, char* desfile);
//int mv_file(int ino, char* srcfile, char* desfile);
int open_file(int ino, char* name);
int close_file(int ino, char* name);

/*Func of disk*/
int load_super_block();//加载
//void reset_disk();
int fmt_disk();//格式化
int close_disk();//关闭

void change_path(int old_inode_id, char* name);
int parseline(const char* cmd, char** argv);
#endif // !FILESYSTEM

```

#### ❖ filesystem.c

```

#include "filesystem.h"

FILE* disk;//使用文件模拟磁盘
super_block super;
int cur_inode_id;
inode cur_inode;
directory cur_dir_content[DIRMAXNUM];
int cur_dir_num;
int cur_user_id;
char path[128];

int alloc_block() {
    //创建普通块
    int i;
    if (super.block_free_num <= 0) {
        printf("Allocte block failed\n");
        return -1;
    }
    for (i = 0; i < BLOCKNUM; i++) {
        if (super.block_map[i] == 0) {
            super.block_map[i] = 1;
            break;
        }
    }
}

```

```

        super.block_free_num--;
        return i;
    }

int free_block(int bno) {
    //释放块
    super.block_free_num++;
    super.block_map[bno] = 0;
    return 1;
}

int alloc_inode() {
    //分配 i 节点
    //返回分配的 i 节点号
    int i;
    if (super.inode_free_num <= 0) {
        printf("Allocte inode failed\n");
        return -1;
    }
    for (i = 0; i < INODENUM; i++) {
        if (super.inode_map[i] == 0) {
            //当前的 inode 为空闲
            super.inode_map[i] = 1;
            break;
        }
    }
    super.inode_free_num--;
    return i;
}

int free_inode(int ino) {
    //释放 i 节点,回收本身的 i 节点块, 同时, 回收其所指向的所有普通块
    inode node;
    fseek(disk, INODEPOS + INODESIZE * ino, 0); //找到要释放的 i 节点的位置
    fread(&node, sizeof(node), 1, disk); //查询 i 节点信息
    //释放该 i 节点指向的所有普通块
    for (int i = 0; i < node.block_used_num; i++)
        free_block(node.block_used[i]);
    //回收本 i 节点
    super.inode_map[ino] = 0;
    super.inode_free_num++;
    return 1;
}

```

```

int init_dir_inode(int new_ino, int ino) {
    //目录节点初始化, 参数: 当前目录项的 i 节点号, 父目录的 i 节点号
    inode node;
    fseek(disk, INODEPOS + INODESIZE * new_ino, 0);
    fread(&node, sizeof(inode), 1, disk);

    int bno = alloc_block();
    node.block_used[0] = bno; //需要用到一个块保存当前目录和父目录信息
    node.block_used_num = 1;
    time_t timer;
    time(&timer);
    node.creat_time = node.modify_time = timer;
    //node.mode = oct2dec(1755); //权限
    node.size = 2 * sizeof(directory);
    node.type = 0;

    //向所在的 i 节点中写入 i 节点的信息
    fseek(disk, INODEPOS + INODESIZE * new_ino, 0);
    fwrite(&node, sizeof(inode), 1, disk);

    directory dirlink[2];
    strcpy(dirlink[0].name, ".");
    dirlink[0].inode_id = new_ino;
    strcpy(dirlink[1].name, "..");
    dirlink[1].inode_id = ino;

    //向所在的块中写入信息: ...
    fseek(disk, BLOCKPOS + BLOCKSIZE * bno, 0);
    fwrite(dirlink, sizeof(directory), 2, disk);
    return 1;
}

int init_file_inode(int new_ino) {
    //初始化文件 i 节点信息, 和目录 i 节点类似, 但是没有. 和..
    inode node;
    fseek(disk, INODEPOS + INODESIZE * new_ino, 0);
    fread(&node, sizeof(inode), 1, disk);

    node.block_used_num = 0;
    time_t timer;
    time(&timer);
    node.creat_time = node.modify_time = timer;
    //node.mode = oct2dec(644);
    node.size = 0;

```

```

node.type = 1;

//向所在的 i 节点中写入 i 节点的信息
fseek(disk, INODEPOS + INODESIZE * new_ino, 0);
fwrite(&node, sizeof(inode), 1, disk);
return 1;
}

int creat_dir(int ino, char* name, int type) {
    //创建目录
    //参数: 父目录 i 结点号, 创建的目录名, 类型 (为 0, 代表创建目录, 为 1 代表创建文件)
    if (cur_dir_num >= DIRMAXNUM) {
        printf("Directory is full\n");
        return -1;
    }
    //检查要创建的目录名字是否重名
    for (int i = 0; i < cur_dir_num; i++) {
        if (strcmp(cur_dir_content[i].name, name) == 0) {
            printf("Directory Exist\n");
            return -1;
        }
    }
    //如果当前块已满, 需要开辟新的块存放
    int block_need = 1;
    if (cur_dir_num / DIRMAXINBLK != (cur_dir_num + 1) / DIRMAXINBLK)
        block_need++;
    if (block_need > super.block_free_num) {
        printf("The block num is less\n");
        return -1;
    }
    if (block_need == 2)
        cur_inode.block_used[++cur_inode.block_used_num] = alloc_block();

    int new_ino = alloc_inode();
    if (new_ino == -1) {
        printf("When create dir, alloc inode failed\n");
        return -1;
    }

    if (type == 0)
        init_dir_inode(new_ino, ino);
    else
        init_file_inode(new_ino);
}

```

```

//初始化创建的新节点
cur_dir_content[cur_dir_num].inode_id = new_ino;
strcpy(cur_dir_content[cur_dir_num].name, name);
time_t timer;
time(&timer);
cur_inode.modify_time = timer;
cur_dir_num++;
return 1;
}

int rm_dir(int ino, char* name) {
    //删除目录
    //检查输入的目录是否是当前目录.或者父目录..
    if (strcmp(name, ".") == 0 || strcmp(name, "..") == 0) {
        printf("Can't delete . and ..\n");
        return -1;
    }
    //检查输入的名字是否存在
    int i;
    for (i = 0; i < cur_dir_num; i++) {
        if (strcmp(cur_dir_content[i].name, name) == 0)
            break;
    }
    if (i == cur_dir_num) {
        printf("File Not Exist\n");
        return -1;
    }
    int rm_inode = cur_dir_content[i].inode_id; //要删除目录对应的 i 节点号
    inode node;
    fseek(disk, INODEPOS + INODESIZE * rm_inode, 0);
    fread(&node, sizeof(inode), 1, disk);

    if (node.type == 0) {
        //如果是目录
        cd_dir(ino, name); //在当前工作目录下，打开要删除的文件目录
        if (cur_dir_num != 2) { //要删除的子目录中还含有文件，不删除
            cd_dir(rm_inode, (char*)"..");
            printf("Dir has Files\n");
            return -1;
        }
        cd_dir(rm_inode, (char*)"..");
    }

    int pos;

```

```

free_inode(rm_inode);

//修改 cur_dir_content 内容，所删除位置之后的内容全部往前移
for (pos = 0; pos < cur_dir_num; pos++) {
    if (strcmp(cur_dir_content[pos].name, name) == 0)
        break;
}
for (; pos < cur_dir_num - 1; pos++)
    cur_dir_content[pos] = cur_dir_content[pos + 1];
cur_dir_num--;

if (cur_dir_num / DIRMAXINBLK != (cur_dir_num - 1) / DIRMAXINBLK) {
    //如果要删除的文件所在的块只有他一项，则连同块本身一起回收
    cur_inode.block_used_num--;
    free_block(cur_inode.block_used[cur_inode.block_used_num]);
}

//更新时间
time_t timer;
time(&timer);
cur_inode.modify_time = timer;
return 1;
}

int cd_dir(int ino, char* name)
{
    //打开目录

    //检查输入是否存在
    int i;
    for (i = 0; i < cur_dir_num; i++) {
        if (strcmp(cur_dir_content[i].name, name) == 0)
            break;
    }
    if (i == cur_dir_num) {
        printf("Dir Not Exist\n");
        return -1;
    }

    int cd_node = cur_dir_content[i].inode_id;
    inode node;
    fseek(disk, INODEPOS + cd_node * INODESIZE, SEEK_SET);
    fread(&node, sizeof(node), 1, disk);
    if (node.type != 0) {

```

```

        printf("Is Not a Dir\n");
        return -1;
    }

    close_dir(ino);
    cur_inode_id = cd_node;
    open_dir(cd_node);
    return 1;
}

int ls_dir() {
    //show
    //保存状态
    close_dir(cur_inode_id);
    open_dir(cur_inode_id);

    int i;
    inode node;
    for (i = 0; i < cur_dir_num; i++) {
        fseek(disk, INODEPOS + INODESIZE * cur_dir_content[i].inode_id, 0);
        fread(&node, sizeof(node), 1, disk);
        if (node.type == 0)
            printf("\e[1;34m%s\t\e[0m", cur_dir_content[i].name);
        else
            printf("%s\t", cur_dir_content[i].name);
    }
    printf("\n");
    return 1;
}

int open_dir(int ino) {
    //打开一个目录
    fseek(disk, INODEPOS + INODESIZE * ino, 0);
    int fd = fread(&cur_inode, sizeof(inode), 1, disk);
    if (fd != 1) {
        printf("Open inode failed\n");
        return -1;
    }

    //读取该目录下的所有目录项
    int i;
    for (i = 0; i < cur_inode.block_used_num - 1; i++) {
        fseek(disk, BLOCKPOS + BLOCKSIZE * cur_inode.block_used[i], 0);
        fread(cur_dir_content + i * DIRMIXINBLK, sizeof(directory), DIRMIXINBLK, disk);
    }
}

```



```

    }
    //最后一个块不一定是满的，需要单独列出
    int end_block_dirnum = cur_inode.size / sizeof(directory) - DIRMAXINBLK *
(cur_inode.block_used_num - 1);
    fseek(disk, BLOCKPOS + BLOCKSIZE * cur_inode.block_used[i], 0);
    fread(cur_dir_content + i * DIRMAXINBLK, sizeof(directory), end_block_dirnum, disk);

    //修改时间
    time_t timer;
    time(&timer);
    cur_inode.modify_time = timer;
    cur_dir_num = i * DIRMAXINBLK + end_block_dirnum;
    return 1;
}

int close_dir(int ino) {
    //关闭目录，并将指向的普通块的内容写回到磁盘中
    //和 open 类似，只是一个 read 一个 write
    int i;
    for (i = 0; i < cur_inode.block_used_num - 1; i++) {
        fseek(disk, BLOCKPOS + cur_inode.block_used[i] * BLOCKSIZE, SEEK_SET);
        fwrite(cur_dir_content + i * DIRMAXINBLK, sizeof(directory), DIRMAXINBLK,
disk);//fwrite
    }
    int end_block_dirnum = cur_dir_num - i * DIRMAXINBLK;
    fseek(disk, BLOCKPOS + cur_inode.block_used[i] * BLOCKSIZE, SEEK_SET);
    fwrite(cur_dir_content + i * DIRMAXINBLK, sizeof(directory), end_block_dirnum, disk);

    cur_inode.size = cur_dir_num * sizeof(directory);
    fseek(disk, INODEPOS + ino * INODESIZE, 0);
    int fd = fwrite(&cur_inode, sizeof(inode), 1, disk);
    if (fd != 1) {
        printf("Open current inode failed\n");
        return -1;
    }
    return 1;
}

/*Func of File*/
int open_file(int ino, char* name) {
    //打开文件，将其内容输出到 tmp file 中
    char block[BLOCKSIZE];
    FILE* pbuf = fopen("my_tmp_disk", "w+");

```

```

//检查要打开的文件是否存在
int i;
for (i = 0; i < cur_dir_num; i++) {
    if (strcmp(cur_dir_content[i].name, name) == 0)
        break;
}
if (i == cur_dir_num) {
    printf("File Not Exist\n");
    return -1;
}
int open_inode = cur_dir_content[i].inode_id; //找到对应的 i 结点
inode node;
fseek(disk, INODEPOS + INODESIZE * open_inode, 0);
fread(&node, sizeof(inode), 1, disk);

if (node.type == 0) {
    printf("It's a dir instead of a file\n");
    return -1;
}
if (node.size == 0) {
    //文件为空,不需要打开
    fclose(pbuf);
    return 1;
}

int bno;
//从磁盘中读取内容, 类似于 open_dir
for (i = 0; i < node.block_used_num - 1; i++) {
    memset(block, 0, BLOCKSIZE); //务必先清零
    bno = node.block_used[i]; //得到对应的块号
    fseek(disk, BLOCKPOS + BLOCKSIZE * bno, 0);
    fread(block, sizeof(char), BLOCKSIZE, disk); //从磁盘中读取块的内容
    fwrite(block, sizeof(char), BLOCKSIZE, pbuf); //将内容复制到 pbuf 缓冲中
    free_block(bno); //读取完内容后, 回收块
    node.size -= BLOCKSIZE;
}
//最后一个块, 读取实际长度
bno = node.block_used[i];
fseek(disk, BLOCKPOS + BLOCKSIZE * bno, 0);
fread(block, sizeof(char), node.size, disk); //从磁盘中读取块的内容
fwrite(block, sizeof(char), node.size, pbuf); //将内容复制到 pbuf 缓冲中
free_block(bno); //读取完内容后, 回收块
node.size = 0;
node.block_used_num = 0;

```

```

//保存结点
fseek(disk, INODEPOS + open_inode * INODESIZE, 0);
fwrite(&node, sizeof(node), 1, disk);

fclose(pbuf);
return 1;
}

int close_file(int ino, char* name) {
    //关闭文件，并将内容写回磁盘中
    char block[BLOCKSIZE];
    FILE* pbuf = fopen("my_tmp_disk", "r");

    //检查要打开的文件是否存在
    int i;
    for (i = 0; i < cur_dir_num; i++) {
        if (strcmp(cur_dir_content[i].name, name) == 0)
            break;
    }
    if (i == cur_dir_num) {
        printf("File Not Exist\n");
        return -1;
    }
    int close_inode = cur_dir_content[i].inode_id; //找到对应的 i 结点
    inode node;
    fseek(disk, INODEPOS + INODESIZE * close_inode, 0);
    fread(&node, sizeof(inode), 1, disk);

    if (node.type == 0) {
        printf("It's a dir instead of a file\n");
        return -1;
    }

    int bno;
    memset(block, 0, BLOCKSIZE);
    int readsize = fread(block, sizeof(char), BLOCKSIZE, pbuf);
    while (readsize != 0) {
        bno = alloc_block();
        if (bno == -1) {
            printf("Allocate block failed\n");
            return -1;
        }
        fseek(disk, BLOCKPOS + BLOCKSIZE * bno, 0);

```

```

        fwrite(block, sizeof(char), BLOCKSIZE, disk);
        node.block_used[node.block_used_num] = bno;
        node.size += readsize;
        node.block_used_num++;

        memset(block, 0, BLOCKSIZE);
        readsize = fread(block, sizeof(char), BLOCKSIZE, pbuf);
    }

    fseek(disk, INODEPOS + close_inode * INODESIZE, 0);
    fwrite(&node, sizeof(inode), 1, disk);
    fclose(pbuf);
    return 1;
}

int cat_file() {
    //显示文件内容
    char block[BLOCKSIZE];
    FILE* pbuf = fopen("my_tmp_disk", "r");
    memset(block, 0, BLOCKSIZE);

    //从缓冲区文件中读取内容
    int readsize;
    while ((readsize = fread(block, sizeof(char), BLOCKSIZE, pbuf) != 0))
        printf("%s", block);
    fclose(pbuf);
    return 1;
}

int cp_file(int ino, char* srcfile, char* desfile) {
    //从 srcfile 复制到 desfile
    char block[BLOCKSIZE];
    FILE* pbuf = fopen("my_tmp_disk", "w+");

    //检查文件是否存在
    int i;
    for (i = 0; i < cur_dir_num; i++) {
        if (strcmp(cur_dir_content[i].name, srcfile) == 0)
            break;
    }
    if (i == cur_dir_num) {
        printf("File Not Exist\n");
        return -1;
    }
}

```

```

int src_inode = cur_dir_content[i].inode_id;
inode node;
fseek(disk, INODEPOS + INODESIZE * src_inode, 0);
fread(&node, sizeof(inode), 1, disk);
if (node.type == 0) {
    printf("It's a dir instead of a file\n");
    return -1;
}
if (node.size == 0) {
    fclose(pbuf);
    return 1;
}
int bno;
for (i = 0; i < node.block_used_num - 1; i++) {
    memset(block, 0, BLOCKSIZE); //务必先清零
    bno = node.block_used[i];      //得到对应的块号
    fseek(disk, BLOCKPOS + BLOCKSIZE * bno, 0);
    fread(block, sizeof(char), BLOCKSIZE, disk); //从磁盘中读取块的内容
    fwrite(block, sizeof(char), BLOCKSIZE, pbuf); //将内容复制到 pbuf 缓冲中
}
//最后一个块，读取实际长度
bno = node.block_used[i];
fseek(disk, BLOCKPOS + BLOCKSIZE * bno, 0);
fread(block, sizeof(char), node.size, disk); //从磁盘中读取块的内容
fwrite(block, sizeof(char), node.size, pbuf); //将内容复制到 pbuf 缓冲中
fclose(pbuf);

if (desfile[strlen(desfile) - 1] == '/') { //复制到其他目录下,此时 desfile 是一个目录名
    desfile[strlen(desfile) - 1] = '\0';
    int despos;
    //检查要转的文件是否已经存在
    for (despos = 0; despos < cur_dir_num; despos++) {
        if (strcmp(cur_dir_content[despos].name, desfile) == 0)
            break;
    }
    if (despos == cur_dir_num) {
        printf("Desfile Not Exist\n");
        return -1;
    }
    int desnode = cur_dir_content[despos].inode_id;
    inode des_inode;
    fseek(disk, INODEPOS + INODESIZE * desnode, 0);
    fread(&des_inode, sizeof(inode), 1, disk);

```

```

if (des_inode.type == 1) {
    printf("The desfile is a file instead of a dir\n");
    return -1;
}
//进入到要复制的目录
cd_dir(cur_inode_id, desfile);
for (int i = 0; i < cur_dir_num; i++) {
    if (strcmp(cur_dir_content[i].name, srcfile) == 0) {
        //如果要粘贴的目录下存在 srcfile 文件，再次复制失败
        close_dir(cur_inode_id);
        cur_inode_id = ino; //回到原来目录
        open_dir(ino);
        printf("The des dir has had the srcfile\n");
        return -1;
    }
}
close_dir(cur_inode_id);
cur_inode_id = ino;
open_dir(ino);

//创建新的目录项
cd_dir(cur_inode_id, desfile);
printf("when create new directory, ls:\n");
ls_dir();
creat_dir(cur_inode_id, srcfile, 1);
printf("after creat new directory, ls:\n");
ls_dir();
close_file(cur_inode_id, srcfile); //保存
printf("after close_file(cur_inode_id, srcfile), ls:\n");
ls_dir();

close_dir(cur_inode_id);
cur_inode_id = ino;
open_dir(ino); //回到 src 目录下
}
else {
    //目的在当前目录下，直接创建
    for (int i = 0; i < cur_dir_num; i++) {
        if (strcmp(cur_dir_content[i].name, srcfile) == 0) {
            //如果要粘贴的目录下存在 srcfile 文件，再次复制失败
            printf("The des dir has had the srcfile\n");
            return -1;
        }
    }
}

```

```

        }
        creat_dir(ino, desfile, 1);
        close_file(ino, desfile);
    }
    return 1;
}

int init_file_node(int new_ino) {
    inode node;
    fseek(disk, INODEPOS + INODESIZE * new_ino, 0);
    fread(&node, sizeof(inode), 1, disk);

    node.block_used_num = 0;
    time_t timer;
    time(&timer);
    node.creat_time = node.modify_time = timer;
    node.size = 0;
    node.type = 1;

    fseek(disk, INODEPOS + INODESIZE * new_ino, 0);
    fwrite(&node, sizeof(inode), 1, disk);
    return 1;
}

int load_super_block() {
    //从磁盘中加载超级管理块
    fseek(disk, SUPERPOS, 0);
    int fd = fread(&super, sizeof(super_block), 1, disk);
    if (fd != 1) {
        printf("Load super block failed\n");
        return -1;
    }
    cur_inode_id = 0;
    fd = open_dir(cur_inode_id); //打开根目录
    if (fd != 1) {
        printf("Super block open failed\n");
        return -1;
    }
    return 1;
}

int fmt_disk() {
    //格式化
    memset(super.inode_map, 0, sizeof(super.inode_map));
    memset(super.block_map, 0, sizeof(super.block_map));

```

```

super.block_free_num = BLOCKNUM - 1; //除去管理块
super.inode_free_num = INODENUM - 1;
super.block_map[0] = 1;
super.inode_map[0] = 1;

cur_inode_id = 0;
fseek(disk, INODEPOS, 0);
int fd = fread(&cur_inode, sizeof(inode), 1, disk);
if (fd != 1) {
    printf("When fnt disk, open cur_inode failed\n");
    return -1;
}
cur_inode.block_used[0] = 0;
cur_inode.block_used_num = 1;
time_t timer;
time(&timer);
cur_inode.creat_time = cur_inode.modify_time = timer;
cur_inode.size = 2 * sizeof(directory);
cur_inode.type = 0;

cur_dir_num = 2;
strcpy(cur_dir_content[0].name, (char*)"");
strcpy(cur_dir_content[1].name, (char*)"..");
cur_dir_content[0].inode_id = 0;
cur_dir_content[1].inode_id = 0; //?

strcpy(path, "root@localhost: / >");
return 1;
}

int close_disk() {
    fseek(disk, SUPERPOS, 0);
    int fd = fwrite(&super, sizeof(super_block), 1, disk);
    if (fd != 1) {
        printf("Close super block failed\n");
        return -1;
    }
    //cur_inode_id = 0;
    fd = close_dir(cur_inode_id); //打开根目录
    if (fd != 1) {
        printf("Super block close failed\n");
        return -1;
    }
    return 1;
}

```



```

}

void change_path(int old_inode_id, char* name) {
    //当更换目录时，改变路径
    int pos;
    if (!strcmp(name, ".") || (!strcmp(name, "..") && (old_inode_id == 0)))
        return;
    else if (!strcmp(name, "..") && cur_inode_id != 0) {
        for (pos = strlen(path) - 1; pos >= 0; pos--) {
            if (path[pos] == '/') {
                path[pos] = '\0';
                strcat(path, ">");
                break;
            }
        }
    }
    else if (!strcmp(name, ".") && cur_inode_id == 0) {
        for (pos = strlen(path) - 1; pos >= 0; pos--) {
            if (path[pos] == '/') {
                path[pos + 1] = '\0';
                strcat(path, ">");
                break;
            }
        }
    }
    else if (path[strlen(path) - 3] == '/') {
        path[strlen(path) - 2] = '\0';
        strcat(path, name);
        strcat(path, ">");
    }
    else {
        path[strlen(path) - 2] = '\0';
        strcat(path, "/");
        strcat(path, name);
        strcat(path, ">");
    }
}

```

```

int parseline(const char* cmd, char** argv) {
    /* Holds local copy of command line */
    static char array[1024];
    char* buf = array;
    char* delim;
    int argc;

```

```

strcpy(buf, cmd);
buf[strlen(buf) - 1] = ' ';
/* Ignore leading spaces */
while (*buf && (*buf == ' '))
    buf++;

/* Build the argv list */
argc = 0;
if (*buf == '\n') {
    buf++;
    delim = strchr(buf, '\n');
}
else {
    delim = strchr(buf, ' ');
}

while (delim) {
    argv[argc++] = buf;
    *delim = '\0';
    buf = delim + 1;
    while (*buf && (*buf == ' '))
        buf++;

    if (*buf == '\n') {
        buf++;
        delim = strchr(buf, '\n');
    }
    else {
        delim = strchr(buf, ' ');
    }
}
argv[argc] = NULL;
return argc;
}

```

❖ main.c

```

#include "filesystem.h"
int execute(char* func,int argc,char** argv);
int main(void) {
    disk = fopen("my_disk", "r+");
    if (disk == NULL) {
        printf("Disk file not exist\n");
        return -1;
    }
}

```

```

    }
    char cmd[1024];
    int ans = load_super_block();
    if (ans == -1)
        return -1;
    printf("-----MY FILE SYSTEM-----\n\n");
    while (1) {
        printf("%s", path);
        fflush(stdout);
        if ((fgets(cmd, 1024, stdin) == NULL) && ferror(stdin)) {
            printf("fgets error\n");
            return -1;
        }
        if (feof(stdin)) {
            printf("Exit My File System\n");
            fflush(stdout);
            exit(0);
        }
        char* argv[128];
        int argc = parseline(cmd, argv);
        if (argv[0] == NULL)
            return -1;
        //printf("argv[0]:%s,argv[1]:%s\n",argv[0],argv[1]);
        if (!built_cmd(argv)) {
            if (!execute(argv[0], argc, argv))
                printf("command can't find\n");
        }
    }
}

int execute(char* func, int argc, char* argv[]) {
    if (strcmp(func, "mkdir") == 0) {
        if (argc != 2) {
            printf("Format error\n");
            return -1;
        }
        int ans=creat_dir(cur_inode_id, argv[1], 0);
        if (ans == -1)
            return -1;
        return 1;
    }
    else if (strcmp(func, "rmdir") == 0) {
        if (argc != 2) {
            printf("Format error\n");

```

```

        return -1;
    }
    int ans = rm_dir(cur_inode_id, argv[1]);
    if (ans == -1)
        return -1;
    return 1;
}
else if (strcmp(func, "cd") == 0) {
    if (argc != 2) {
        printf("Format error\n");
        return -1;
    }
    int old_inode = cur_inode_id;
    int ans = cd_dir(cur_inode_id, argv[1]);
    if (ans == -1)
        return -1;
    change_path(old_inode, argv[1]);
    return 1;
}
else if (strcmp(func, "ls") == 0) {
    if (argc != 1) {
        printf("Format error\n");
        return -1;
    }
    int ans = ls_dir();
    if (ans == -1)
        return -1;
    return 1;
}
else if (strcmp(func, "touch") == 0) {
    if (argc != 2) {
        printf("Format error\n");
        return -1;
    }
    int ans = creat_dir(cur_inode_id, argv[1], 1);
    if (ans == -1)
        return -1;
    return 1;
}
else if (strcmp(func, "rm") == 0) {
    if (argc != 2) {
        printf("Format error\n");
        return -1;
    }
}

```

```

        int ans=rm_dir(cur_inode_id, argv[1]);
        if (ans == -1)
            return 1;
        return 1;
    }
    else if (strcmp(func, "vim") == 0) {
        int pid, status;
        char* vim_arg[] = { "vim", "my_tmp_disk", NULL };
        int ans=open_file(cur_inode_id, argv[1]);
        if (ans == -1)
            return -1;
        if ((pid = fork()) == 0)
            execvp("vim", vim_arg);
        wait(&status);
        close_file(cur_inode_id, argv[1]);
        return 1;
    }
    else if (strcmp(func, "cat") == 0) {
        if (argc != 2) {
            printf("Format error\n");
            return -1;
        }
        int ans = open_file(cur_inode_id, argv[1]);
        if (ans == 1) {
            cat_file();
            close_file(cur_inode_id, argv[1]);
            return 1;
        }
        return -1;
    }
    else if (strcmp(func, "cp") == 0) {
        if (argc != 3) {
            printf("Format error\n");
            return -1;
        }
        int ans = cp_file(cur_inode_id, argv[1], argv[2]);
        return 1;
    }
    return 0;
}

int built_cmd(char* argv[]) {
    if (strcmp(argv[0], "fmt") == 0) {
        //if (cur_user_id == 0)

```

```
        fmt_disk();
        return 1;
    }
    else if (strcmp(argv[0], "exit") == 0) {
        close_disk();
        fclose(disk);
        exit(0);
    }
    return 0;
}
```