



# 华中科技大学

## 操作系统原理课程设计报告

姓 名： 郑 舟  
学 院： 计算机科学与技术学院  
专 业： 计算机科学与技术  
班 级： CS1906  
学 号： U201915115  
指导教师： 周正勇

分数	
教师签名	

2022 年 3 月 7 日

# 目 录

<b>1</b>	<b>实验一 Linux 用户界面的使用 .....</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	实验调试及心得.....	3
	附录 实验代码.....	4
<b>2</b>	<b>实验二 编译内核并增加系统调用 .....</b>	<b>15</b>
2.1	实验目的 .....	15
2.2	实验内容 .....	15
2.3	实验设计 .....	15
2.3.1	开发环境.....	15
2.3.2	实验设计.....	15
2.4	实验调试.....	16
2.4.1	实验步骤.....	16
2.4.2	实验调试及心得.....	18
	附录 实验代码.....	18
<b>3</b>	<b>实验三 增加字符驱动设备 .....</b>	<b>20</b>
3.1	实验目的 .....	20
3.2	实验内容 .....	20
3.3	实验设计 .....	20
3.3.1	开发环境.....	20
3.3.2	实验设计.....	20
3.4	实验调试.....	21
3.4.1	实验步骤.....	21
3.4.2	实验调试及心得.....	22
	附录 实验代码.....	22
<b>4</b>	<b>实验四 使用 QT 实现系统监控器 .....</b>	<b>26</b>
4.1	实验目的 .....	26
4.2	实验内容 .....	26
4.3	实验设计 .....	26
4.3.1	开发环境.....	26
4.3.2	实验设计.....	27
4.4	实验调试.....	28
4.4.1	实验步骤.....	28

4.4.2	实验调试及心得.....	31
附录	实验代码.....	31
<b>5</b>	<b>实验五 小型模拟文件系统 .....</b>	<b>42</b>
5.1	实验目的 .....	42
5.2	实验内容 .....	42
5.3	实验设计 .....	42
5.3.1	开发环境.....	42
5.3.2	实验设计.....	42
5.4	实验调试 .....	44
5.4.1	实验步骤.....	44
5.4.2	实验调试及心得.....	47
附录	实验代码.....	47

# 1 实验一 Linux 用户界面的使用

## 1.1 实验目的

掌握 Linux 下的用户界面的使用，以及使用 Linux 进行 C 语言编程。

## 1.2 实验内容

(1) 编写一个 C 程序，其内容为实现文件拷贝的功能。使用系统调用 open/read/write，实现 cp 的功能。

(2) 编写一个 C 程序，其内容为分窗口同时显示三个并法进程的运行结果。使用 Linux 下的图形库 gtk，实现三个简单计算进程。

## 1.3 实验设计

### 1.3.1 开发环境

操作系统：Ubuntu 20.04.3

操作系统类型：64bit

内核：Linux version 5.13.0-30-generic

内存：7.7GB

处理器：Intel® Core™ i5-8265U CPU @ 1.60GHz × 4

磁盘：85.9GB

Qt 版本：5.9.9

### 1.3.2 实验设计

#### (1) 文件拷贝

文件拷贝实验上学期已经做过了，并且编写了实验报告，本次课程设计和上学期实验的区别是本次实验不要求使用多进程和环形缓冲，但是要求使用系统调用进行文件操作而不是 C 语言库函数。方便起见，本次实验依然沿用上学期的实验三的四路，用两个进程通过共享的环形缓冲进行誊抄，基本的异常提示，比如文件无法打开或者创建等都已经实现过了，并且通过命令行传参的形式也能使

我们的程序用类似 `cp` 命令的形式运行，不过没有实现 `cp` 命令参数提供的功能。

需要在上学期实验三基础上修改的地方主要是文件的读取，需要将库函数修改为系统调用，系统调用的使用方法如下：

1. 打开或者创建文件：`int open(const char* pathname, int flags, mode_t mode);`  
第一个参数是文件路径，第二个参数指定打开文件的读写模式，第三个参数指定创建文件时文件的权限，其中第三个参数在打开仅以读模式打开文件时不用设置，函数返回文件描述符，打开失败则返回 -1。
2. 读取文件：`ssize_t read(int fd, void *buf, size_t count)`  
`fd` 为文件描述符，`buf` 为读入的缓冲区指针，`count` 指定要读入的字符数，函数返回实际读到的字节数，发生异常时返回 -1。
3. 写入文件：`ssize_t write(int fd, const void *buf, size_t count)`  
`fd` 为文件描述符，`buf` 为存有将要写入的字符的数组指针，`count` 指写入的字符数，函数返回实际写入的字节数，当有错误发生时返回 -1。

## (2) 使用窗口显示并发程序运行

由于对 Qt 比较熟悉，该实验我选择使用 Qt 完成。在父进程中 `fork` 三个子进程，每个子进程显示一个窗口，分别打印数字 1~10，小写字母 a~z，大写字母 A~Z。为了实现三个窗口的动态打印效果，打印字符的函数被设置为槽，与 1s 发生一次的 `timeout` 信号绑定，从而实现三个窗口都每秒打印一个字符。

## 1.4 实验调试

### 1.4.1 实验步骤

#### (1) 文件拷贝

在上学期实验三的基础上，将读取文件、写入文件程序的源代码中的 `fopen`、`fread`、`fwrite` 函数更换为 `open`、`read`、`write`，并修改对应参数。编译主程序、读文件程序、写文件程序，在终端中执行主程序，同时输入源文件名和目标文件名，这里使用视频文件 1.mp4，如图 1.1 所示。复制过程中会输出环形缓冲中每一个缓冲区的使用情况，如图 1.2 所示，最后使用 `cmp` 命令比较源文件和目标文件，结果如图 1.3 所示，可知程序可以正确运行。最后尝试传入一个不存在的源文件，结果如图 1.4 所示，输出了源文件不存在的提示信息，可见程序具有一定的容错能力。

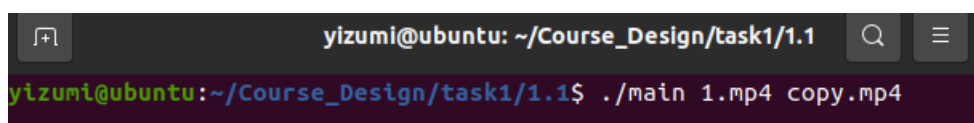


图 1.1 复制 1.mp4 到 copy.mp4

```

write runs 29339 times in the 3 shm
read runs 29341 times in the 1 shm
write runs 29340 times in the 0 shm
read runs 29342 times in the 2 shm
write runs 29341 times in the 1 shm
read runs 29343 times in the 3 shm
write runs 29342 times in the 2 shm
read runs 29344 times in the 0 shm
write runs 29343 times in the 3 shm
read finish
write runs 29344 times in the 0 shm
write finish
delete shm
delete sem
yizumi@ubuntu:~/Course_Design/task1/1.1$

```

图 1.2 缓冲区使用情况输出

```

yizumi@ubuntu:~/Course_Design/task1/1.1$ cmp 1.mp4 copy.mp4
yizumi@ubuntu:~/Course_Design/task1/1.1$

```

图 1.3 比较源文件与复制文件

```

yizumi@ubuntu:~/Course_Design/task1/1.1$ ./main asdf copy.mp4
read created

write created

open origin file failed!
delete shm
delete sem
yizumi@ubuntu:~/Course_Design/task1/1.1$

```

图 1.4 异常情况处理

(2) 使用窗口显示并发程序运行。

在 Qt 项目中新增三个窗口，每个窗口上添加一个文本框控件用来显示 pid，添加一个 textbrowser 控件用来显示打印的字符。为每个窗口类添加一个槽用来打印字符，并在构造函数中将其与 Qtime 类的一个实例的 timeout 信号绑定，同时将 timeout 信号发生的时间设置为 1000ms。在主函数中 fork 三个进程，每个进程创建一种窗口，点击运行，效果如图 1.5 所示

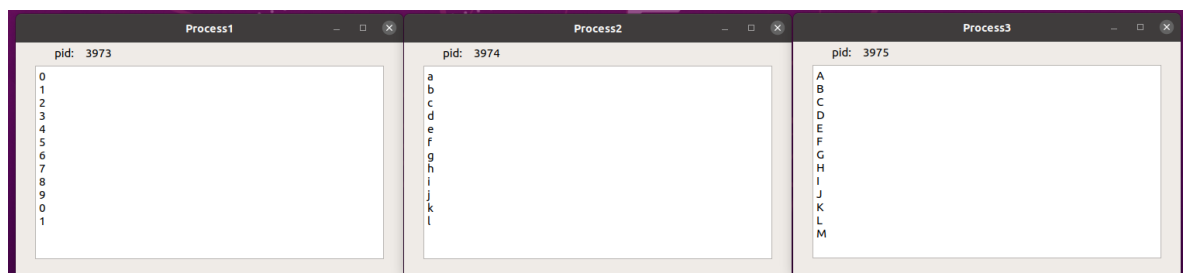


图 1.5 三个窗口显示并发进程

## 1.4.2 实验调试及心得

由于之前实现过文件的复制与誊抄，所以第一个实验比较简单，只是修改了几个函数，但是通过查找这些函数的用法，我对于 Linux 系统的系统调用函数更

熟悉了，特别是在查找 `open` 函数后面两个参数的使用方法时，这些宏之间进行位运算操作让我想起了上学期最后一个实验判断文件权限时，也是通过一系列宏的位运算实现的，这使我大致理解了 **Linux** 提供的系统调用的一些套路。

第二个实验其实是我退而求其次的结果，我本来是想做三个进程间的誊抄的，并且不带窗口显示的三进程誊抄代码我已经写出来了，基本思路是 `read` 进程将内容读到缓冲区 1，`copy` 进程将缓冲区 1 内容复制到缓冲区 2，`write` 进程将缓冲区 2 的内容复制到文件，但是当我想要将运行的提示信息显示到窗口时碰到了困难，因为在 **Qt** 里想要实现三进程誊抄，我能想到的办法是在每个窗口的构造函数中再开一个线程区执行 `read`、`copy` 和 `write`，但是这样似乎会使信号灯失效，一开始誊抄程序，对一个信号灯进行 **P** 操作之后就卡住了。由于实在解决不了这个问题，于是实现了一个简单版本。

## 附录 实验代码

实验 1.1:

main.c:

```
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include "signal.h"
#include <sys/wait.h>
#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "unistd.h"
#include "sys/sem.h"

#define SHMKEY1 11
#define SHMKEY2 22
#define SHMKEY3 33
#define SHMKEY4 44

int semid;

int shmid1, shmid2, shmid3, shmid4;

int pid_read, pid_write;

union semun{
    int val;
};
```

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



```

#include <sys/shm.h>
#include <sys/ipc.h>
#include <signal.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/sem.h>

#define SHMKEY1 11
#define SHMKEY2 22
#define SHMKEY3 33
#define SHMKEY4 44

void V(int semid, int index);
void P(int semid, int index);
union semun{
    int val;
};
int main(int argc, char *argv[])
{

    char *buf[4];
    int *len;
    int shmid1, shmid2, shmid3, shmid4;
    shmid1 = shmget(SHMKEY1, 101 * sizeof(char), 0666);
    shmid2 = shmget(SHMKEY2, 101 * sizeof(char), 0666);
    shmid3 = shmget(SHMKEY3, 101 * sizeof(char), 0666);
    shmid4 = shmget(SHMKEY4, 101 * sizeof(char), 0666);
    buf[0] = (char *)shmat(shmid1, 0, 0);
    buf[1] = (char *)shmat(shmid2, 0, 0);
    buf[2] = (char *)shmat(shmid3, 0, 0);
    buf[3] = (char *)shmat(shmid4, 0, 0);
    int semid;
    semid = semget((key_t)1, 3, IPC_CREAT | 0666);

    int file=open(argv[1],O_RDONLY);
    if (file == -1){
        printf("open origin file failed!\n");
        exit(-1);
    }
}

```

```

int i = 0;
int d = 0;
while (1)
{
    P(semid, 0);
    P(semid, 2);
    buf[i][100]=(char)read(file, buf[i], 100);
    if (buf[i][100] == 0){
        V(semid, 2);
        V(semid, 1);
        close(file);
        break;
    }
    V(semid, 2);
    V(semid, 1);
    printf("read runs %d times in the %d shm\n",d++, i);
    i = (i + 1) % 4;
}
printf("read finish\n");
exit(0);
}

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

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;
}

```

writebuf.c:

```
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <signal.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/sem.h>
#include <fcntl.h>
#define SHMKEY1 11
#define SHMKEY2 22
#define SHMKEY3 33
#define SHMKEY4 44
void V(int semid, int index);
void P(int semid, int index);
union semun{
    int val;
};
int main(int argc, char *argv[])
{
    char *buf[4];
    int *len;
    int shmid1, shmid2, shmid3, shmid4, l;
    shmid1 = shmget(SHMKEY1, 101 * sizeof(char), 0666);
    shmid2 = shmget(SHMKEY2, 101 * sizeof(char), 0666);
    shmid3 = shmget(SHMKEY3, 101 * sizeof(char), 0666);
    shmid4 = shmget(SHMKEY4, 101 * sizeof(char), 0666);
    buf[0] = (char *)shmat(shmid1, 0, 0);
    buf[1] = (char *)shmat(shmid2, 0, 0);
    buf[2] = (char *)shmat(shmid3, 0, 0);
    buf[3] = (char *)shmat(shmid4, 0, 0);
    int semid;
    semid = semget((key_t)1, 3, IPC_CREAT | 0666);
    int file=open(argv[1], O_RDWR | O_CREAT | O_TRUNC, S_IRUSR |
S_IWUSR | S_IXUSR);
    if (file == -1){
        printf("create copy file failed!\n");
        exit(-1);
    }
    int i = 0;
    int d = 0;
```

```

while (1)
{
    P(semid, 1);
    P(semid, 2);
    if (buf[i][100] == 0)
    {
        V(semid, 2);
        V(semid, 0);
        close(file);
        break;
    }
    // printf("%s",buf[i]);
    write(file,buf[i],buf[i][100]);
    V(semid, 2);
    V(semid, 0);
    printf("write runs %d times in the %d shm\n",d++, i);
    i = (i + 1) % 4;
}
printf("write finish\n");
exit(0);
}

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

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;
}

```

main.c

```
#include "process1.h"
#include "process2.h"
#include "process3.h"
#include <unistd.h>
#include <QApplication>

int main(int argc, char *argv[])
{
    int pid_1=0;
    int pid_2=0;
    int pid_3=0;
    if((pid_1==fork())==0){
        QApplication a1(argc, argv);
        process1 w1;
        w1.setWindowTitle("Process1");
        w1.show();
        a1.exec();
        exit(0);
    }
    else if((pid_2==fork())==0){
        QApplication a2(argc, argv);
        process2 w2;
        w2.setWindowTitle("Process2");
        w2.show();
        a2.exec();
        exit(0);
    }
    else if((pid_3==fork())==0){
        QApplication a3(argc, argv);
        process3 w3;
        w3.setWindowTitle("Process3");
        w3.show();
        a3.exec();
        exit(0);
    }
    return 0;
}
```

process1.h

```
#ifndef PROCESS1_H
#define PROCESS1_H
#include <QWidget>
#include <QTimer>
```

```

namespace Ui {
class process1;
}

class process1 : public QWidget
{
    Q_OBJECT

public:
    explicit process1(QWidget *parent = nullptr);
    ~process1();
public slots:
    void print();

private:
    Ui::process1 *ui;
    int num;
};
#endif // PROCESS1_H

```

process1.cpp

```

#include "process1.h"
#include "ui_process1.h"
#include<unistd.h>
process1::process1(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::process1),num(0)
{
    ui->setupUi(this);
    ui->pid->setText(QString::number(getpid()));
    QTimer *t=new QTimer(this);
    connect(t,SIGNAL(timeout()),this,SLOT(print()));
    t->start(1000);
}

void process1::print(){
    ui->textBrowser->append(QString::number(this->num));
    this->num=(this->num+1)%10;
}

process1::~process1()
{
    delete ui;
}

```

由于 ui 是通过可视化工具设计的，代码意义不大，外观前面已经贴过图了，所以就不贴 ui 代码了。

Process2.h

```
#ifndef PROCESS2_H
#define PROCESS2_H

#include <QWidget>
#include <QTimer>
namespace Ui {
class process2;
}

class process2 : public QWidget
{
    Q_OBJECT

public:
    explicit process2(QWidget *parent = nullptr);
    ~process2();
public slots:
    void print();

private:
    Ui::process2 *ui;
    int i;
};

#endif // PROCESS2_H
```

Process2.cpp

```
#include "process2.h"
#include "ui_process2.h"
#include<unistd.h>
process2::process2(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::process2),i(0)
{
    ui->setupUi(this);
    ui->pid->setText(QString::number(getpid()));
    QTimer *t=new QTimer(this);
    connect(t,SIGNAL(timeout()),this,SLOT(print()));
    t->start(1000);
}
```

```

void process2::print(){
    ui->textBrowser->append(QString('a'+i));
    i=(i+1)%26;
}

process2::~~process2()
{
    delete ui;
}

```

process3.h

```

#ifndef PROCESS3_H
#define PROCESS3_H

#include <QWidget>
#include <QTimer>
namespace Ui {
class process3;
}

class process3 : public QWidget
{
    Q_OBJECT

public:
    explicit process3(QWidget *parent = nullptr);
    ~process3();
public slots:
    void print();

private:
    Ui::process3 *ui;
    int i;
};

#endif // PROCESS3_H

```

process3.cpp

```

#include "process3.h"
#include "ui_process3.h"
#include<unistd.h>
process3::process3(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::process3),i(0)

```



```

{
    ui->setupUi(this);
    ui->pid->setText(QString::number(getpid()));
    QTimer *t=new QTimer(this);
    connect(t,SIGNAL(timeout()),this,SLOT(print()));
    t->start(1000);
}

void process3::print(){
    ui->textBrowser->append(QString('A'+i));
    i=(i+1)%26;
}

process3::~~process3()
{
    delete ui;
}

```

## 2 实验二 编译内核并增加系统调用

### 2.1 实验目的

掌握系统调用的实现过程,通过编译内核方法,增加一个新的系统调用。另编写一个应用程序,使用新增加的系统调用。

### 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

操作系统: Ubuntu 20.04.3

操作系统类型: 64bit

内核: Linux version 5.13.0-30-generic

内存: 7.7GB

处理器: Intel® Core™ i5-8265U CPU @ 1.60GHz × 4

磁盘: 85.9GB

新增系统内核: Linux version 4.14.267

#### 2.3.2 实验设计

实验二有两个主要的任务,一是编译并安装新的系统内核,二是使用编译内核的方法添加新的系统调用。所以在实验前需要知道两个任务如何操作。

编译并安装新内核的一般方式如下:

1. 下载内核源代码
2. 使用 `make menuconfig` 命令生成.config 文件
3. 使用 `make` 命令编译内核
4. 使用 `make modules_install` 和 `make install` 安装内核模块与内核

5. 重新启动，在内核载入前按 shift 键进入 grub，更换内核启动添加系统调用的一般方式如下：
  1. 在系统调用服务例程源文件中添加自定义系统调用函数
  2. 在系统调用服务例程头文件中添加函数的声明
  3. 在系统调用表中添加新的系统调用的系统调用号及其关联的应用服务例程函数名
  4. 重新编译内核并安装。

## 2.4 实验调试

### 2.4.1 实验步骤

实验步骤基本按照实验设计的进行，只需根据具体的内核版本对不同的文件进行修改，具体步骤如下：

1. 在官网下载内核，我下载的内核版本为 4.14.267。下载到本地后解压至 /usr/src 文件夹。
2. 使用实验设计中的几个命令编译并安装新内核。需要注意的是在生成.config 文件后，需要删除 CONFIG\_SYSTEM\_TRUSTED\_KEYS 的安全检查项，同时需要修改删除其中有关五级页面缓存的设置，将其注释掉并将缓存页面数设置为 4，否则之后系统会无法启动并提示 CPU 不支持五级页面缓存。在使用 make 命令编译时可以加上 -j[n] 参数指定编译线程数，加快编译过程。使用 make 命令时会提示缺少某些模块，缺什么安装什么即可。
3. 重启系统并更换内核，注意需要进入 recovery 模式并选择 resume，否则会黑屏。顺利进入后查看内核版本，如图 2.1 所示，说明编译和安装内核这些步骤没有问题，可以开始编写新的系统调用。
4. 在 /usr/src/linux-4.14.267/kernel/sys.c 下添加系统调用代码，我选择的是添加文件拷贝功能，如图 2.2 所示。注意在编写系统调用代码时需要使用内核函数，如在进行文件读写时需要使用 sys\_open、sys\_read、sys\_write。同时由于传入的文件名是存放在用户空间下，所以在使用文件名时需要使用 set\_fs 改变内核对内存地址检查的处理方式以绕过用户空间参数检查，最后使用 get\_fs 恢复原来的内存地址检查。
5. 在 /usr/src/linux-4.14.267/include/linux/syscall.h 中添加函数声明，如图 2.3 所示。
6. 在 /usr/src/linux-4.14.267/arch/x86/entry/syscalls/syscall\_64.tbl 中添加编号

以及系统调用的名称，如图 1.4 所示。

7. 重新编译内核并安装，更换新内核启动，编写如图 2.5 所示的测试代码，运行结果如图 2.6 所示，可以看到返回值为 0，复制成功，使用 `cmp` 命令可以看到两个文件完全一致。

```
yizumi@ubuntu: ~/Desktop
yizumi@ubuntu:~/Desktop$ cat /proc/version
Linux version 4.14.267 (root@ubuntu) (gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04)) #1 SMP Tue Feb 22 22:37:56 CST 2022
yizumi@ubuntu:~/Desktop$
```

图 2.1 更换内核成功

```
2567 asmlinkage long sys_mycopy(const char *src_file, const char *copy_file){
2568     int src, target, count;
2569     char buf[256];
2570     mm_segment_t old_fs=get_fs();
2571     set_fs(KERNEL_DS);
2572     if((src=sys_open(src_file,O_RDONLY,0)) == -1) {
2573         return 1;
2574     }
2575     if((target=sys_open(copy_file,O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR))== -1){
2576         return 2;
2577     }
2578     while((count = sys_read(src,buf,256))>0)
2579     {
2580         if(sys_write(target,buf,count)!= count)
2581             return 3;
2582     }
2583     if(count == -1) return 4;
2584     sys_close(src);
2585     sys_close(target);
2586     set_fs(old_fs);
2587     return 0;
2588 }
```

图 2.2 系统调用代码

```
943 asmlinkage long sys_mycopy(const char *src_file, const char *copy_file);
944
945 #endif
```

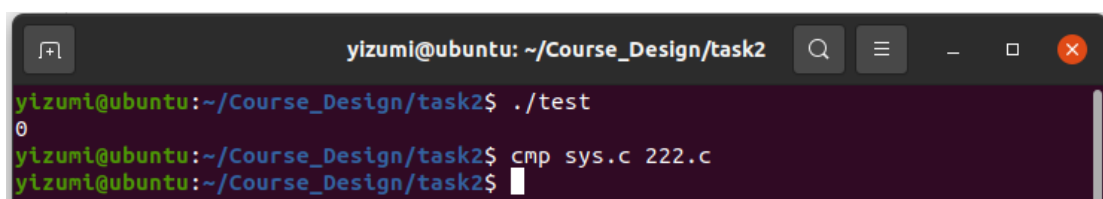
图 2.3 系统调用头文件

```
341 332     common statx             sys_statx
342 333     common mycopy            sys_mycopy
343 #
344 # x32-specific system call numbers start at 512 to avoid cache impact
345 # for native 64-bit operation.
```

图 2.4 添加系统调用号

```
1 #include<linux/kernel.h>
2 #include<sys/syscall.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 int main()
7 {
8
9     long int aaa=syscall(333,"sys.c","222.c");
10    printf("%ld\n",aaa);
11    return 0;
12 }
```

图 2.5 测试代码

A terminal window with a dark background. The title bar shows 'yizumi@ubuntu: ~/Course\_Design/task2'. The prompt is 'yizumi@ubuntu:~/Course\_Design/task2\$'. The first command is './test', followed by a carriage return. The second command is 'cmp sys.c 222.c', followed by a carriage return. The third command is 'cmp sys.c 222.c', followed by a carriage return and a cursor.

```
yizumi@ubuntu:~/Course_Design/task2$ ./test
yizumi@ubuntu:~/Course_Design/task2$ cmp sys.c 222.c
yizumi@ubuntu:~/Course_Design/task2$
```

图 2.6 比较复制文件与源文件

## 2.4.2 实验调试及心得

实验二是本次课设中耗时最长的一个实验，也是难度比较大的一个实验。首先是编译内核的时间就很长，后面了解到可以增加编译线程数以提高编译速度，但是由于是在虚拟机里进行实验，而且我本机的 CPU 也不太行，每次重新编译依然要等很长时间；

其次由于我没有意识到内核编译完很大，我连续编译了好几个内核，由于磁盘空间不够大，有一个内核死活编译不过去，由于没有仔细看报错提醒，在这里也卡了很长时间，最后删掉了已经确定无法进行实验的内核腾出空间解决问题

最后也是本次实验最大的困难是，由于教程和网上的资料都比较老，操作系统和 Linux 内核的版本都对不上，忽略版本只是按照资料很可能会实验失败，我第一次选择的内核是最新的 5.16 版本，发现不允许使用 `set_fs` 和 `get_fs` 函数，之后选择 5.4 版本内核，又不允许使用 `sys_open` 等函数，之后选择 4.14 版本内核，编译安装后却进不去系统，并显示 CPU 不支持 5 level paging，且在网上搜不到对应的解决方法。受到之前修改 `.config` 种 `CONFIG_SYSTEM_TRUSTED_KEYS` 项的启发，我尝试着在 `.config` 文件搜索有关 level 的设置项，并将有关五级页面缓存的项删掉或者修改页面缓存数量为 4，最后才成功。

本次任务帮助我了解了 Linux 系统内核代码结构，同时了掌握系统调用的实现过程，很有收获。

## 附录 实验代码

系统调用代码

```
asmlinkage long sys_mycopy(const char *src_file,const char *copy_file){
    int src,target,count;
    char buf[256];
    mm_segment_t old_fs=get_fs();
    set_fs(KERNEL_DS);
    if((src=sys_open(src_file,O_RDONLY,0)) == -1)    {
        return 1;
    }
}
```

```

        if((target=sys_open(copy_file,O_WRONLY | O_CREAT, S_IRUSR |
S_IWUSR))== -1){
            return 2;
        }
        while((count = sys_read(src,buf,256))>0)
        {
            if(sys_write(target,buf,count)!= count)
                return 3;
        }
        if(count == -1) return 4;
        sys_close(src);
        sys_close(target);
        set_fs(old_fs);
        return 0;
    }

```

测试代码

```

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

int main()
{

    long int aaa=syscall(333,"sys.c","222.c");
    printf("%ld\n",aaa);
    return 0;
}

```

## 3 实验三 增加字符驱动设备

### 3.1 实验目的

掌握增加设备驱动程序的方法。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。

### 3.2 实验内容

要求演示实现字符设备读、写,选择实现键盘缓冲区,不同进程、追加、读取。

### 3.3 实验设计

#### 3.3.1 开发环境

操作系统: Ubuntu 20.04.3

操作系统类型: 64bit

内核: Linux version 4.14.267

内存: 7.7GB

处理器: Intel® Core™ i5-8265U CPU @ 1.60GHz × 4

磁盘: 85.9GB

#### 3.3.2 实验设计

设备驱动程序是操作系统内核和机器硬件之间的接口,其为应用程序屏蔽了硬件的细节,将硬件设备抽象为一个设备文件,使应用程序可以像操作普通文件一样对硬件设备进行操作。设备驱动程序的主要功能有以下几点:

1. 对设备的初始化和释放
2. 把数据从内核传送到硬件和从硬件读取数据
3. 读取应用程序传送给设备文件的数据和回送应用程序请求的数据
4. 检测和处理设备出现的错误

Linux 支持 3 种设备: 字符设备、块设备和网络设备。我们要实现的是字符

设备，对字符设备发出读/写请求时，实际的硬件 I/O 一般就紧接着发生了，不需要设计缓冲区。

在设备驱动程序中有一个非常重要的结构 `file_operations`，该结构的每个域都对应着一个系统调用，包括设备文件的打开，写入，读取和关闭，呼应了前面说的设备驱动程序将硬件设备抽象为文件，所以编写设备驱动程序，本质上就是完成上述的四个系统调用。

编写完后编译程序，在内核中挂载该模块，然后创建新的虚拟设备文件，最后测试即可。

## 3.4 实验调试

### 3.4.1 实验步骤

具体的实验步骤分为以下几步：

1. 编写驱动程序源文件，主要是填写结构体 `file_operation` 的各个域，并完成注册设备函数，注销设备函数，以及声明模块许可证。
2. 编写 `makefile` 文件，写法参照 PPT 给出的通用 `Makefile`
3. 编译源文件，得到 `.ko` 驱动程序。
4. 使用 `insmod` 命令挂载模块，然后使用 `cat /proc/devices` 查看系统分配的设备号，如图 3.1 所示，可以看到新增设备 `yizumi` 的设备号为 240。
5. 使用 `mknod /dev/yizumi 240 0` 创建新的虚拟设备，该命令的最后三个参数说明了设备是字符设备，主设备号为 240，次设备号为 0。之后使用 `ls /dev` 命令查看虚拟设备文件，如图 3.2，可以看到 `yizumi` 已经添加成功

```
189 usb_device
202 cpu/msr
204 ttyMAX
226 drm
240 yizumi
241 aux
242 hidraw
```

图 3.1 查看系统分配设备号

```
hwrng      rtc      tty23     tty53     ttyS24    vfio
initctl    rtc0     tty24     tty54     ttyS25    vga_arbiter
input      sda      tty25     tty55     ttyS26    vhci
kmsg       sda1     tty26     tty56     ttyS27    vhost-net
kvm        sda2     tty27     tty57     ttyS28    vhost-vsock
lightnvme  sda3     tty28     tty58     ttyS29    vmci
log        sda5     tty29     tty59     ttyS30    vsock
loop0      sg0      tty3      tty6      ttyS31    yizumi
loop1      sg1      tty30     tty60
yizumi@ubuntu:~/Course_Design/task3$
```

图 3.2 查看虚拟设备文件



6. 编写测试代码，测试读写功能，代码见附录，测试结果如图 3.3，可以看到读写功能可以正确执行。
7. 测试结束卸载设备，最后使用 `demsg` 命令查看内核中输出的调试信息，观察设备的创建和卸载是否正确执行，如图 3.4。

```
yizumi@ubuntu:~/Course_Design/task3$ sudo ./test
[sudo] password for yizumi:
read result:
you can rewrite this sentence.
input a string to test write:
To be or not to be, that is the question.
read after write result:
To be or not to be, that is the question.
yizumi@ubuntu:~/Course_Design/task3$
```

图 3.3 使用测试代码测试读写功能

```
[ 8558.690402] the device is opened!
[ 8558.690406] read success!
[ 8607.047371] write success!
[ 8607.047374] read success!
[ 8607.047396] the device is released!
[ 8684.973118] the driver has been cleaned!
yizumi@ubuntu:~/Course_Design/task3$
```

图 3.4 查看调试信息是否正确输出

### 3.4.2 实验调试及心得

实验三的资料比较好找，所以写起来比较流畅，因为实验并不要求实现很复杂的内核功能，而注册和注销的代码网上的资料也都讲的很清楚，除了最开始忽视了用户空间和内核空间之间的区别造成了读写失败以外，其他都几乎没有碰到困难。需要注意的是实验过程中的所有命令，包括编译、`insmod`、`cat /proc/devices`、`mknod`、以及测试程序的运行，都需要 `sudo` 权限。

本次实验使我对 `linux` 的模块和设备管理、运行机制有了更深刻的了解，理解了设备驱动的原理以及添加设备驱动的方法。

## 附录 实验代码

设备驱动代码 `yizumi.c`

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

```

#include <linux/errno.h>
#include <asm/segment.h>
long int dev_no = 0;
int mutex=0;
char dev_info[1024]="you can rewrite this sentence.";

ssize_t myread(struct file *flip,char *buf,size_t count, loff_t*f_pos)
{
    int res = copy_to_user(buf, dev_info, sizeof(dev_info));
    if(res == 0)
    {
        printk("read success!\n");
        return count;
    }
    else
    {
        printk("can't read!\n");
        return -1;
    }
}

ssize_t mywrite(struct file *flip,const char *buf,size_t count, loff_t*f_pos)
{
    int res = copy_from_user(dev_info, buf, sizeof(dev_info));
    if(res == 0)
    {
        printk("write success!\n");
        return 0;
    }
    else
    {
        printk("can't write!\n");
        return -1;
    }
}

int myopen(struct inode *inode,struct file *file )
{
    if (mutex == 0)
    {
        mutex++;
        try_module_get(THIS_MODULE);
        printk("the device is opened!\n");
    }
}

```

```

        return 0;
    }
    else
    {
        printk("another process open the device!\n");
        return -1;
    }
}

int myrelease (struct inode *inode,struct file *file )
{
    mutex--;
    module_put(THIS_MODULE);
    printk("the device is released!\n");
    return 0;
}

struct file_operations fops={
    .read = myread,
    .write=mywrite,
    .open=myopen,
    .release=myrelease
};

static int init_mymodule(void)
{
    int result;
    result = register_chrdev(0, "yizumi", &fops);
    if (result < 0) {
        printk("register failed!\n");
        return result;
    }
    if (dev_no == 0) dev_no = result;
    printk("register succeed!\n");
    return 0;
}

static void cleanup_mymodule(void)
{
    unregister_chrdev(dev_no,"yizumi");
    printk("the driver has been cleaned!\n");
}

```

```
module_init(init_mymodule);
module_exit(cleanup_mymodule);
MODULE_LICENSE("GPL");
```

测试代码 test.c

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

int main()
{
    char buf_read[1024];
    char buf_write[1024];
    int file = open("/dev/yizumi", O_RDWR | O_NONBLOCK);
    if (file == -1)
    {
        printf("open failed!\n");
        return -1;
    }
    else
    {
        read(file, buf_read, sizeof(buf_read));
        printf("read result:\n%s\n", buf_read);
        printf("input a string to test write:\n");
        scanf("%[^\n]", buf_write);
        write(file, buf_write, sizeof(buf_write));
        read(file, buf_read, sizeof(buf_read));
        printf("read after write result:\n%s\n", buf_read);
        close(file);
        return 0;
    }
}
```

## 4 实验四 使用 QT 实现系统监控器

### 4.1 实验目的

了解/proc 文件的特点和使用方法，并能够通过/proc 文件了解系统信息以及监控系统中进程运行情况，使用图形化界面展示获取的系统信息。

### 4.2 实验内容

通过读取 proc 文件系统，获取系统各种信息，并以图形化界面的方式显示出来，具体包括：

1. 获取并显示主机名
2. 获取并显示系统启动的时间
3. 显示系统到目前为止持续运行的时间
4. 显示系统的版本号
5. 显示 CPU 的型号和主频的大小
6. 通过 pid 或者进程名查询一个进程，并显示该进程的详细信息，提供杀掉该进程的功能
7. 显示系统所有进程的一些信息，包括 pid, ppid, 占用内存大小, 优先级等
8. 两分钟内 CPU 使用率的历史记录曲线的图形化显示
9. 两分钟内内存和交换分区使用率历史记录曲线的图形化显示
10. 在状态栏显示当前时间
11. 在状态栏显示当前 CPU 使用率
12. 在状态栏显示当前内存使用情况
13. 用新进程运行一个其他程序
14. 关机

### 4.3 实验设计

#### 4.3.1 开发环境

操作系统：Ubuntu 20.04.3

操作系统类型：64bit

内核：Linux version 5.13.0-30-generic

内存：7.7GB

处理器：Intel® Core™ i5-8265U CPU @ 1.60GHz × 4

磁盘：85.9GB

Qt 版本：5.9.9

## 4.3.2 实验设计

实验指导中已经给出了实验要求的信息应该从哪些文件获取，所以我们只需要看懂这些文件的内容并处理即可，具体如下：

1. 主机名从文件`/proc/sys/kernel/hostname`中获取。文件中只有一个字符串，就是主机名
2. 系统的启动时间和持续运行时间从`/proc/uptime`中获取。文件中有两个浮点数，第一个是系统从启动到现在的时间，第二个时间是系统空闲时间。第一个时间也就是我们需要的系统持续运行时间，第二个时间对我们没有用，另一个我们需要的系统启动时间应使用当前时间减去系统启动到现在的时间。
3. 实验指导给出的获得系统版本号方法是从`/proc/sys/kernel/ostype`以及`/proc/sys/kernel/osrelease`中获得，前一个文件是系统的类型，后一个文件中存有系统版本号，拼接即可得到完整系统版本号。而在之前更换系统内核时还使用过`/proc/version`文件，其中有系统版本和 gcc 版本，可以通过截取获得系统版本号，我使用的时`/proc/version`文件
4. CPU 型号和主频从`/proc/cpuinfo`文件中获得，该文件中有很多信息，但都是以键值对的方式显示的，可以通过字符串查找获得 CPU 型号和主频
5. 通过进程名或者 pid 查找进程，显示所有进程的信息通过`/proc`文件夹下的数字文件夹中的`stat`文件完成，数字文件夹的名称就是进程号，其中的`stat`文件包含一系列字符串，包含了我们需要的所有信息，可根据资料取出对应信息
6. 杀死进程和启动进程不需要使用`/proc`文件，可以使用 shell 命令启动
7. CPU 的使用率从`/proc/stat`中获得，文件中首先是整个 CPU 从开机到打开文件时在各个状态运行的时间以及空闲时间，之后是 CPU 各个核对应的运行和空闲时间，之后是一系列中断次数、切换上下文次数等信息。我们只需要第一行数据，需要注意的是这些时间是从开机到当前时间

CPU 在各个状态的时间，直接用这些数据计算使用率的话得到是从开机到现在的平均速率。我使用的办法是在一个较短时间内两次打开文件，求在这段时间内的速率，具体的操作方式是打开一次文件，读取所有时间，之后休眠 100ms，再打开文件读取所有时间，讲两次获得的对应时间做差，即可求出 100ms 内 cpu 处于各个状态的时间，进而算出使用率，使用这 100ms 的速率作为这一秒的瞬时使用率。记录两分钟内每秒的使用率，使用 qCustomPlot 绘图。

8. 内存和交换分区使用情况从 /proc/meminfo 中获得，文件中的信息以键值对的形式给出，截取需要的信息即可。绘图方式和绘制 CPU 图像一致
9. 关机功能使用 shell 命令实现

## 4.4 实验调试

### 4.4.1 实验步骤

首先确定窗口布局，参考 Windows 任务管理器的布局，我使用了一个 TabWidget，其中一个 tab 用来显示进程信息，一个 tab 用来显示 CPU、内存和交换分区的历史曲线，剩下的相对静态的信息使用一个 tab。实时的 CPU 和内存使用率、当前时间以及关机键则放在 TabWidget 的下面。

之后怎样更新信息。我使用了一个槽 update，其与 QTimer 的一个实例的 timeout 信号绑定，timeout 被设定为 1 秒发生一次。update 中调用了更新三个 tab 的函数，为了实现方便，更新 CPU 和内存的实时使用率的代码放在了更新历史曲线 tab 的函数里。

至于具体的更新页面的代码，就是读取对应文件数据并处理，然后更新对应的控件。首先是显示系统信息的 tab，其中的数据按实验设计中的方式处理后以文本的方式显示出来即可，效果图如图 4.1。

之后是进程的详细信息，我是用一个 listWidget 显示的，每一个进程的信息占一行，通过制表符实现每一列的对齐，效果图如图 4.2。

对于进程的搜索，我提供的是按进程名搜索进程，具体方法是遍历所有进程文件并比较进程名，搜索到之后再 listWidget 中只显示目标进程。为了方便使用，我设置了一个变量标识当前是否在查询特定进程，若是，则 update 调用显示特定进程的函数，否则调用显示所有进程的函数。运行和杀死进程直接使用 system 函数调用 shell 命令实现，需要注意的是运行进程时需要将进程挂到后台运行，即在进程名后加 ‘&’，否则会阻塞。为了测试功能的正确性，我编写了一个死循环程序 show，首先运行两个 show，然后搜索 show，结果如图 4.3，然后按 pid

杀死一个 show，结果如图 4.4。

对于 CPU、内存和交换分区的历史记录曲线，按照实验设计中的方法求出实时使用率后，首先将此前记录的时间和使用率的对应关系全部后移，即之前存的第 120 个数据移动到第 119 个，以此类推，原来的第一个数字丢弃，新的数据放在第 120 个的位置，然后绘图即可。效果如图 4.5 所示。其中 CPU 和内存的使用情况在启动新进程可以看到明显变化，由于我的虚拟机的内存有 8 个 G，所以一直没有用到交换分区，交换分区使用率一直为 0。

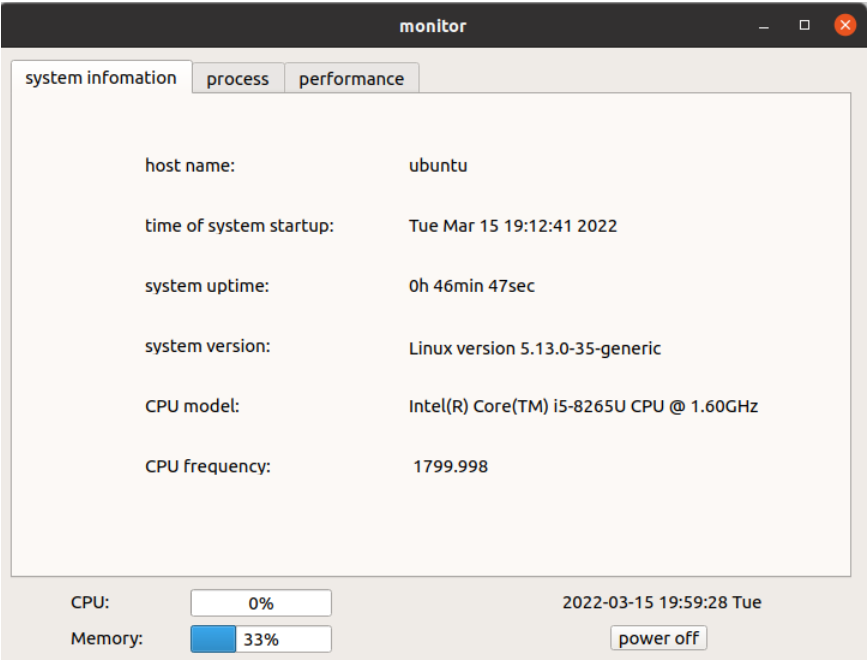


图 4.1 显示系统信息

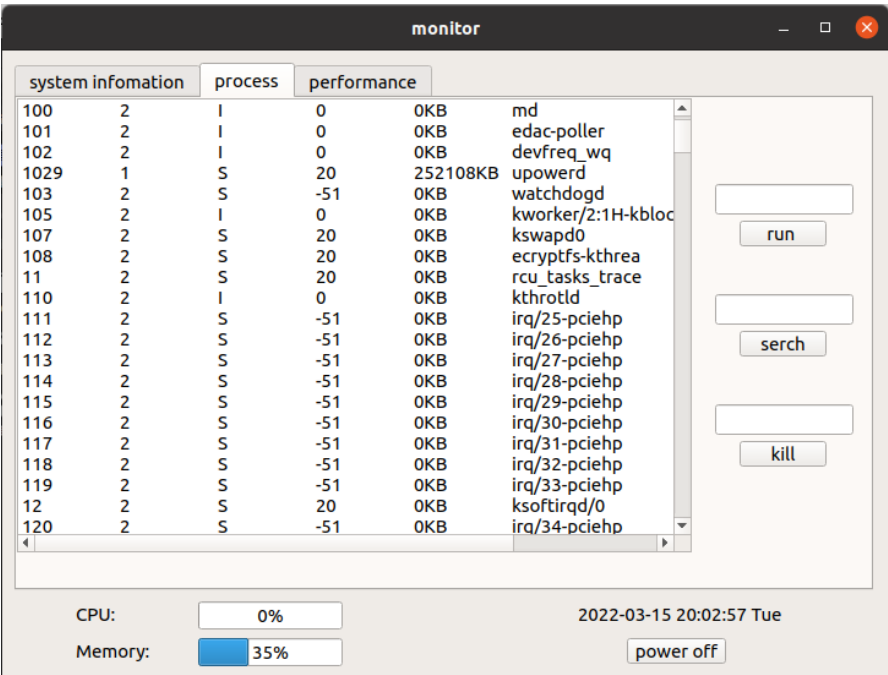


图 4.2 显示进程信息



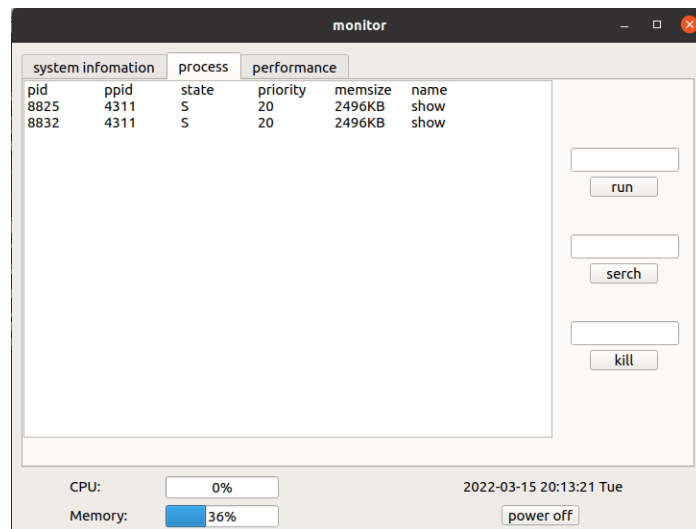


图 4.3 运行两次 show 后查找 show 进程结果

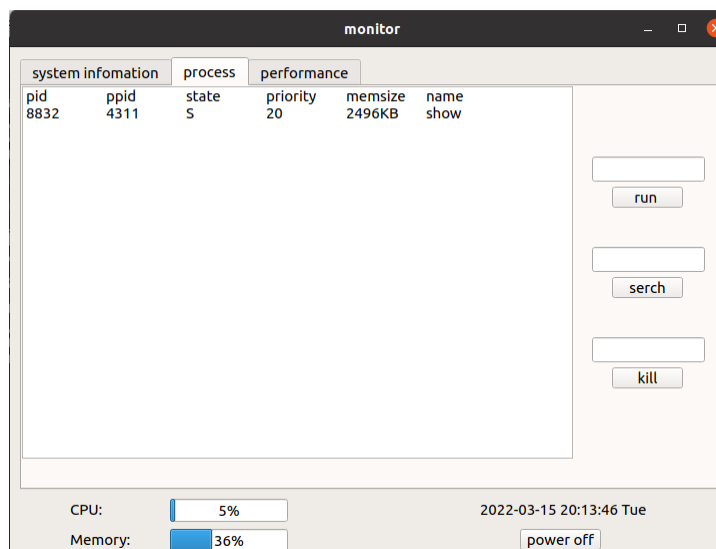


图 4.4 杀死 8825 号进程结果

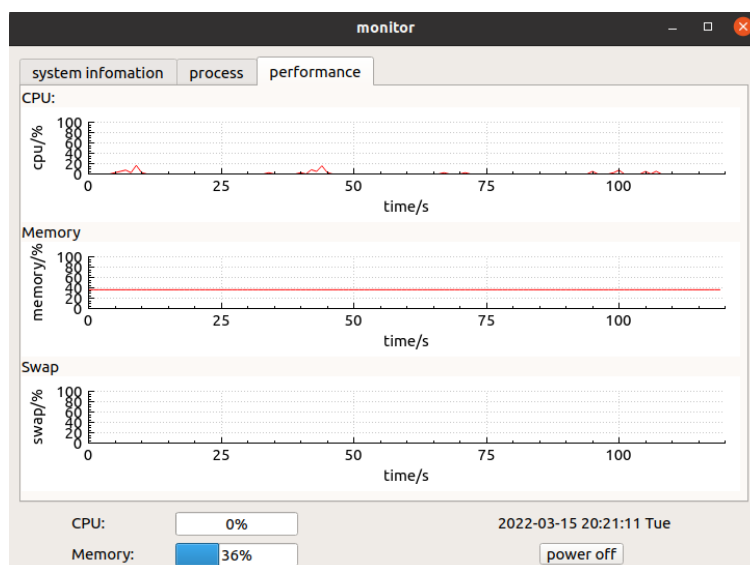


图 4.5 CPU、内存和交换分区使用率

## 4.4.2 实验调试及心得

本次实验主要的任务在于熟悉/proc 下的各个文件，了解每个文件的内容后，数据的处理就不困难了，之后的时间基本是花费在可视化界面的调整上。

本次实验中碰到的一个问题是运行新的进程时我的系统监视器会卡死，由于测试程序写的是一个死循环程序，我一直没搞明白问什么会这样，直到之后试了一个正常的程序，才发现这个程序退出后系统监视器就恢复正常了，我才意识到需要将程序放到后台运行才不会发生阻塞。

本次实验我熟悉了 proc 目录下各个文件的功能和参数，并对 QT 软件的使用更加熟悉了。

## 附录 实验代码

由于 UI 是使用可视化工具设计的，代码意义不大，所以不贴上来了，具体效果见实验步骤中的截图。由于使用了 qcustomplot 绘图，工程中包含了 qcustomplot 源代码，但是和实验关系不大，所以也不贴上来了。

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 slots:
    void update();
    void on_poweroff_clicked();

    // void on_pushButton_clicked();
```

```

void on_pushButton_run_clicked();

void on_pushButton_search_clicked();

void on_pushButton_kill_clicked();

private:
    double time[120]={0};
    double cpu[120]={0};
    double mem[120]={0};
    double swap[120]={0};
    bool isSearching;
    QString pro_name;
    Ui::MainWindow *ui;
    bool show_certain_pro();
    void show_system_info();
    void show_process_info();
    void show_performance();
};
#endif // MAINWINDOW_H

```

main.cpp

```

#include "mainwindow.h"

#include <QApplication>

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

```

mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QTimer>
#include <QTime>
#include <QFile>
#include <QDir>
#include <QMessageBox>

```

```

#include <unistd.h>
#include <QDebug>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent),isSearching(false)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    QTimer *timer=new QTimer(this);
    timer->start(1000);
    connect(timer,SIGNAL(timeout()),this,SLOT(update()));
    for(int i=0;i<120;i++)
    {
        time[i]=119-i;
    }
    ui->cpu_graphic->xAxis->setLabel("time/s");
    ui->cpu_graphic->yAxis->setLabel("cpu/%");
    ui->cpu_graphic->xAxis->setRange(0,120);
    ui->cpu_graphic->yAxis->setRange(0,100);
    ui->mem_graphic->xAxis->setLabel("time/s");
    ui->mem_graphic->yAxis->setLabel("memory/%");
    ui->mem_graphic->xAxis->setRange(0,120);
    ui->mem_graphic->yAxis->setRange(0,100);
    ui->swap_graphic->xAxis->setLabel("time/s");
    ui->swap_graphic->yAxis->setLabel("swap/%");
    ui->swap_graphic->xAxis->setRange(0,120);
    ui->swap_graphic->yAxis->setRange(0,100);
}

void MainWindow::show_system_info(){
    QFile f;
    QString info;
    //get host name
    f.setFileName("/proc/sys/kernel/hostname");
    f.open(QIODevice::ReadOnly);
    info = f.readLine();
    ui->hostName->setText(info);
    f.close();

    //get startup time and up time
    //there are two num in the file,the first is what we need
    f.setFileName("/proc/uptime");
    f.open(QIODevice::ReadOnly);
    info = f.readLine();
    float alltime=info.section(" ",0,0).toFloat();
    int h = alltime/3600;

```

```

int min = (alltime - h*3600)/60;
int sec = alltime - h*3600 - min*60;
QString all_time=info.sprintf ("%dh %dmin %dsec",h,min,sec);
ui->startUpTime->setText(QDateTime::currentDateTime().addSecs(alltime*(-1)).
toString());
ui->uptime->setText(all_time);
f.close();

//get system version
//use '(' as the end of version string
f.setFileName("/proc/version");
f.open(QIODevice::ReadOnly);
info = f.readLine();
ui->version->setText(info.mid(0,info.indexOf('(')));
f.close();

//get cpu model and frequency
//lots of information in the file, just use what we need
f.setFileName("/proc/cpuinfo");
f.open(QIODevice::ReadOnly);
while (1)
{
    info = f.readLine();
    if(info == NULL)
    {
        break;
    }
    int pos=0;
    if((pos=info.indexOf("model name"))!=-1)
    {
        QString cpu_id=info.mid(pos+13,info.length());
        ui->cpu->setText(cpu_id);
    }
    if((pos=info.indexOf("cpu MHz"))!=-1)
    {
        QString cpu_hz=info.mid(pos+10,info.length());
        ui->frequency->setText(cpu_hz);
    }
}
f.close();
}

void MainWindow::show_process_info(){

```

```

QFile f;
ui->listWidget->clear();
//the information of process is stored in folders named with numbers
//so we should get these folders first
QDir qd("/proc");
QStringList qslst=qd.entryList();
QString qs=qslst.join("\n");
//    qDebug()<<qs;
//the first two folders are '.' and '..',so we should find '\n' after the first three char
int find_start = 3;
//    int all_pid_num=0;
bool back;
int l=0,r=0;
ui->listWidget->addItem("pid\tppid\tstate\tpriority\tmemsize\tname");
while(find_start!=-1)
{
    l=qs.indexOf("\n",find_start);
    r=qs.indexOf("\n",l+1);
    find_start=r;
    QString name_of_pro=qs.mid(l+1,r-l-1);
    name_of_pro.toInt(&back,10);
    if(!back)
        continue;
//    all_pid_num++;
    f.setFileName("/proc/"+ name_of_pro +"/stat");
    f.open(QIODevice::ReadOnly);
    QString content = f.readAll();
    QStringList temp=content.split(' ');
    //get process name
    QString name=temp.at(1);
    QString temp_proc_name=name.mid(1,name.length()-2);
//    qDebug()<<name_of_pro<<" "<<temp_proc_name;
    //get process state
    QString temp_proc_state=temp.at(2);
    //get ppid
    QString temp_proc_ppid=temp.at(3);//ppid
    //get process priority
    QString temp_proc_priority=temp.at(17);
    //get process memory use
    QString temp_proc_mem=temp.at(22);
    //change mem_used unit from b to kb for better displaying
    int size=temp_proc_mem.toUInt()/1024;

```

```

        ui->listWidget->addItem(name_of_pro+"\t"+temp_proc_ppid+"\t"+temp_proc_s
tate+"\t"+temp_proc_priority+"\t"+QString::number(size)+"KB"+" \t"+temp_proc_name)
;
        f.close();
    }
//    qDebug()<<all_pid_num;
}

bool MainWindow::show_certain_pro(){
    QFile f;
    QDir qd("/proc");
    QStringList qlist=qd.entryList();
    QString qs=qlist.join("\n");
    int find_start = 3;
    bool back,found=false;
    int l=0,r=0;
    QStringList temp;
    QString content;
    QString temp_proc_name;
    ui->listWidget->clear();
    ui->listWidget->addItem("pid\tppid\tstate\tpriority\tmemsize\tname");
    while(find_start!=-1){
        l=qs.indexOf("\n",find_start);
        r=qs.indexOf("\n",l+1);
        find_start=r;
        QString name_of_pro=qs.mid(l+1,r-l-1);
        name_of_pro.toInt(&back,10);
        if(!back)
            break;
        f.setFileName("/proc/"+ name_of_pro +"/stat");
        f.open(QIODevice::ReadOnly);
        content = f.readAll();
        temp=content.split(' ');
        //get process name
        temp_proc_name=temp.at(1);
        temp_proc_name=temp_proc_name.mid(1,temp_proc_name.length()-2);
        if(this->pro_name==temp_proc_name){
            found=true;
            //get pid
            QString pid_num=temp.at(0);
            //get process state
            QString temp_proc_state=temp.at(2);
            //get ppid
            QString temp_proc_ppid=temp.at(3);//ppid

```

```

        //get process priority
        QString temp_proc_priority=temp.at(17);
        //get process memory use
        QString temp_proc_mem=temp.at(22);
        //change mem_used unit from b to kb for better displaying
        int size=temp_proc_mem.toUInt()/1024;
        ui->listWidget->addItem(pid_num+"\t"+temp_proc_ppid+"\t"+temp_proc_st
ate+"\t"+temp_proc_priority+"\t"+QString::number(size)+"KB"+" \t"+temp_proc_name);
    }
    f.close();
}
return found;
}

void MainWindow::show_performance(){
    QFile f;
    QStringList info;
    QString temp;
    int totaltime[2]={0,0};
    int cputime[2][9];
    for(int k=0;k<2;k++)
    {
        f.setFileName("/proc/stat");
        f.open(QIODevice::ReadOnly);
        temp =f.readLine();
        //    qDebug()<<temp;
        info=temp.split(' ');
        for(int i=0;i<9;i++)
        {
            cputime[k][i] = info.at(i+2).toInt();
            totaltime[k]+=cputime[k][i];
        }
        f.close();
        usleep(100000);
    }
    double idle=qAbs(cputime[0][3]-cputime[1][3]);
    double total=qAbs(totaltime[0]-totaltime[1]);
    double pcpu = 100* (total-idle)/total;
    QString cpu_used=temp.sprintf ("% .4f",pcpu);
    ui->cpu_used->setValue(pcpu);
    f.setFileName("/proc/meminfo");
    double mem_size;
    double mem_free_size;
    double swap_size;

```



```

double swap_free_size;
double mem_use=0;
double swap_use=0;
f.open(QIODevice::ReadOnly);
while ((temp = f.readLine())!=NULL)
{
    int from=0;
    if((from=temp.indexOf("MemTotal"))!=-1)
    {
        QString mem_total=temp.mid(from+10,temp.length()-from-14).trimmed();
        mem_size = mem_total.toInt();
    }
    if((from=temp.indexOf("MemFree"))!=-1)
    {
        QString mem_free=temp.mid(from+10,temp.length()-from-14).trimmed();
        mem_free_size = mem_free.toInt();
    }
    if((from=temp.indexOf("SwapTotal"))!=-1)
    {
        QString
swap_total=temp.mid(from+10,temp.length()-from-14).trimmed();
        swap_size = swap_total.toInt();
    }
    if((from=temp.indexOf("SwapFree"))!=-1)
    {
        QString swap_free=temp.mid(from+10,temp.length()-from-14).trimmed();
        swap_free_size = swap_free.toInt();
    }
}
f.close();
mem_use= 100*(mem_size-mem_free_size)/mem_size;
ui->mem_used->setValue(mem_use);
swap_use = 100*(swap_size-swap_free_size)/swap_size;
QVector<double> x1(120);
QVector<double> y1(120);
QVector<double> x2(120);
QVector<double> y2(120);
QVector<double> x3(120);
QVector<double> y3(120);
for(int i=0;i<119;i++)
{
    cpu[i]=cpu[i+1];
    mem[i]=mem[i+1];
    swap[i]=swap[i+1];
}

```

```

    }
    mem[119]=mem_use;
    swap[119]=swap_use;
    cpu[119]=pcpu;
    for(int i=0;i<120;i++)
    {
        x1[i]=time[i];
        y1[i]=cpu[i];
        x2[i]=time[i];
        y2[i]=mem[i];
        x3[i]=time[i];
        y3[i]=swap[i];
    }
    ui->cpu_graphic->addGraph();
    ui->cpu_graphic->graph(0)->setPen(QPen(Qt::red));
    ui->cpu_graphic->graph(0)->setData(x1,y1);
    ui->cpu_graphic->replot();

    ui->mem_graphic->addGraph();
    ui->mem_graphic->graph(0)->setPen(QPen(Qt::red));
    ui->mem_graphic->graph(0)->setData(x2,y2);
    ui->mem_graphic->replot();

    ui->swap_graphic->addGraph();
    ui->swap_graphic->graph(0)->setPen(QPen(Qt::red));
    ui->swap_graphic->graph(0)->setData(x3,y3);
    ui->swap_graphic->replot();
}

void MainWindow::update()
{
    this->show_system_info();
    if(!isSearching)
        this->show_process_info();
    else
        this->show_certain_pro();
    this->show_performance();
    ui->time ->setText(QDateTime::currentDateTime().toString("yyyy-MM-dd
hh:mm:ss ddd"));
}

MainWindow::~MainWindow()
{

```

```

        delete ui;
    }

void MainWindow::on_poweroff_clicked()
{
    system("shutdown -h now");
}

void MainWindow::on_pushButton_run_clicked()
{
    QString pro = ui->new_pro->text();
    int status=system(pro.toLatin1()+"&");
    if(status==-1||!WIFEXITED(status)||WEXITSTATUS(status)){
        QMessageBox::warning(this, tr("run"), QString::fromUtf8("run process
failed!"), QMessageBox::Yes);
    }
    ui->new_pro->clear();
}

void MainWindow::on_pushButton_search_clicked()
{
    QString name = ui->targrt_pro->text();
    if(name.isEmpty()){
        if(!isSearching)
            QMessageBox::warning(this, tr("run"), QString::fromUtf8("input a process
name first!"), QMessageBox::Yes);
        else{
            isSearching=false;
            this->show_process_info();
        }
        return;
    }
    this->pro_name=name;
    isSearching=true;
    if(!this->show_certain_pro()){
        QMessageBox::warning(this, tr("run"), QString::fromUtf8("process not
exist!"), QMessageBox::Yes);
    }
    ui->targrt_pro->clear();
}

void MainWindow::on_pushButton_kill_clicked()
{
    QString pid_num = ui->kill_pro->text();

```

```

int status = system("kill "+pid_num.toLatin1());
ui->kill_pro->clear();
if(status==-1||!WIFEXITED(status)||WEXITSTATUS(status)){
    QMessageBox::warning(this, tr("run"), QString::fromUtf8("kill process
failed!"), QMessageBox::Yes);
    return;
}
QMessageBox::information(this, tr("run"), QString::fromUtf8("kill process
success"), QMessageBox::Yes);
}

//void MainWindow::on_pushButton_clicked()
//{
//    this->show_system_info();
//    this->show_process_info();
//    this->show_performance();
//}

```

## 5 实验五 模拟文件系统

### 5.1 实验目的

设计并实现一个模拟的文件系统。

### 5.2 实验内容

基本要求为用磁盘中的一个文件模拟一个磁盘，并设计一个模拟文件系统，设计文件目录项的结构以及空白块的管理方法，并提供命令实现文件的操作，如 touch、read、write、rm 等。可以选择支持多用户、树型目录以及图形界面。

### 5.3 实验设计

#### 5.3.1 开发环境

操作系统：Ubuntu 20.04.3

操作系统类型：64bit

内核：Linux version 5.13.0-30-generic

内存：7.7GB

处理器：Intel® Core™ i5-8265U CPU @ 1.60GHz × 4

磁盘：85.9GB

Qt 版本：5.9.9

#### 5.3.2 实验设计

由于虚拟机实在装不下其他 ide 了，所以本实验使用 QT 开发，但是依然是控制台程序。

首先确定文件目录项的结构，我才用的是类似 Unix 文件系统的方式，每个目录项由文件名和一个指向 i 节点的指针组成，文件的详细信息存储在 i 节点中，目录项和 i 节点的结构方式如图 5.1 所示。目录项内容存储在磁盘块中，和普通文件一样，方便目录文件和普通文件用相同的方法处理。

为了方便文件大小的扩展和空闲块的分配，我使用的文件物理结构为串联结

构于是在 i 节点的物理结构相关成员中需要有文件的起始块和中止块，为了方便程序的编写我还记录了总块数。空闲块的管理本想采用位示图法，但是由于时间关系，采用了用一个布尔类型的数组来记录每一个块使用情况，管理的方法和位示图一样但是花费的存储空间要大得多，有时间的话可以修改为位示图。

至此基本上需要的数据结构都已设计完毕。文件系统启动时，载入 inode 表和块使用记录表，根目录的 inode 固定为 inode 表的 0 号项。为了方便以后的使用，文件系统启动时还需要确定当前目录信息，之后切换目录时修改对应信息即可。文件系统的结构如图 5.2 所示

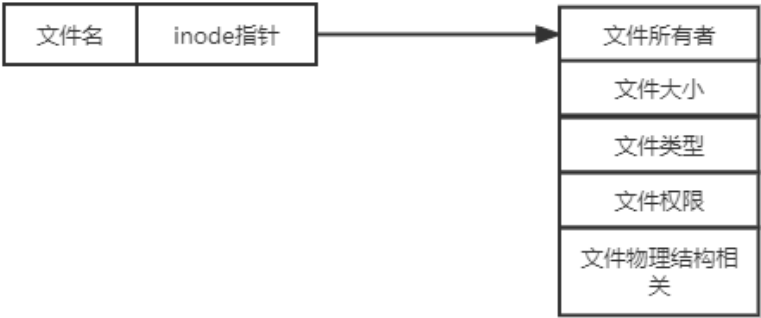


图 5.1 目录项结构

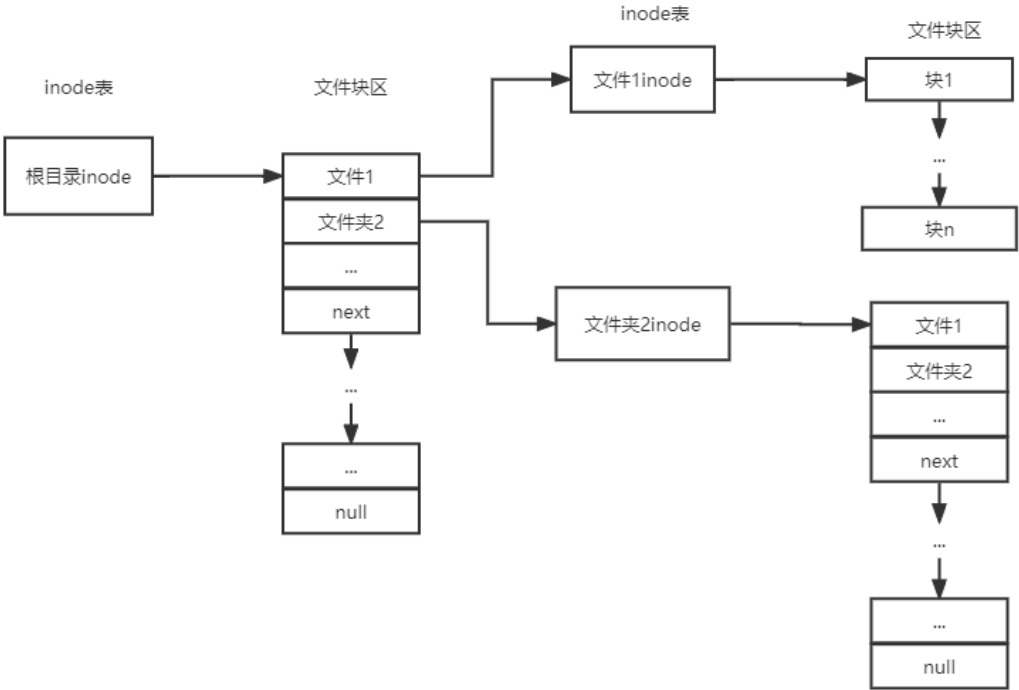


图 5.2 文件系统结构

之后就可以开始设计文件系统功能了，其中主要是对块的读写操作。  
首先设计读功能，对于普通文件，我们可以直接一块一块的读，计算文件在

最后一块的偏移并在最后一块只读偏移的部分，需要注意的细节是每一块最后四个字节是链表的指针，读的内容大小应该为块大小减 4 个字节，然后将最后四个字节读出来赋给用于遍历链表的块指针。对于目录文件，读取时更多是为了查找某个文件而不是获得所有文件，所以我使用的方法是一个目录项一个目录项的读，直到最后四个字节，用和读普通文件一样的思路再转到下一块。

之后是写功能。写功能对目录文件和普通文件一样，都是先判断最后一块能否写下，能的话直接写，否则先写能写下的部分，然后申请新块，将新块作为最后一块重复上述过程。创建文件本质上也是写文件，申请完需要的资源之后将新的目录项写道当前目录文件下。

接下来是删除文件，删除文件本身很简单，就是将其使用的块在块表中标记为未使用，然后将 `inode` 标记为未使用。难点在于删除目录文件中的目录项，按照一般的思路需要将被删除的目录项后面的所有目录项前移，比较难实现。在和同学讨论过后我决定不使用这种方法，而是遍历所有当前目录文件下所有目录项，将除了需要删除的目录项以外所有目录项复制下来，并释放块，然后重新将复制得到的目录项表写到目录文件中。

至此针对文件的操作基本设计完毕，接下来需要目录的跳转。由图 5.2 可知文件系统的目录是成树型结构的，跳转的一个最简单的思路是使用绝对路径，然后从根节点开始找对应目录。但是用户不一定输入绝对路径，所以第一步需要化绝对路径为相对路径，由于我们记录了当前目录的信息，将相对路径转为绝对路径不是难事，可以使用从根节点开始层层寻找的思路。

接下来是多用户的支持，一个简单的思路是为每一个文件添加一个所有者，然后根据操作者的不同给予不同的权限。由于时间比较紧，我这里设计的比较简单，只有两个权限，一个是所有人可读写，一个是只有所有者可读写，且除根目录外所有文件的权限都是只有所有者可读写。有时间的话可以增加一个管理员权限并提供修改文件权限的功能，并增加更多的权限选择。

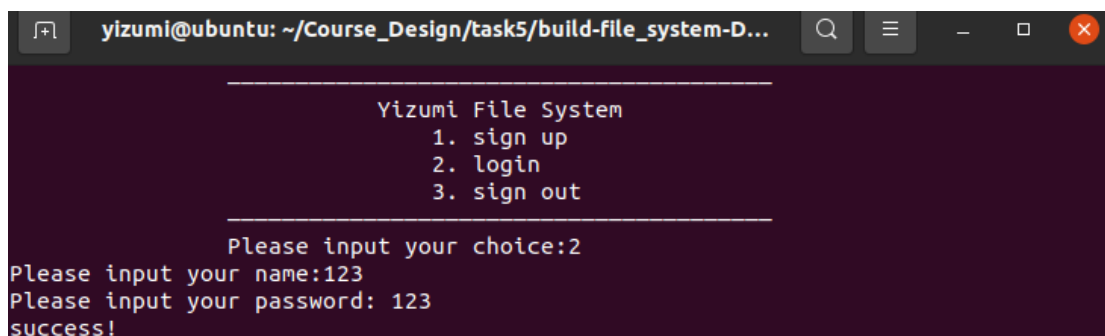
至此文件系统的基本功能设计完毕，再加上登录功能，并对一些提供的指令添加一些细节即可。

## 5.4 实验调试

### 5.4.1 实验步骤

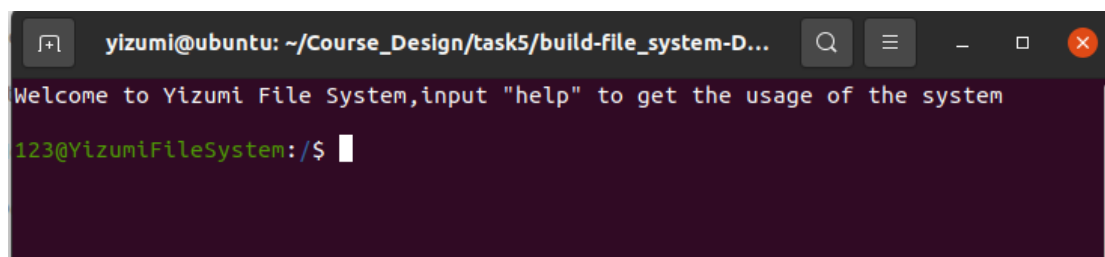
为了方便代码的编写，主函数是一个登录界面，其使用一个登录的工具类 `login` 管理用户的登录和注册，用户信息记录在本地文件中。一旦登录成功，使用用户名初始化文件系统类 `file_system`，之后 `file_system` 进入循环，等待用户指

令。登录界面和登录成功后的界面如图 5.3、5.4 所示，为了美观，登录成功后使用了彩色输出模拟 ubuntu shell。



```
yizumi@ubuntu: ~/Course_Design/task5/build-file_system-D...  
  
Yizumi File System  
1. sign up  
2. login  
3. sign out  
  
Please input your choice:2  
Please input your name:123  
Please input your password: 123  
success!
```

图 5.3 登陆界面



```
yizumi@ubuntu: ~/Course_Design/task5/build-file_system-D...  
  
Welcome to Yizumi File System,input "help" to get the usage of the system  
  
123@YizumiFileSystem:/$
```

图 5.4 登录成功后的界面

根据实验要求，决定支持创建文件指令 `touch`、创建文件夹指令 `mkdir`、读文件 `read`、写文件 `write`、拷贝文件 `cp`、删除文件 `rm`、显示当前文件夹下所有文件 `ls`，更换当前文件夹 `cd`。输入 `help` 显示指令的使用方法。其中大部分功能的实现思路在实验设计中已经阐述过，需要进一步说明的有以下几点：

1. `read` 和 `write` 使用的是 `read_to_s` 和 `write_from_s`，后两个函数是将文件内容读到字符串和将字符串写到文件里，更接近系统调用的功能，`read` 和 `write` 是对这两个函数的封装。
2. `cp` 命令是综合使用了 `read_to_s`、`write_from-s`、`touch` 和 `cd` 命令，基本思路是保存当前目录信息，然后 `cd` 到源文件目录，读文件内容，再 `cd` 到目的文件目录，创建文件，写入内容。
3. 删除指令没有设计 `-r` 参数，是直接根据删除的文件属性执行直接删除或者递归删除。当文件为普通文件或者为目录文件但是文件大小为 0，则直接删除，直接删除的思路就是实验设计中阐述的思路；否则遍历每个目录项，对每一项执行删除指令，然后再删除指定目录，注意删除单个文件时会同步更新目录，即删掉第一项后目录中的第二项自动变为第一项，所以我们只需要一直对第一个目录项执行删除就行了。
4. 所有对文件的操作都会检查权限，没有权限时会报 `permission deny`
5. 原本打算设置块大小 516 字节（因为一个目录项 16 字节），但是块太大了不利于查看串联文件能否正确运行，所以测试时块大小为 36，但是块



数量暂时没有改变，还是按照 516 字节计算得到。  
各个指令的运行结果如图所示

```
123@YizumiFileSystem:/$ touch a
success!

123@YizumiFileSystem:/$ mkdir b
success!

123@YizumiFileSystem:/$ ls
a
b
```

图 5.5 touch mkdir ls 指令运行结果

```
123@YizumiFileSystem:/$ write a
please input your text
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

123@YizumiFileSystem:/$ read a
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

图 5.6 write read 指令运行结果

```
123@YizumiFileSystem:/$ cd b

123@YizumiFileSystem:/b$ touch a
success!

123@YizumiFileSystem:/b$ touch b
success!

123@YizumiFileSystem:/b$ ls
a
b
```

图 5.7 cd 指令运行结果

```
123@YizumiFileSystem:/b$ cd ..

123@YizumiFileSystem:/$ ls
a
b

123@YizumiFileSystem:/$ cp a b/c
success!

123@YizumiFileSystem:/$ cd b

123@YizumiFileSystem:/b$ read c
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

图 5.8 cp 指令运行结果

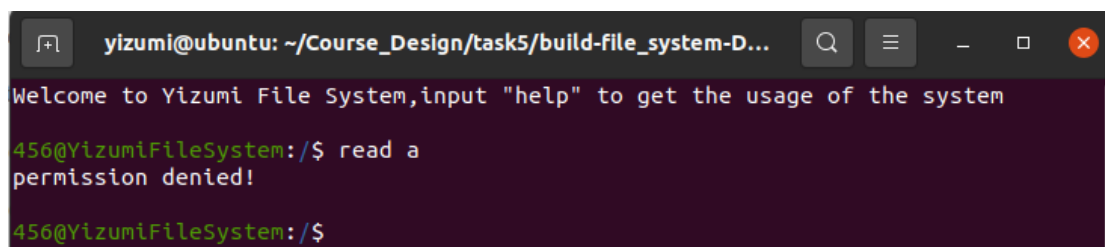
```
123@YizumiFileSystem:/b$ cd ..

123@YizumiFileSystem:/$ rm b

123@YizumiFileSystem:/$ ls
a

123@YizumiFileSystem:/$
```

图 5.9 rm 指令运行结果



```
yizumi@ubuntu: ~/Course_Design/task5/build-file_system-D...
Welcome to Yizumi File System,input "help" to get the usage of the system
456@YizumiFileSystem:/$ read a
permission denied!
456@YizumiFileSystem:/$
```

图 5.10 用户 456 登录后查看用户 123 创建的文件 a 显示无权限

## 5.4.2 实验调试及心得

本次实验我认为最大的难点在于对文件的操作，因为之前从来没有这样仔细的操作过文件，基本上只是打开读取关闭，而模拟文件系统对磁盘的读取时需要频繁修改文件指针位置，这一点我非常不熟练，加上 Qt 无法对有输入输出的程序进行单步调试，而读写文件由不像读写内存那样可以方便的输出中间信息，以至于在一些逻辑并不复杂的地方花费了大量的时间去调试。

其次由于设计的数据结构比较多，有 inode 表，inode，块，不同的块的结构也不相同，文件系统类中还维护了几个成员变量用以监控文件系统状态，导致编写代码的时候很容易出现一些低级错误，比如打错结构体成员的名称，忘记更新某些状态变量等等，由于无法单步调试这种错误还不容易发现，花费了大量的时间。

至于文件系统结构的设计，虽然我一开始像就照着 UNIX 文件系统实现，但是由于 UNIX 文件系统根据文件大小不同使用不同的索引，这就意味着对文件处理需要处理三种索引，加大了工作量，所以我还是选择了简单的串联结构文件，对空闲块的管理也使用了空间利用率较低的布尔数组。

为了提高编写代码的效率，我的很多功能都是通过复用一些函数实现的，但是由于顶层设计没有设计好，代码之间的耦合程度还是很高的，导致一些函数在某些功能中使用的起始并不是那么自然，最突出的就是一些函数判断了权限或者存在性，另一些没有，一些函数会输出报错信息，另一些没有。结果就是到后面发现了一些 bug 之后找起来非常痛苦，这也让我知道了写代码前先设计好总体框架的重要性，磨刀不误砍柴工。

最后，此次任务对我的帮助非常大，不仅让我更加深刻的理解了文件系统的组成，还锻炼了我设计数据结构和代码工程的能力。

## 附录 实验代码

```
def.h
#ifndef DEF_H
```

```

#define DEF_H

#include<cstring>
#define MAX_FILE_NUM 65536
#define MAX_BLOCK_NUM 196000
// #define BLOCK_SIZE 516
#define BLOCK_SIZE 36
#define __IS_FOLDER__ 1
#define __IS_FILE__ 0
#define __ONLY_OWNER__ 0
#define __EVERYONE__ 1
typedef struct inode
{
    char i_uid[44]; //owner of the file
    int i_size; //size of the file
    int i_mode; //0:only owner 1:everyone
    int first_block;
    int last_block;
    bool i_type; //file:0,folder:1
    int block_num
;   inode():i_size(-1),i_mode(-1),first_block(-1),last_block(-1),i_type(0),block_num(
0){
    }
}inode;
typedef struct menu{
    char name[12];
    int inode;
    menu():inode(-1){
        memset(name,0,sizeof(name));
    }
    menu(const char *s,int n){
        int i=0;
        for( ; i<12&& s[i]!='\0';i++)
            name[i]=s[i];
        name[i]='\0';
        inode=n;
    }
}menu;
#endif // DEF_H

```

file\_system.h

```

#ifndef FILE_SYSTEM_H
#define FILE_SYSTEM_H

```

```

#include "def.h"
#include <string>
#include <vector>
using namespace std;
class file_system
{
    //system
    const int block_number;
    const int block_size;
    int free_nblock;
    int free_ninode;
    inode *inode_table;
    bool *block_used;
    const int BLOCK_OFFSET;
    const int MENU_SIZE;
    //user
    string pwd;
    string user;
    int pwd_menu_inode;
public:
    void debug(string name);
    file_system(string s);
    ~file_system();
    void start();
private:
    void help();
    int find_file_pwd(string name);
    bool check_pwd();
    bool check_file(int inode);
    void save(FILE* &fp);
    int creat_file(string name,int type);
    int remove_single_file(int target_file_inode,int index,FILE* fp);
    int remove_file(int target_file_inode,int index,FILE* fp);
    int read_to_s(string name,string& s);
    int write_from_s(string name,string s);
    vector<string> simplify_path(string path);

    int touch(string name);
    int rm(string name);
    int mkdir(string name);
    int read(string name);
    int write(string name);
    void ls();
    int cd(string dest);

```

```

int cp(string src,string tar);

string char_to_string(char *s);
void string_to_char(char *s, string t);
int find_free_inode();
int find_free_block();
int find_menuID(string fileName);
};

#endif // FILE_SYSTEM_H

```

login.h

```

#ifndef LOGIN_H
#define LOGIN_H
#include<string>
#include<unordered_map>
using namespace std;

class login
{
    unordered_map<string,string> users;
public:
    login();
    bool user_register();
    string user_login();
};

#endif // LOGIN_H

```

main.cpp

```

#include <iostream>
#include<login.h>
#include<file_system.h>
using namespace std;
int main() {
    cout<<endl;
    system("clear");
    int choice;
    string name,pass;
    login tool;
    while (true) {
        system("clear");

```



```

    if ((fp = fopen("user.ini", "r")) == nullptr) {
        printf("File client.txt could not be opened\n");
        fclose(fp);
        return;
    }
    char user[44];
    char pass[256];
    while (!feof(fp)) {
        fscanf(fp, "%s%s", user, pass);
        this->users[string(user)]=string(pass);
    }
    fclose(fp);
}

bool login::user_register()
{
    string name,pass;
    cout << "Please input your name:";
    cin >> name;
    cout << "Please input your password: ";
    cin >> pass;
    if(name.length()>43){
        cout<<"user's name is too long ,please limit length to 43 bytes!"<<endl;
        getchar();
        getchar();
        return false;
    }
    if(this->users.find(name)!=this->users.end()){
        cout<<"name has already existed!"<<endl;
        getchar();
        getchar();
        return false;
    }
    this->users[name]=pass;
    FILE *fp;
    if ((fp = fopen("user.ini", "r+")) == nullptr) {
        printf("File client.txt could not be opened\n");
        getchar();
        getchar();
        fclose(fp);
        return false;
    }
    for(auto& p:this->users){
        fprintf(fp,"%s %s\n",p.first.c_str(),p.second.c_str());
    }
}

```

```

    }
    fclose(fp);
    cout<<"sign up success!"<<endl;
    getchar();
    getchar();
    return true;
}

string login::user_login(){
    string name,pass;
    cout << "Please input your name:";
    cin >> name;
    cout << "Please input your password: ";
    cin >> pass;
    if(this->users.find(name)==this->users.end()){
        cout<<"sign up first!"<<endl;
        getchar();
        getchar();
        return "";
    }
    if(this->users[name]==pass){
        cout<<"success!"<<endl;
        getchar();
        getchar();
        return name;
    }
    return "";
}

```

file\_system.cpp

```

#include "file_system.h"
#include <cstdio>
#include<iostream>
#include<vector>
#include<cmath>
using namespace std;
file_system::file_system(string name):
    block_number(MAX_BLOCK_NUM),
    block_size(BLOCK_SIZE),
    BLOCK_OFFSET(sizeof
(int)*2+sizeof(inode)*MAX_FILE_NUM+sizeof(bool)*MAX_BLOCK_NUM),
    MENU_SIZE(sizeof(menu)),
    pwd("/"),user(name),pwd_menu_inode(0)
{
    this->inode_table=new inode[MAX_FILE_NUM];
}

```



```

this->block_used=new bool[MAX_BLOCK_NUM];
memset(this->block_used, 0,MAX_BLOCK_NUM);
FILE *fp = fopen("disk", "r");
if (fp == nullptr)
{
    this->free_nblock=MAX_BLOCK_NUM-1;
    this->free_ninode=MAX_FILE_NUM-1;
    FILE *newfs = fopen("disk", "w");
    //set root dir
    this->inode_table[0].i_type=__IS_FOLDER__;
    this->inode_table[0].i_mode=__EVERYONE__;
    this->inode_table[0].i_size=0;
    strcpy(this->inode_table[0].i_uid,"root");
    this->inode_table[0].first_block=this->inode_table[0].last_block=0;
    this->block_used[0]=true;
    this->inode_table[0].block_num=1;
    //save
    fwrite(&this->free_nblock,sizeof(int),1,newfs);
    fwrite(&this->free_ninode,sizeof(int),1,newfs);
    fwrite(this->inode_table, sizeof(inode), MAX_FILE_NUM, newfs);
    fwrite(this->block_used, sizeof(bool), MAX_BLOCK_NUM, newfs);
    fseek(newfs, 100 * 1024 * 1024 - 1, SEEK_SET);
    const char END_OF_FILE = EOF;
    fwrite(&END_OF_FILE, 1, 1, newfs);
    fclose(newfs);
}
else
{
    fread(&this->free_nblock, sizeof(int), 1, fp);
    fread(&this->free_ninode, sizeof(int), 1, fp);
    fread(this->inode_table, sizeof(inode), MAX_FILE_NUM, fp);
    fread(this->block_used, sizeof(bool), MAX_BLOCK_NUM, fp);
    fclose(fp);
}
cout << "load file system done." << endl;
}

file_system::~file_system(){
    delete this->inode_table;
    delete this->block_used;
}

void file_system::help(){
    cout << endl;
}

```

```

    cout << "Yizumi file system,version 1.0" << endl;
    cout << "touch <filename>\t\t\tCreate a new file named <filename>" << endl;
    cout << "mkdir <dirname>\t\t\tCreate a new folder named <dirname>" << endl;
    cout << "read <filename>\t\t\tread txt file named <filename>" << endl;
    cout << "write <filename>\t\t\twrite to txt file named <filename>" << endl;
    cout << "cp <src> <target>\t\t\tcopy <src> to <target>" << endl;
    cout << "rm <filename>\t\t\tremove file or folder named <filename>" << endl;
    cout << "ls\t\t\t\tlist all files and folders in current menu" << endl;
    cout << "cd <dirname>\t\t\tgo to folder named <dirname>" << endl;
    cout << "help\t\t\t\tlist command of Yizumi File System" << endl;
    cout << "exit\t\t\t\tback to login page" << endl;
    cout << endl;
    cout << endl;
}

bool file_system::check_pwd()
{
    return !(this->inode_table[this->pwd_menu_inode].i_mode ==
__ONLY_OWNER__ &&
strcmp(this->user.c_str(),this->inode_table[this->pwd_menu_inode].i_uid)!=0);
}

bool file_system::check_file(int inode){
    if(this->inode_table[inode].i_type==__IS_FILE__)
        return !(this->inode_table[inode].i_mode == __ONLY_OWNER__ &&
strcmp(this->user.c_str(),this->inode_table[inode].i_uid)!=0);
    return false;
}

int file_system::find_file_pwd(string name)
{
    if(name.length()==0)    return -1;
    FILE *fp = fopen("disk", "r");
    menu temp;
    int block_iter = inode_table[this->pwd_menu_inode].first_block;
    int menu_file_size_to_search = inode_table[pwd_menu_inode].i_size;
    int blocks_to_search = inode_table[pwd_menu_inode].block_num;
    int ans=-1;
    while (blocks_to_search)
    {
        fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * block_iter, SEEK_SET);
        for (int i = 0; i < min((BLOCK_SIZE - 4) / MENU_SIZE,
menu_file_size_to_search / MENU_SIZE); ++i)
        {

```

```

        fread(&temp, MENU_SIZE, 1, fp);
        if (temp.inode == -1) continue;
        string temp_file_name = string(temp.name);
        if (temp_file_name == name)
        {
            ans = temp.inode;
            break;
        }
    }
    if (ans != -1) break;
    blocks_to_search--;
    if (blocks_to_search!=0)
    {
        menu_file_size_to_search -= (BLOCK_SIZE - 4);
        fread(&block_iter, sizeof(int), 1, fp);
    }
}

fclose(fp);
return ans;
}

int file_system::find_free_inode(){
    for (int i = 0; i < MAX_FILE_NUM; ++i)
    {
        if (this->inode_table[i].i_size == -1)
            return i;
    }
    return -1;
}

int file_system::find_free_block()
{
    for (int i = 0; i < MAX_BLOCK_NUM; ++i){
        if (!this->block_used[i])
            return i;
    }
    return -1;
}

void file_system::save(FILE*& fp){
    fseek(fp, 0, SEEK_SET);
    fwrite(&this->free_nblock,sizeof(int),1,fp);
    fwrite(&this->free_ninode,sizeof(int),1,fp);
}

```

```

        fwrite(this->inode_table, sizeof(inode), MAX_FILE_NUM, fp);
        fwrite(this->block_used, sizeof(bool), MAX_BLOCK_NUM, fp);
        fclose(fp);
    }

int file_system::creat_file(string name,int type)
{
    if (name == "")
    {
        cout << "input a file name first!" << endl;
        return -1;
    }
    if(name.length()>11){
        cout<<"filename is too long ,please limit length to 11 bytes!"<<endl;
        return -1;
    }
    if(!check_pwd())
    {
        cout<<"Permission denied!"<<endl;
    }
    if (this->free_nblock==0 || this->free_nblock==0)
    {
        cout << "file system is full!" << endl;
        return -1;
    }
    if(this->find_file_pwd(name)!=-1)
    {
        cout<<name<<" has already exist!"<<endl;
        return -1;
    }
    int new_file_inode=find_free_inode();
    int new_file_block=find_free_block();
    this->free_nblock--;
    this->free_ninode--;
    this->inode_table[new_file_inode].block_num=1;
    this->inode_table[new_file_inode].i_size=0;
    this->block_used[new_file_block]=true;
    menu t(name.c_str(),new_file_inode);
    FILE *fp = fopen("disk", "r+");

    //add menu of new file to pwd
    int last_block_of_pwd=this->inode_table[this->pwd_menu_inode].last_block;
    int
    offset_in_last_block=inode_table[pwd_menu_inode].i_size%(this->block_size-4);

```

```

//need new block
if(inode_table[pwd_menu_inode].i_size!=0&&offset_in_last_block==0)
{
    if(this->free_nblock==0)
    {
        cout<<"file system is full!"<<endl;
        return -1;
    }
    int new_menu_block=find_free_block();
    this->block_used[new_menu_block]=true;
    this->free_nblock--;
    this->inode_table[this->pwd_menu_inode].block_num++;
    this->inode_table[this->pwd_menu_inode].last_block=new_menu_block;
    fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * last_block_of_pwd +
BLOCK_SIZE - 4, SEEK_SET);
    fwrite(&new_menu_block, sizeof(int), 1, fp);
    fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * new_menu_block,
SEEK_SET);
    fwrite(&t,MENU_SIZE,1,fp);
}
//not need new block
else
{
    fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * last_block_of_pwd +
offset_in_last_block, SEEK_SET);
    fwrite(&t,MENU_SIZE,1,fp);
}
this->inode_table[this->pwd_menu_inode].i_size+=MENU_SIZE;
//add menu to pwd success,now change filesystem argument and set inode of new
file
strcpy(this->inode_table[new_file_inode].i_uid,this->user.c_str());
this->inode_table[new_file_inode].i_mode=__ONLY_OWNER__;
this->inode_table[new_file_inode].i_type=type;
this->inode_table[new_file_inode].first_block=this->inode_table[new_file_inode]
.last_block=new_file_block;
save(fp);
cout<<"success!"<<endl;
return 0;
}

int file_system::remove_single_file(int target_file_inode,int index,FILE *fp){
    //copy old index without taget file
    vector<menu> copy;
    int block_iter = inode_table[index].first_block;

```

```

int menu_file_size_to_search = inode_table[index].i_size;
int blocks_to_search = inode_table[index].block_num;
while (blocks_to_search)
{
    this->free_nblock++;
    this->block_used[block_iter]=false;
    fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * block_iter, SEEK_SET);
    for (int i = 0; i < min((BLOCK_SIZE - 4) / MENU_SIZE,
menu_file_size_to_search / MENU_SIZE); ++i)
    {
        menu temp;
        fread(&temp, MENU_SIZE, 1, fp);
        if(temp.inode!=target_file_inode){
            copy.emplace_back(temp);
        }
    }
    blocks_to_search--;
    if (blocks_to_search!=0)
    {
        menu_file_size_to_search -= (BLOCK_SIZE - 4);
        fread(&block_iter, sizeof(int), 1, fp);
    }
} //copy complete
// for(menu m:copy){
//     cout<<m.name<<" "<<m.inode<<endl;
// }
//set new pwd inode
inode_table[index].i_size=copy.size()*MENU_SIZE;
inode_table[index].block_num=ceil(((double)(inode_table[index].i_size)/(this->blo
ck_size-4)));
if(inode_table[index].i_size==0){
    inode_table[index].block_num=1;
}
//write new pwd
int new_block_for_menu = find_free_block();
this->free_nblock--;
block_used[new_block_for_menu] = true;
inode_table[index].first_block = new_block_for_menu;
int offset = 0;
int menu_cnt=copy.size();
while (1)
{
    if (menu_cnt <= (BLOCK_SIZE - 4) / 4)
    {

```

```

        fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * new_block_for_menu,
SEEK_SET);
        fwrite(&copy[0] + offset, MENU_SIZE, menu_cnt, fp);
        inode_table[index].last_block = new_block_for_menu;
        break;
    }
    else
    {
        fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * new_block_for_menu,
SEEK_SET);
        fwrite(&copy[0] + offset, sizeof(int), (BLOCK_SIZE - 4) / 4, fp);
        menu_cnt -= (BLOCK_SIZE - 4) / 4;
        offset += (BLOCK_SIZE - 4) / 4;
        new_block_for_menu = find_free_block();
        this->free_nblock--;
        block_used[new_block_for_menu] = 1;
        fwrite(&new_block_for_menu, sizeof(int), 1, fp);
    }
}

//rm target file
block_iter=inode_table[target_file_inode].first_block;
int last_block_of_target_file = inode_table[target_file_inode].last_block;
while (1)
{
    this->free_nblock++;
    block_used[block_iter] = 0;
    if (block_iter == last_block_of_target_file)
    {
        break;
    }
    else
    {
        fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * block_iter +
BLOCK_SIZE - 4, SEEK_SET);
        fread(&block_iter, sizeof(int), 1, fp);
    }
}
this->free_ninode++;
inode_table[target_file_inode].i_size=-1;
return 0;
}

int file_system::remove_file(int target,int index,FILE *fp)
{

```

```

        if(inode_table[target].i_type==__IS_FILE__ ||
(inode_table[target].i_type==__IS_FOLDER__&&inode_table[target].i_size==0))
        {
            return this->remove_single_file(target,index,fp);
        }

        else
        {
            menu temp;
            int block_iter = inode_table[target].first_block;
            int menu_file_size_to_search = inode_table[target].i_size;
            //      cout<<block_iter<<endl;
            for (int i = 0; i < menu_file_size_to_search / MENU_SIZE; ++i)
            {
                fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * block_iter, SEEK_SET);
                fread(&temp, MENU_SIZE, 1, fp);
                //      cout<<temp.name<<" "<<temp.inode<<endl;
                if(this->remove_file(temp.inode,target,fp)!=0){
                    cout<<"remove "<<temp.name<<" failed! Something wrong
happened!"<<endl;
                    return -1;
                }
            }
            return this->remove_single_file(target,index,fp);
            return 0;
        }
    }

int file_system::read_to_s(string name,string& s)
{
    if (name == "")
    {
        cout << "input a file name first!" << endl;
        return -1;
    }
    if(name.length()>11)
    {
        cout<<"filename is too long ,please limit length to 11 bytes!"<<endl;
        return -1;
    }
    int target_file_inode=find_file_pwd(name);
    if(target_file_inode==-1)
    {
        cout<<"file not exist!"<<endl;
    }
}

```



```

        return -1;
    }
    if(!check_file(target_file_inode))
    {
        cout<<"permission denied!"<<endl;
        return -1;
    }
    int block_iter=inode_table[target_file_inode].first_block;
    int last_block=inode_table[target_file_inode].last_block;
    char buf[this->block_size];
    FILE *fp = fopen("disk", "r");
    s="";
    while(1)
    {
//        cout<<block_iter<<endl;
        if(block_iter==last_block)
        {
            int offset=inode_table[target_file_inode].i_size%(this->block_size-4);
            if(offset==0){
                offset=this->block_size;
            }
            fseek(fp,BLOCK_OFFSET+this->block_size*block_iter,SEEK_SET);
            memset(buf,0,sizeof(buf));
            fread(buf,sizeof(char),offset,fp);
//            cout<<buf<<endl;
            s+=buf;
            break;
        }
        else
        {
            fseek(fp,BLOCK_OFFSET+this->block_size*block_iter,SEEK_SET);
            memset(buf,0,sizeof(buf));
            fread(buf,sizeof(char),this->block_size-4,fp);
//            cout<<buf<<endl;
            s+=buf;
            fread(&block_iter,sizeof(int),1,fp);
        }
    }
    save(fp);
    return 0;
}

int file_system::write_from_s(string name,string s)
{

```

```

if (name == "")
{
    cout << "input a file name first!" << endl;
    return -1;
}
if(name.length()>11)
{
    cout<<"filename is too long ,please limit length to 11 bytes!"<<endl;
    return -1;
}
int target_file_inode=find_file_pwd(name);
if(target_file_inode==-1)
{
    cout<<"file not exist!"<<endl;
    return -1;
}
if(!check_file(target_file_inode))
{
    cout<<"permission denied!"<<endl;
    return -1;
}
const char *buf=s.c_str();
int len=s.length();
int offset=this->inode_table[target_file_inode].i_size%(BLOCK_SIZE-4);
int max_free_byte = BLOCK_SIZE - 4 - offset + this->free_nblock *
(BLOCK_SIZE - 4);
if(max_free_byte< len ){
    cout<<"file system is full!"<<endl;
    return -1;
}
FILE *fp = fopen("disk", "r+");
inode_table[target_file_inode].i_size += len;
int char_pos=0;
while(len>0)
{
    if (len <= BLOCK_SIZE - offset - 4)
    {
        fseek(fp, BLOCK_OFFSET + BLOCK_SIZE *
this->inode_table[target_file_inode].last_block + offset, SEEK_SET);
        fwrite(buf + char_pos, sizeof(char), len, fp);
        break;
    }
    else
    {

```

```

        fseek(fp, BLOCK_OFFSET + BLOCK_SIZE *
this->inode_table[target_file_inode].last_block + offset, SEEK_SET);
        fwrite(buf + char_pos, sizeof(char), BLOCK_SIZE - offset - 4, fp);
        char_pos+=BLOCK_SIZE - offset - 4;
        int new_block=this->find_free_block();
//        cout<<new_block<<endl;
        fwrite(&new_block,sizeof(int),1,fp);
        inode_table[target_file_inode].block_num++;
        inode_table[target_file_inode].last_block=new_block;
        this->free_nblock--;
        this->block_used[new_block]=true;
        len-=BLOCK_SIZE - offset - 4;
        offset=0;
    }
}
save(fp);
return 0;
}

vector<string> file_system::simplify_path(string path) {
    auto split = [](const string& s, char delim) -> vector<string> {
        vector<string> ans;
        string cur;
        for (char ch: s) {
            if (ch == delim) {
                ans.push_back(move(cur));
                cur.clear();
            }
            else {
                cur += ch;
            }
        }
        ans.push_back(move(cur));
        return ans;
    };

    vector<string> names = split(path, '/');
    vector<string> stack;
    for (string& name: names) {
        if (name == "..") {
            if (!stack.empty()) {
                stack.pop_back();
            }
        }
    }
}

```

```

        else if (!name.empty() && name != ".") {
            stack.push_back(move(name));
        }
    }
    return stack;
}

int file_system::touch(string name){
    return this->creat_file(name,__IS_FILE__);
}

int file_system::rm(string name){
    if (name == "")
    {
        cout << "input a file name first!" << endl;
        return -1;
    }
    if(name.length()>11){
        cout<<"filename is too long ,please limit length to 11 bytes!"<<endl;
        return -1;
    }
    int target=find_file_pwd(name);
    if(target==-1)
    {
        cout<<"file not exist!"<<endl;
        return -1;
    }
    FILE* fp=fopen("disk","r+");
    int ret = this->remove_file(target,this->pwd_menu_inode,fp);
    save(fp);
    return ret;
}

int file_system::mkdir(string name){
    return this->creat_file(name,__IS_FOLDER__);
}

int file_system::read(string name){
    string buf;
    int res=this->read_to_s(name,buf);
    if(res!=0){
        return res;
    }
}

```

```

        cout<<buf<<endl;
        return 0;
    }

int file_system::write(string name)
{
    if(find_file_pwd(name)==-1){
        cout<<"file not exist!"<<endl;
        return -1;
    }
    string s;
    cout << "please input your text" << endl;
    getline(cin,s);
    return this->write_from_s(name,s);
}

void file_system::ls()
{
    FILE *fp = fopen("disk", "r");
    menu temp;
    int block_iter = inode_table[this->pwd_menu_inode].first_block;
    int menu_file_size_to_search = inode_table[pwd_menu_inode].i_size;
    int blocks_to_search = inode_table[pwd_menu_inode].block_num;
    // cout<<"menu size:"<<menu_file_size_to_search<<endl;
    // cout<<"menu blocks:"<<blocks_to_search<<endl;
    while (blocks_to_search)
    {
        // cout<<"block iter:"<<block_iter<<endl;
        fseek(fp, BLOCK_OFFSET + BLOCK_SIZE * block_iter, SEEK_SET);
        for (int i = 0; i < min((BLOCK_SIZE - 4) / MENU_SIZE,
menu_file_size_to_search / MENU_SIZE); ++i)
        {
            fread(&temp, MENU_SIZE, 1, fp);
            string temp_file_name = string(temp.name);
            if(inode_table[temp.inode].i_type==__IS_FILE__)
                cout<<temp_file_name<<endl;
            else
                cout << "\033[34m"<<temp_file_name<<"\033[0m"<< endl ;
        }
        blocks_to_search--;
        if (blocks_to_search!=0)
        {
            menu_file_size_to_search -= (BLOCK_SIZE - 4);
            fread(&block_iter, sizeof(int), 1, fp);
        }
    }
}

```

```

    }
}

fclose(fp);
}

int file_system::cd(string dest)
{
    int old_pwd_inode=this->pwd_menu_inode;
    string old_path=this->pwd;
    if(dest.substr(0,2)== "./")
        dest=pwd+dest.substr(1);
    else if(dest.substr(0,2)== ".."){
        if (pwd == "/")
            return 0;
        for (int i = pwd.length(); i >= 0; --i)
        {
            if (pwd[i] == '/')
            {
                if (i == 0) pwd = "/";
                else pwd.erase(i);
                break;
            }
        }
        dest=pwd+dest.substr(2);
    }
    else if(dest[0]!='/')
        dest=pwd+'/'+dest;
    // cout<<dest<<endl;
    vector<string> path=this->simplify_path(dest);
    this->pwd="";
    // for(string s:path){
    //     cout<<s<<" ";
    // }
    cout<<endl;
    if (path.empty()) {
        this->pwd = "/";
        this->pwd_menu_inode=0;
    }
    else {
        this->pwd_menu_inode=0;
        for (string& name: path) {
            this->pwd += "/" + name;
            int temp=this->find_file_pwd(name);
        }
    }
}

```

```

        if(inode_table[temp].i_type!=__IS_FOLDER__){
            cout<<"path not exist!"<<endl;
            this->pwd=old_path;
            this->pwd_menu_inode=old_pwd_inode;
            return -1;
        }
        this->pwd_menu_inode=temp;
    }
}
return 0;
}

```

```

int file_system::cp(string src,string target){
//    cout<<src<<" " <<target<<endl;
    string filename=src;
    int old_menu_inode=this->pwd_menu_inode;
    string old_pwd=this->pwd;
    bool is_file_in_pwd=true;
    for (int i = src.length(); i >= 0; --i)
    {
        if (src[i] == '/')
        {
            is_file_in_pwd=false;
            filename=src.substr(i+1);
            src.erase(i);
            break;
        }
    }
    if(!is_file_in_pwd){
        if(this->cd(src)!=0){
            cout<<"open source file failed!"<<endl;
            return -1;
        }
    }
    string content;
    if(this->read_to_s(filename,content)!=0){
        cout<<"open source file failed!"<<endl;
        return -1;
    }
//    cout<<content<<endl;
    this->pwd=old_pwd;
    this->pwd_menu_inode=old_menu_inode;
    filename=target;
    is_file_in_pwd=true;
}

```

```

for (int i = src.length(); i >= 0; --i)
{
    if (target[i] == '/')
    {
        is_file_in_pwd=false;
        filename=target.substr(i+1);
        target.erase(i);
        break;
    }
}
if(!is_file_in_pwd){
    if(this->cd(target)!=0){
        cout<<"open source file failed!"<<endl;
        return -1;
    }
}
if(this->touch(filename)!=0){
    cout<<"create target file failed!"<<endl;
    return -1;
}
if(this->write_from_s(filename,content)!=0)
{
    cout<<"open target file failed!"<<endl;
    return -1;
}
this->pwd=old_pwd;
this->pwd_menu_inode=old_menu_inode;
return 0;
}

void file_system::start(){
    system("clear");
    cout<<"Welcome to Yizumi File System,input \"help\" to get the usage of the
system"<<endl;
    while (1)
    {
        cout << endl;
        cout << "\033[32m"<<this->user<<" @ YizumiFileSystem\033[0m"<<":"<<
"\033[34m"<<pwd<<"\033[0m" << "$ ";
        string order;
        string opt = "";
        getline(cin, order);
        if (order == "") continue;
        int tempLoc = -1;

```



```

int len=order.length();
for (int i = 0; i < len; ++i)
    if (order[i] == ' ')
    {
        tempLoc = i;
        break;
    }
    else opt += order[i];

string temp = "";
if (opt == "touch")
{
    if (tempLoc != -1)
        temp = order.substr(tempLoc + 1, order.length() - tempLoc);
    touch(temp);
}
else if(opt=="mkdir"){
    if (tempLoc != -1)
        temp = order.substr(tempLoc + 1, order.length() - tempLoc);
    mkdir(temp);
}
else if (opt == "rm")
{
    if (tempLoc != -1)
        temp = order.substr(tempLoc + 1, order.length() - tempLoc);
    rm(temp);
}
else if (opt == "read")
{
    if (tempLoc != -1)
        temp = order.substr(tempLoc + 1, order.length() - tempLoc);
    read(temp);
}
else if (opt == "write")
{
    if (tempLoc != -1)
        temp = order.substr(tempLoc + 1, order.length() - tempLoc);
    write(temp);
}
else if (opt == "cp")
{
    if (tempLoc == -1)
        cout << "cp: missing file operand" << endl;
    else

```

```

        {
            int _tempLoc = -1;
            for (int i = tempLoc + 2; i < len; ++i)
                if (order[i] != ' ' && order[i - 1] == ' ')
                {
                    _tempLoc = i - 1;
                    break;
                }
            if (_tempLoc == -1)
                cout << "cp: missing file operand" << endl;
            else
            {
                string tempS = order.substr(tempLoc + 1, _tempLoc - tempLoc -
1);
                string tempT = order.substr(_tempLoc + 1, order.length() -
_tempLoc);
                cp(tempS, tempT);
            }
        }
    }
    else if (opt == "ls")
    {
        ls();
    }
    else if (opt == "cd")
    {
        if (tempLoc == -1)
        {
            pwd = "/";
            this->pwd_menu_inode = 0;
        }
        else
        {
            if (tempLoc != -1)
                temp = order.substr(tempLoc + 1, order.length() - tempLoc);
            cd(temp);
        }
    }
    else if (opt == "exit")
    {
        break;
    }
    else if (opt == "help")
    {

```

```

        help();
    }
    else if(opt == "d"){
        if (tempLoc != -1)
            temp = order.substr(tempLoc + 1, order.length() - tempLoc);
        debug(temp);
    }
    else
    {
        cout << opt << ": command not found" << endl;
    }
}
cout << endl;
}

void file_system::debug(string name){
    cout<<this->free_nblock<<" "<<MAX_BLOCK_NUM<<endl;
    cout<<this->free_ninode<<" "<<MAX_FILE_NUM<<endl;
    cout<<"pwd info:"<<endl;
    cout<<"inode:"<<this->pwd_menu_inode<<endl;
    cout<<"first block:"<<inode_table[pwd_menu_inode].first_block<<endl;
    cout<<"last block:"<<inode_table[pwd_menu_inode].last_block<<endl;
    cout<<"top 20 block"<<endl;
    for(int i=0 ; i<20;i++){
        cout<<block_used[i]<<" ";
    }
    cout<<endl;
    cout<<"file "<<name<<" : "<<endl;
    int index=find_file_pwd(name);
    if(index==-1)    cout<<"not exist"<<endl;
    else{
        cout<<"type:"<<inode_table[index].i_type<<endl;;
        cout<<"first block:"<<inode_table[index].first_block<<endl;
        cout<<"block number:"<<inode_table[index].block_num<<endl;
        cout<<"size:"<<inode_table[index].i_size<<endl;
    }
}
}

```