



# 华中科技大学

## 操作系统原理课程设计报告

姓 名： 沈承磊  
学 院： 计算机科学与技术学院  
专 业： 计算机科学与技术  
班 级： CS 1903  
学 号： U201914376  
指导教师： 胡 侃

分数	
教师签名	

2022 年 3 月 7 日

# 目 录

<b>1. 实验一 Linux 下编程相关知识 .....</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 .....	4
1.4.1 实验步骤 .....	4
1.4.2 实验调试及心得 .....	5
附录 实验代码 .....	7
<b>2. 实验二 系统调用相关知识 .....</b>	<b>18</b>
2.1 实验目的 .....	18
2.2 实验内容 .....	18
2.3 实验设计 .....	18
2.3.1 开发环境 .....	18
2.3.2 实验设计 .....	18
2.4 实验调试 .....	19
2.4.1 实验步骤 .....	19
2.4.2 实验调试及心得 .....	21
附录 实验代码 .....	21
<b>3 实验三 增加设备驱动程序 .....</b>	<b>24</b>
3.1 实验目的 .....	24
3.2 实验内容 .....	24
3.3 实验设计 .....	24
3.3.1 开发环境 .....	24
3.3.2 实验设计 .....	24
3.4 实验调试 .....	25
3.4.1 实验步骤 .....	25
3.4.2 实验调试及心得 .....	27
附录 实验代码 .....	27
<b>4 实验四 使用 QT 实现系统监视器 .....</b>	<b>31</b>
4.1 实验目的 .....	31
4.2 实验内容 .....	31
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 .....	40
4.4.1 实验步骤 .....	40

4.4.2 实验调试及心得.....	40
附录 实验代码.....	43
<b>5 实验五 小型文件系统.....</b>	<b>45</b>
5.1 实验目的.....	45
5.2 实验内容.....	45
5.3 实验设计.....	46
5.3.1 开发环境.....	46
5.3.2 实验设计.....	46
5.4 实验调试.....	51
5.4.1 实验步骤.....	51
5.4.2 实验调试及心得.....	54
附录 实验代码.....	54

# 1. 实验一 Linux 下编程相关知识

## 1.1 实验目的

掌握 Linux 操作系统的基本使用方法，包括键盘命令、系统调用等方法；熟悉 Linux 下的编程环境，使用 Linux 进行 C 语言编程。

## 1.2 实验内容

- a) 编一个 C 程序，其内容为实现文件拷贝的功能。
- b) 基本要求:使用系统调用 `open/read/write...`；选择:容错、`cp`。
- c) 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。(gtk/Qt)
- d) 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

1. 硬件环境：
  - a) 中央处理器 CPU：AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
  - b) 物理内存 RAM：16GB
2. 编译环境：
  - a) 虚拟机软件：VMware Workstation 16 Pro
  - b) 虚拟机系统版本：Ubuntu14.04 操作系统
  - c) 内核：Linux4.4.10
  - d) 虚拟机核数：8
  - e) 虚拟机内存：4.00 GB
  - f) 虚拟机磁盘：60 GB
  - g) 编译器版本：GCC 7.5.0
  - h) 调试器版本：GNU gdb 8.1.0

### 1.3.2 实验设计

**任务 1：实现 `cp` 命令/文件拷贝的功能：**

任务 1 的要求是使用系统功能调用实现文件拷贝功能，首先，使用 `open` 系统调用函数打开文件，然后使用 `read` 和 `write` 系统功能调用函数对文件进行读取与写入，最后使用 `close` 系统功能调用关闭两个文件即可实现拷贝功能。

## 1. 了解 read, write, open 系统调用函数的使用

### a) 文件描述符

每一个进程都有一个与之相关的文件描述符，它们是一些小值整数，我们可以通过这些文件描述符来访问打开的文件。

一般地，一个程序开始运行时，会自动打开 3 个文件描述符：

0—— - 标准输入———stdin;

1—— - 标准输出———stdout;

2—— - 标准错误———stderr。

### b) open 系统调用

open 可以创建自定义文件的文件描述符，供其他系统调用函数（eg. write, read）来使用。

Open 系统调用原型：

#### i. `int open(const *path, int oflags);`

将准备打开的文件或是设备的名字（带扩展名）作为参数 path 传给函数，oflags 用来指定文件访问模式。open 系统调用成功返回一个新的文件描述符，失败返回-1。

其中，oflags 是由必需文件访问模式和可选模式一起构成的（通过按位或“|”）：

必需文件访问模式有：

O\_RDONLY———以只读方式打；

O\_WRONLY———以只写方式打开；

O\_RDWR———以读写方式打开。

可选文件访问模式有：

O\_CREAT———按照参数 mode（见第二种调用方法）给出的访问模式创建文件；

O\_EXCL——— - 与 O\_CREAT 一起使用，确保创建出文件，避免两个程序同时创建同一个文件，如文件存在则 open 调用失败；

O\_APPEND———把写入数据追加在文件的末尾；

#### ii. `int open(const *path, int oflags, mode_t mode);`

在第一种调用方式上，加上了第三个参数 mode，主要是搭配 O\_CREAT 使用，同样地，这个参数规定了属主、同组和其他人对文件的文件操作权限。

S\_IRUSR———读权限

S\_IWUSR———写权限 ——文件属主

S\_IXUSR———执行权限

S\_IRGRP———读权限  
S\_IWGRP———写权限 ——文件所属组  
S\_IXGRP———执行权限  
S\_IROTH———读权限  
S\_IWOTH———写权限——其他人  
S\_IXOTH———执行权限

常用数字设定法:

0———无权限;  
1———只执行;  
2———只写;  
4———只读。

Eg. 0600 表示文件属主可写可读。

### c) read/write 系统调用

read 系统调用, 是从与文件描述符 `flides` 相关联的文件中读取前 `nbytes` 字节的内容, 并且写入到数据区 `buf` 中。read 系统调用返回的是实际读入的字节数

write 系统调用, 是把缓存区 `buf` 中的前 `nbytes` 字节写入到与文件描述符 `flides` 有关的文件中, write 系统调用返回的是实际写入到文件中的字节数。

Read 系统调用原型:

```
size_t read(int flides, void *buf, size_t nbytes);
```

Write 系统调用原型:

```
size_t write(int flides, const void *buf, size_t nbytes);
```

## 2. 权限设置:

为使得文件拷贝功能更加灵活方便, 使用命令行参数的方式输入拷贝的源文件与目标文件名。由于源文件只需要读操作, 因此, 使用 `O_RDONLY` 作为 `flag`, 目标文件如果不存在需要进行创建, 因此需要使用 `O_WRONLY|O_CREAT` 作为 `flag`, 文件权限设置为 `0777`, 表示文属主、用户组、其他人可读可写可执行。

然后, 使用单层循环的方式对文件进行拷贝, 每次循环读, 再在循环体内将所读内容写入目的文件。需要注意的是, 在拷贝时需要对文件操作进行异常检测如 图 1-1, 以免出现读写错误。本次实验中采用了 `errno` 和 `strerror` 系统调用函数来进行异常检测以及错因分析, 前者会返回错因对应的序号, 后者将错误原因序号转化为字符串, 此函数在 debug 时非常有用。

```

while( (lenr = read(fread,bufs,1024)) > 0){
    if((lenw = write(fwrite,bufs,lenr))<0)break;
}
if(lenr<0){
    printf("读文件失败! \n");
    printf("失败原因是%s",strerror(errno));
}
else if(lenw<0){
    printf("写文件失败! \n");
    printf("失败原因是%s",strerror(errno));
}
else printf("copy完成!\n");

```

图 1-1 文件拷贝核心代码

### 任务 2：分窗口显示三个并发进程运行的结果：

任务 2 要求编一个 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 (gtk/Qt)。

本次任务选择使用 QT 进行图形化编程进行五个进程的显示，分别是文件拷贝以及四种不同的资源管理器窗口。使用 fork 函数创建子进程，使用 execv 函数替换程序段，打开新程序即可实现多窗口。使用 QT 自带的 ui 编程进行窗口编程，使用 QT 自带的窗口控件对文本和数据进行显示与读取。

其主要的实现过程就是去获取 Linux 下面的系统资源文件，并通过 QT 里的 slot 传到前端，并且隔固定的时间刷新一次

具体代码见附录。

## 1.4 实验调试

### 1.4.1 实验步骤

#### 任务 1：实现 cp 命令/文件拷贝的功能

在 Linux 平台下，使用 VS Code 进行代码编辑，然后在命令行中使用 gcc 命令进行编译。本次实验实现的 cp 命令使用命令行进行源文件名与目标文件名参数的输入，因此需在终端输入“./cp 源文件 目标文件”来进行文件拷贝。

#### 任务 2：分窗口显示多个并发进程运行结果

使用 QT 进行程序的编辑、编译与运行，并进行窗口显示。使用 QT 创建项目，编写代码并进行测试，使用 fork 函数创建子进程，execv 函数替换代码。

具体步骤如下：1.从系统资源文件中获取 CPU MEM TIME DISK 等信息 2.每 500 毫秒刷新一次，并通过 slot 传输到 QT 3.编写 main 函数管理进程 4.编写 CPU 信息界面、编写 MEM 信息界面、编写 TIME 信息界面、编写 DISK 信息界面、套用任务一代码并编写 CP 界面 5.通过 QString 的 setText 方法来给以上几个界面赋值。

# 1.4.2 实验调试及心得

## 任务 1：实现 cp 命令/文件拷贝的功能

使用 gcc 命令编译 cp.c 文件，生成可执行文件 cp，在命令行进行测试。按照实验步骤进行测试，如图 1-2 所示，当输入的参数个数不正确时，系统会报错“参数参数数量有误”；当输入的源文件不存在时，系统通过系统调用函数 strerror 打印出相关错误；如果拷贝成功，系统打印“COPY 成功!”。

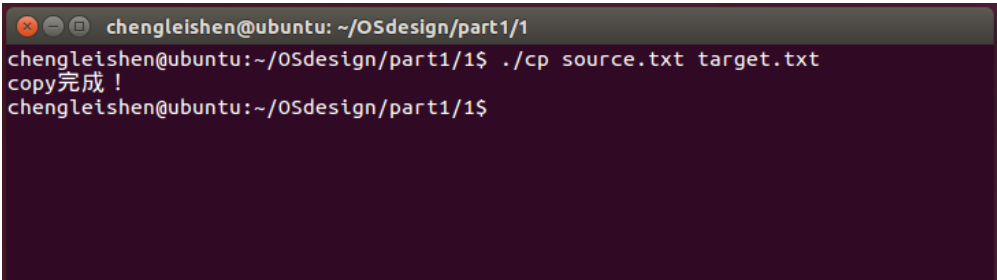


图 1-2 文件拷贝命令行效果图成功

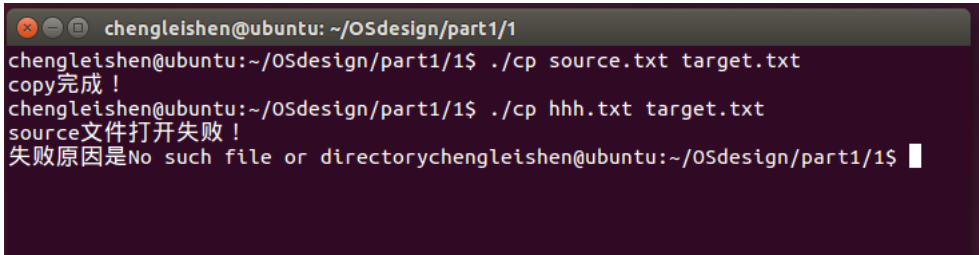


图 1-3 文件拷贝命令行效果图 error

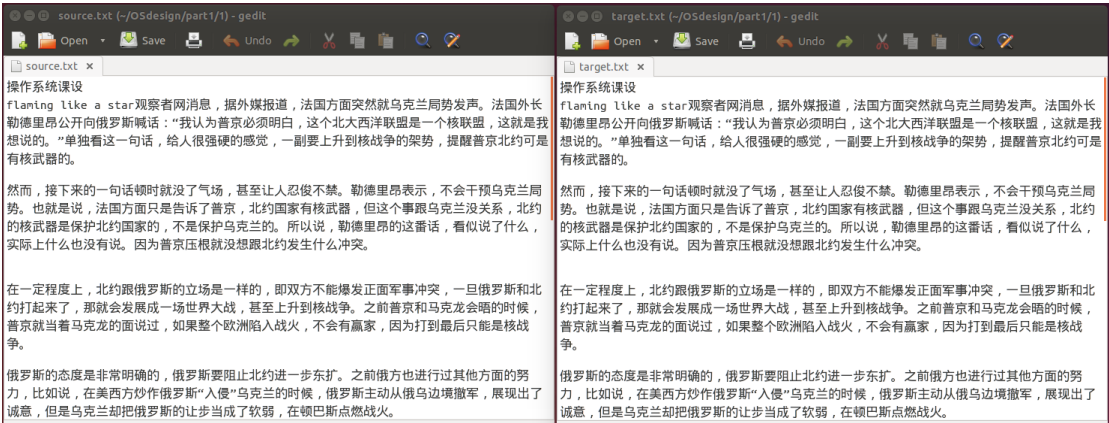


图 1-4 文件拷贝内容比对

## 任务 2：分窗口显示多个并发进程运行结果

左边窗口是文件拷贝过程监视窗口，从上方的两个文本输入框输入源文件与目标文件后，点击 START 按钮即可开始拷贝，进度条会显示拷贝进度，下方的提示窗口会显示提示信息，点击 CANCEL 暂停拷贝，点击 END 即可关闭窗口。

右边窗口是四个监视窗口，显示内容为进程号，CPU 使用情况内存使用情况，磁盘使用情况，系统时间。



```

1  #include "mainwindow.h"
2  #include <QApplication>
3  #include <QTextCodec>
4  #include "res_pro.h"
5  #include "res_disk.h"
6  #include "res_mem.h"
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <unistd.h>
10 #include "widget.h"
11 int main(int argc, char *argv[])
12 {
13     int pid;
14     if((pid = fork()) == 0){
15         QApplication a(argc,argv);
16         res_pro w;
17         w.setWindowTitle("CPU Monitor");
18         w.show();
19         a.exec();
20         exit(0);
21     }
22     if((pid = fork()) == 0){
23         QApplication a(argc,argv);
24         res_disk w;
25         w.setWindowTitle("DISK Monitor");
26         w.show();
27         a.exec();
28         exit(0);
29     }
30     if((pid = fork()) == 0){
31         QApplication a(argc,argv);

```

图 1-5 多进程主程序代码展示

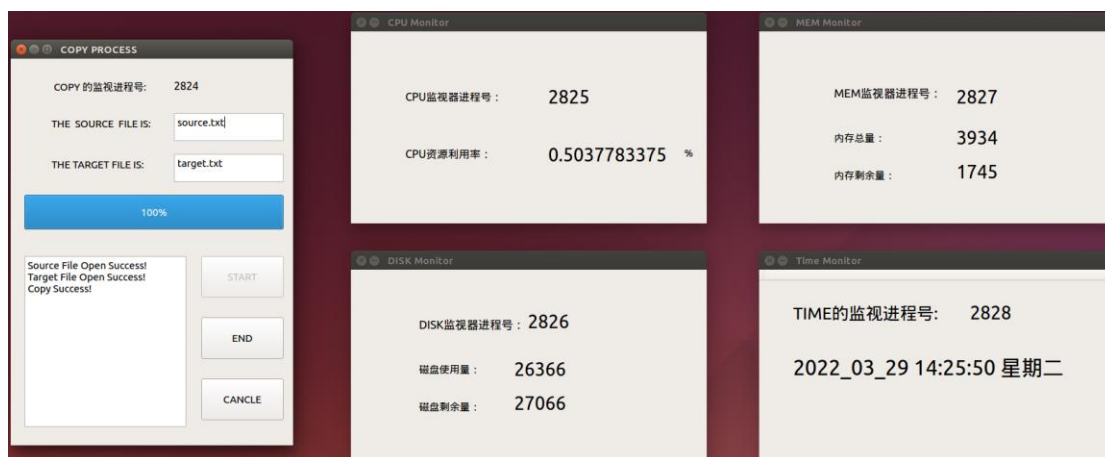


图 1-6 多进程界面效果展示

通过本次实验有以下收获：

- (1) 了解到了在 Linux 下进行编程的基本知识
- (2) 学习到了在 Linux 下终端的使用方法
- (3) 学习到了 QT 程序编程的基础
- (4) 巩固了在上学期操作系统课程设计当中的多进程与多线程的内容

(5) 学习到了 Linux 系统文件结构

## 附录 实验代码

### 1. 任务 1: 实现 cp 命令/文件拷贝的功能

```
2. #include <sys/types.h>
3. #include <sys/stat.h>
4. #include <unistd.h>
5. #include <fcntl.h>
6. #include <stdio.h>
7. #include <stdlib.h>
8. #include <string.h>
9. #include <errno.h>
10. int main(int argc, char* argv[]){
11.     extern int errno;
12.     int fread, fwrite;
13.     char bufs[1024];
14.     int lenr, lenw;
15.     //argc 表示参数个数, 默认为 1
16.     //argv[0] 为程序位置
17.     if(argc != 3) //传入参数数量不等于 3
18.     {
19.         printf("传入参数数量有误\n");
20.     }
21.     else
22.     {
23.         fread = open(argv[1], O_RDONLY);
24.         //argv[1] 为 source 文件
25.         if(fread == -1){
26.             printf("source 文件打开失败! \n");
27.             printf("失败原因是%s", strerror(errno));
28.             // }
29.
30.         else{//打开 source 文件了
31.             fwrite = open(argv[2], O_WRONLY|O_CREAT, 0777);
32.             if(fwrite == -1){
33.                 printf("target 文件打开或创建失败! \n");
34.                 printf("失败原因是%s", strerror(errno));
35.             }
36.
37.             else{//打开 target 文件了
38.                 while( (lenr = read(fread, bufs, 1024)) > 0){
```

```

39.             if((lenw = write(fwrite,bufs,lenr))<0)break;
40.         }
41.         if(lenr<0){
42.             printf("读文件失败! \n");
43.             printf("失败原因是%s",strerror(errno));
44.         }
45.         else if(lenw<0){
46.             printf("写文件失败! \n");
47.             printf("失败原因是%s",strerror(errno));
48.         }
49.         else printf("copy 完成! \n");
50.     }
51. }
52. }
53. close(fread);
54. close(fwrite);
55. return 0;
56. }

```

## 2: 分窗口显示多个并发进程运行结果

```

1. #include "mainwindow.h"
2. #include <QApplication>
3. #include <QTextCodec>
4. #include "res_pro.h"
5. #include "res_disk.h"
6. #include "res_mem.h"
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <unistd.h>
10. #include "widget.h"
11. int main(int argc, char *argv[])
12. {
13.     int pid;
14.     if((pid = fork()) == 0){
15.         QApplication a(argc,argv);
16.         res_pro w;
17.         w.setWindowTitle("CPU Monitor");
18.         w.show();
19.         a.exec();
20.         exit(0);
21.     }
22.     if((pid = fork()) == 0){
23.         QApplication a(argc,argv);
24.         res_disk w;

```

```

25.         w.setWindowTitle("DISK Monitor");
26.         w.show();
27.         a.exec();
28.         exit(0);
29.     }
30.     if((pid = fork()) == 0){
31.         QApplication a(argc,argv);
32.         res_mem w;
33.         w.setWindowTitle("MEM Monitor");
34.         w.show();
35.         a.exec();
36.         exit(0);
37.     }
38.     if((pid = fork()) == 0){
39.         QApplication a(argc, argv);
40.         MainWindow w;
41.         w.setWindowTitle("Time Monitor");
42.         w.show();
43.         a.exec();
44.         exit(0);
45.
46.     }
47.     QApplication a(argc,argv);
48.     Widget w;
49.     w.setWindowTitle("COPY PROCESS");
50.     w.show();
51.     return a.exec();
52. }

```

### 3: 分窗口代码

#### disk:

```

1. #include "res_disk.h"
2. #include "sys/statfs.h"
3. #include "ui_res_disk.h"
4. #include <QTimer>
5. #include <unistd.h>
6.
7. res_disk ::res_disk(QWidget *parent):
8.     QWidget (parent),
9.     ui(new Ui::res_disk)
10. {
11.     ui->setupUi(this);
12.     this->setFixedSize(this->width(),this->height());
13.     this->move(600,550);
14.

```

```

15.     int pid = getpid();// get current pid
16.     ui->Disk->setText(QString::number(pid,10));
17.
18.     Disk_Used = 0;
19.     Disk_Free = 0;
20.     ui->Disk_Monitor->setText(QString::number(Disk_Used,'f',0));
21.     ui->Disk_Free->setText(QString::number(Disk_Free,'f',0));
22.
23.
24.     QTimer*timer = new QTimer (this);
25.     connect(timer,SIGNAL(timeout()),this,SLOT(Update()));
26.     timer->start(1000);
27.
28. }
29.
30. void res_disk::Update()
31. {
32.     QProcess process;
33.     process.start("df -k");
34.     process.waitForFinished();
35.     process.readLine();
36.     while(!process.atEnd())
37.     {
38.         QString str = process.readLine();
39.         if(str.startsWith("/dev/sda"))
40.         {
41.             str.replace("\n","");
42.             str.replace(QRegExp("( ){1,}")," ");
43.             auto lst = str.split(" ");
44.             if(lst.size() > 5){
45.                 Disk_Used = lst[2].toDouble()/1024.0;
46.                 ui->Disk_Monitor->setText(QString::number(Disk_Used,'
f',0));
47.                 Disk_Free = lst[3].toDouble()/1024.0;
48.                 ui->Disk_Free->setText(QString::number(Disk_Free,'f',
0));
49.             }
50.             //qDebug("Disk Used:%.01fMB Free:%.01fMB",lst[2].toDo
uble()/1024.0,lst[3].toDouble()/1024.0);
51.         }
52.     }
53. }
54. res_disk::~res_disk()
55. {

```

```

56.     delete ui;
57. }

```

### Memory:

```

1. #include "res_mem.h"
2. #include "sys/statfs.h"
3. #include "ui_res_mem.h"
4. #include <QTimer>
5. #include <unistd.h>
6.
7. res_mem::res_mem(QWidget *parent):
8.     QWidget (parent),
9.     ui(new Ui::res_mem)
10. {
11.     ui->setupUi(this);
12.     this->setFixedSize(this->width(),this->height());
13.     this->move(1200,200);
14.
15.     int pid = getpid();
16.     ui->Mem->setText(QString::number(pid,10));
17.
18.     Mem_Used = 0;
19.     Mem_Free = 0;
20.     ui->Mem_Monitor->setText(QString::number(Mem_Used,'f',0));
21.     ui->Mem_Free->setText(QString::number(Mem_Free,'f',0));
22.
23.
24.     QTimer*timer = new QTimer (this);
25.     connect(timer,SIGNAL(timeout()),this,SLOT(Update()));
26.     //the timer will send a signal every 1000ms, and then to execute
    Updata().
27.     timer->start(1000);
28.
29. }
30. void res_mem::Update()
31. {
32.     QProcess process;
33.     process.start("free -m");           //使用 free 完成获取
34.     process.waitForFinished();
35.     process.readLine();
36.     QString str = process.readLine();
37.     str.replace("\n","");
38.     str.replace(QRegExp("( ){1,}")," ");//将连续空格替换为单个空格 用于
    分割
39.     auto lst = str.split(" ");

```

```

40.     if(lst.size() > 6)
41.     {
42.         Mem_Used = lst[1].toDouble();
43.         ui->Mem_Monitor->setText(QString::number(Mem_Used,'f',0));
44.         Mem_Free = lst[3].toDouble();
45.         ui->Mem_Free->setText(QString::number(Mem_Free,'f',0));
46.         //QDebug("Mem Total:%.01fMB Free:%.01fMB",lst[1].toDouble(),l
           st[6].toDouble());
47.     }
48. }
49. res_mem::~res_mem()
50. {
51.     delete ui;
52. }

```

CPU:

```

1. #include "res_pro.h"
2. #include "sys/statfs.h"
3. #include "ui_res_pro.h"
4. #include <QTimer>
5. #include <unistd.h>
6.
7. res_pro::res_pro(QWidget *parent):
8.     QWidget (parent),
9.     ui(new Ui::res_pro)
10. {
11.     ui->setupUi(this);
12.     this->setFixedSize(this->width(),this->height());
13.     this->move(600,200);
14.
15.     int pid = getpid();
16.     ui->CPU->setText(QString::number(pid,10));
17.
18.     usage = 0;
19.     ui->CPU_Monitor->setText(QString::number(usage,'f',10));
20.
21.     QTimer*timer = new QTimer (this);
22.     connect(timer,SIGNAL(timeout()),this,SLOT(Update()));
23.     timer->start(1000);
24.
25. }
26.
27. void res_pro::Update()
28. {
29.     QProcess process;

```

```

30.     process.start("cat /proc/stat");
31.     process.waitForFinished();
32.     QString str = process.readLine();
33.     str.replace("\n", "");
34.     str.replace(QRegExp("( ){1,}"), " ");
35.     auto lst = str.split(" ");
36.     if(lst.size() > 3)
37.     {
38.         double use = lst[1].toDouble() + lst[2].toDouble() + lst[3].t
oDouble();
39.         double total = 0;
40.         for(int i = 1; i < lst.size(); ++i)
41.             total += lst[i].toDouble();
42.         if(total - m_cpu_total__ > 0)
43.         {
44.             usage = (use - m_cpu_use__) / (total - m_cpu_total__) * 1
00.0;
45.
46.             ui->CPU_Monitor->setText(QString::number(usage, 'f', 10));
47.
48.             //qDebug("cpu usage:%.2lf%%", (use - m_cpu_use__) / (total
- m_cpu_total__) * 100.0);
49.             m_cpu_total__ = total;
50.             m_cpu_use__ = use;
51.         }
52.     }
53.
54. res_pro::~res_pro()
55. {
56.     delete ui;
57. }

```

**Time:**

```

1. #include "mainwindow.h"
2. #include "sys/statfs.h"
3. #include "ui_mainwindow.h"
4. #include <QMessageBox>
5. #include <unistd.h>
6. MainWindow::MainWindow(QWidget *parent) :
7.     QMainWindow(parent),
8.     ui(new Ui::MainWindow)
9. {
10.     ui->setupUi(this);

```



```

11.     this->setFixedSize(this->width(),this->height());
12.     this->move(1200,550);
13.     int pid = getpid();// get current pid
14.     ui->Tnumber->setText(QString::number(pid,10));
15.     QTimer*timer = new QTimer (this);
16.     connect(timer,SIGNAL(timeout()),this,SLOT(timerUpdate()));
17.     timer->start(1000);
18. }
19.
20. MainWindow::~MainWindow()
21. {
22.     delete ui;
23. }
24.
25. void MainWindow::timerUpdate(void)
26. {
27.     QDateTime time = QDateTime::currentDateTime();
28.     QString str = time.toString("yyyy_MM_dd hh:mm:ss dddd");
29.     ui->Time->setText(str);
30. }

```

#### Copy file:

```

1. #include "widget.h"
2. #include "ui_widget.h"
3. #include <QDebug>
4. #include <QApplication>
5. #include <unistd.h>
6. #include <fcntl.h>
7. #include <stdio.h>
8. #include <sys/types.h>
9. #include <sys/stat.h>
10. #include <errno.h>
11. #include <string.h>
12. #include <time.h>
13. #define BUFFER_SIZE 1024
14.
15.
16. Widget::Widget(QWidget *parent) :
17.     QWidget(parent),
18.     ui(new Ui::Widget)
19. {
20.     ui->setupUi(this);
21.     this->move(100,240);
22.     int pid = getpid();

```

```

23.     ui->COPY->setText(QString::number(pid,10));
24.     ui->pushButton_3->setEnabled(false);
25. }
26.
27. Widget::~Widget()
28. {
29.     delete ui;
30. }
31.
32. void Widget::on_pushButton_clicked()
33. {
34.     int from_fd,to_fd;
35.     int bytes_read,bytes_write;
36.     char buffer[BUFFER_SIZE];
37.     ui->pushButton->setEnabled(false);
38.     ui->pushButton_3->setEnabled(true);
39.     ui->pushButton_2->setEnabled(false);
40.
41.     QString from_name = ui->textEdit->toPlainText();
42.     QString to_name = ui->textEdit_2->toPlainText();
43.     QString tip_message = "";
44.     QString huiche = "\n";
45.
46.     ui->textEdit_3->setPlainText(tip_message);
47.     if((from_fd=open(from_name.toLatin1().data(),O_RDONLY))!=-1){
48.         ui->textEdit_3->setPlainText((tip_message+=from_name+=huiche+=
(QString)strerror(errno)+=huiche));//"From File Open Error!"
49.         ui->pushButton->setEnabled(true);
50.         ui->pushButton_2->setEnabled(true);
51.         //return;
52.     }
53.     else ui->textEdit_3->setPlainText((tip_message+="Source File Open
Success!\n"));
54.
55.     if((to_fd=open(to_name.toLatin1().data(),O_WRONLY|O_CREAT,0777))=
=-1){
56.         ui->textEdit_3->setPlainText((tip_message+=huiche+=(QString)s
trerror(errno)+=huiche));//+="To File Open Error!\n"
57.         ::close(from_fd);
58.         ui->pushButton->setEnabled(true);
59.         ui->pushButton_2->setEnabled(true);
60.         // return;
61.     }

```

```

62.     else ui->textEdit_3->setPlainText((tip_message+="Target File Open
        Success!\n"));
63.
64.     int file_size = lseek(from_fd, 0, SEEK_END);
65.     int sumread =0;
66.     cancel_flag = false;
67.     lseek(from_fd, 0, SEEK_SET);
68.     /* 拷贝文件 */
69.     while((bytes_read=read(from_fd,buffer,BUFFER_SIZE))>0)
70.     {
71.         if(cancel_flag) break;
72.         char *ptr=buffer;
73.         sumread += bytes_read;
74.         while((bytes_write=write(to_fd,ptr,bytes_read))>0){//yi zh
            i xie
75.             {
76.                 if(bytes_write==bytes_read) break;//zhe yi lun xie wa
                    n cheng
77.                 else if(bytes_write>0){
78.                     ptr+=bytes_write;
79.                     bytes_read-=bytes_write;
80.                 }
81.             }
82.             if((bytes_write==-1)&&(errno!=EINTR)){// xie shi bai
83.                 ::close(from_fd);
84.                 ::close(to_fd);
85.                 ui->textEdit_3->setPlainText((tip_message+="WRITE REE
                    OR!\n"));//du de guo cheng shi bai
86.                 ui->pushButton->setEnabled(true);
87.                 ui->pushButton_2->setEnabled(true);
88.                 break;
89.             }
90.             ui->progressBar->setValue(sumread/(file_size)*100);
91.         }
92.         if((bytes_read==-1)&&(errno!=EINTR)) {//du shi bai
93.             ::close(from_fd);
94.             ::close(to_fd);
95.             ui->textEdit_3->setPlainText((tip_message+="READ REEOR!\n"));
                //du de guo cheng shi bai
96.             ui->pushButton->setEnabled(true);
97.             ui->pushButton_2->setEnabled(true);
98.         }
99.         ::close(from_fd);
100.        ::close(to_fd);

```

```

101.
102.     if(cancel_flag) ui->textEdit_3->setPlainText((tip_message+="Co
    py Cancel!\n"));
103.     else if(bytes_read==-1||bytes_write==-1)
104.         ui->textEdit_3->setPlainText((tip_message+="Copy Error!\n"
    ));
105.     else ui->textEdit_3->setPlainText((tip_message+="Copy Success!
    \n"));
106.     ui->pushButton_2->setEnabled(true);
107.     ui->pushButton_3->setEnabled(true);
108. }
109. void Widget::on_pushButton_2_clicked()
110. {
111.     exit(0);
112. }
113. void Widget::on_pushButton_3_clicked()
114. {
115.     cancel_flag = true;
116.     ui->pushButton_3->setEnabled(false);
117.     ui->pushButton->setEnabled(true);
118. }

```

## 2. 实验二 系统调用相关知识

### 2.1 实验目的

掌握系统调用的实现过程，通过编译内核的方法，增加一个新的系统调用。另外编写一个应用程序，使用新添加的系统调用。

### 2.2 实验内容

- 1) 向新内核增添一个新的系统调用，将内核编译、生成，并用新内核启动；
- 2) 新增系统调用实现：文件拷贝，并编写测试应用程序，调用新系统功能调用进行测试。

### 2.3 实验设计

#### 2.3.1 开发环境

1. 硬件环境：
  - a) 中央处理器 CPU: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GH
  - b) 物理内存 RAM: 16GB
2. 编译环境：
  - a) 虚拟机软件: VMware Workstation 16 Pro
  - b) 虚拟机系统版本: Ubuntu14.04 操作系统
  - c) 内核: Linux4.4.10
  - d) 虚拟机核数: 8
  - e) 虚拟机内存: 4.00 GB
  - f) 虚拟机磁盘: 60 GB
  - g) 编译器版本: GCC 7.5.0
  - h) 调试器版本: GNU gdb 8.1.0

#### 2.3.2 实验设计

系统调用是用户应用程序和操作系统内核之间的功能接口，通过系统调用进程可由用户模式转入内核模式，在内核模式下完成相应服务后再返回用户模式。由于要求使用编译内核的方式把新系统调用永久性加入内核中，但因为系统镜像里并不包含内核源码，因此需要额外下载新的内核源码进行内核编译。

然后，将编写好的系统功能调用代码加入内核源码中，并将内核编译即可将新系统调用永久地加入新内核中。

最后，编写测试程序对系统功能调用进行测试。本次实验中新增系统调用的主要功能是文件拷贝，因此在代码结构上可以参考实验一的文件拷贝的实现，但是，由于系统功能调用的在内核空间中运行，系统功能调用的实现与普通用户状态下的程序实现存在一些不同。

由于程序运行在内核空间，因此不能使用用户态的 `open`、`close` 等系统功能调用函数进行文件操作，也不能使用 `printf` 的其他用户空间函数。在内核空间中进行文件打开、关闭以及读写等操作需要使用 `sys_open`、`sys_close`、`sys_read` 以及 `sys_write` 代替系统调用函数（有些内核使用 `ksys_` 函数，不同版本的内核使用的函数存在差异），信息打印函数使用 `printk` 进行输出。

由于内核空间对数据的安全性要求较高，因此会对传入的用户系统参数进行检查，为实现程序功能需要使用 `set_fs(KERNEL_DS)` 规避这种检查，在程序返回时使用 `set_fs` 恢复原状态避免内核出现问题。最后，`sys_open` 函数在打开异常时的返回值不总是 -1（可能是其他负数），因此不能使用 -1 作为打开失败的判断标志。

关于 `set_fs(KERNEL_DS)` 规避检查，可以解释如下。系统调用本来是提供给用户空间的程序访问的，所以，对传递给它的参数（比如上面的 `buf`），它默认会认为来自用户空间，在 `->write()` 函数中，为了保护内核空间，一般会用 `get_fs()` 得到的值来和 `USER_DS` 进行比较，从而防止用户空间程序“蓄意”破坏内核空间；

而现在要在内核空间使用系统调用，此时传递给 `->write()` 的参数地址就是内核空间的地址了，在 `USER_DS` 之上 (`USER_DS ~ KERNEL_DS`)，如果不做任何其它处理，在 `write()` 函数中，会认为该地址超过了 `USER_DS` 范围，所以会认为是用户空间的“蓄意破坏”，从而不允许进一步的执行；为了解决这个问题；`set_fs(KERNEL_DS)`；将其能访问的空间限制扩大到 `KERNEL_DS`，这样就可以在内核顺利使用系统调用了！

## 2.4 实验调试

### 2.4.1 实验步骤

**1. 配置所需包：**使用 `apt` 命令安装或更新一些编译内核需要使用的工具，具体需要安装的软件工具包如下：

```
1.sudo apt-get install libncurses5-dev openssl libssl-dev
2.sudo apt-get install build-essential openssl
3.sudo apt-get install pkg-config sudo apt-get install libc6-dev
4.sudo apt-get install bison sudo apt-get install flex
5.sudo apt-get install libelf-dev
6.sudo apt-get install zlibc minizip
7.sudo apt-get install libidn11-dev libidn11
```

**2. 下载内核源码：**从 Linux 内核下载官网 <http://www.kernel.org/> 或者镜像网站

<http://ftp.sjtu.edu.cn/sites/ftp.kernel.org/pub/linux/kernel/> 下载稳定版本的内核进行编译，经过多次编译尝试，最终本次实验选择内核版本为 4.4.10 的内核进行系统功能实验。

### 3. 编辑内核：打开终端，进入源码压缩包所在的目录，在终端输入命令

```
1. tar -xavf linux-4.4.10.tar.xz -C /usr/src//将内核源码压缩包解压，并移动到 /usr/src 目录下准备开始实验。
```

#### (1) 输入命令

```
1. cd /usr/src/linux-4.4.10//进入源码目录
2. sudo gedit arch/x86/entry/syscalls/syscall_64.tbl//查看并修改系统调用表
```

如图 2-1 所示，按照约定的格式将本次新增的系统功能调用 sys\_copyfile 加入到系统调用表中。

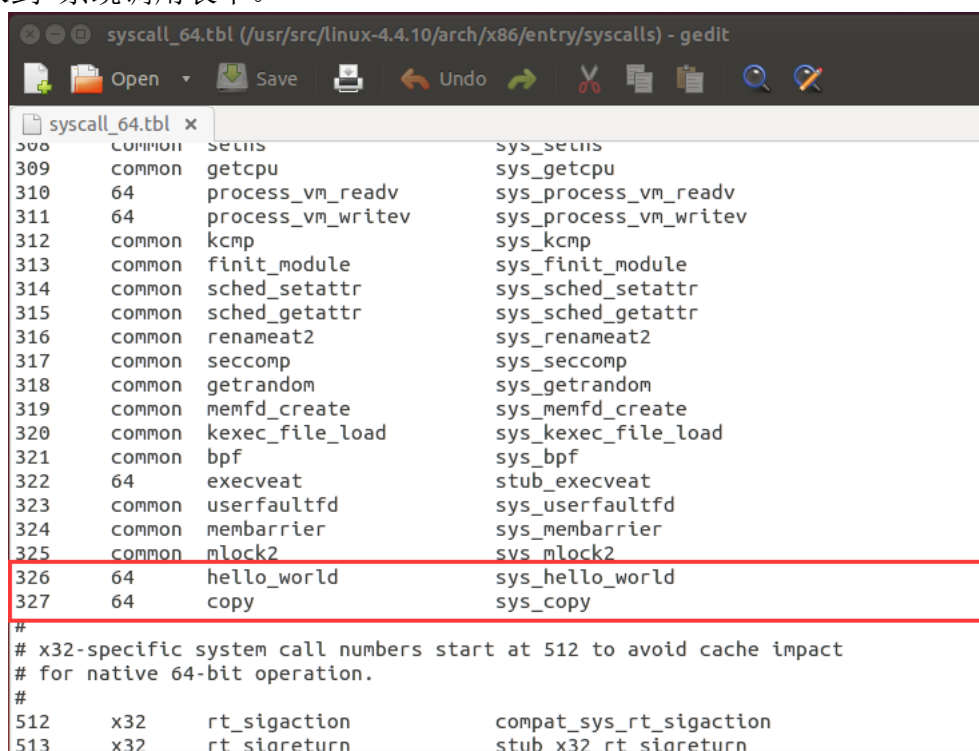


图 2-1 新增系统功能调用号填写图

#### (2) 在终端输入命令

```
1. sudo gedit include/linux/syscalls.h//查看并修改系统功能
```

调用头文件，将新系统功能调用的函数声明加入到文件末尾，保存并关闭文件。

#### (3) 在终端输入命令

```
1. sudo gedit kernel/sys.c//查看并修改系统功能调用
```

将新系统功能调用的函数实现代码加入到文件末尾，保存并关闭文件。

### 4. 编译内核：系统调用代码添加完毕，开始编译内核，分别在终端输入命令

```
1. sudo make mrproper //清除原有编译的痕迹
2. sudo make clean //清除原有编译的痕迹
3. sudo make menuconfig//生成编译配置文件.config。
4. sudo make -j8//开始进行内核的编译，j 之后的数字为内核的数目，本次给虚拟机分配的
   内核数量为 8，为加快编译使用 8 个线程编译。
```

### 5. 安装内核：当内核编译完成后，输入命令

```
1. sudo make modules_install
```

```
2. sudo make install//安装内核模块并将内核加入到系统中，编译完成。
```

## 6. 选择内核：在终端输入命令

```
1. sudo gedit /etc/default/grub//修改 Grub 文件
2. GRUB_TIMEOUT_STYLE=hidde//注释该条语句
3. GRUB_TIMEOUT= -1 //修改为-1 即可在每次启动都进入内核选择界面。
```

## 7. 重启虚拟机：使用新编译完成的内核进入系统，使用编写好的测试程序对程序进行测试

### 2.4.2 实验调试及心得

Make 过程十分的冗长，但是在查阅资料以后，通过修改虚拟器的配置，给虚拟机分配 8 个核，以及在 make 指令当中加入-j8 来实现把八线程的编译，可以使编译速度得到一个可观的提升。

编写新系统功能调用的测试程序对新增的系统功能调用进行测试，使用 syscall 函数调用新系统功能调用，由于新增系统功能调用的系统功能号为 327，且需要两个参数，分别是源文件名以及目标文件名，因此使用

```
1. syscall(327, argv[1], argv[2])
```

调用新系统功能调用，测试效果如 图 2-2 所示：



```
chengleishen@ubuntu: ~/OSdesign/part2
chengleishen@ubuntu:~/OSdesign/part2$ ./test
调用hello_world
系统调用返回值为 1
chengleishen@ubuntu:~/OSdesign/part2$ ./testcopy source.txt target.txt
调用copy
系统调用返回值为 1
chengleishen@ubuntu:~/OSdesign/part2$
```

图 2-2 新增系统功能调用测试结果图

## 附录 实验代码

Test:

```
1. #include <stdio.h>
2. #include <linux/kernel.h>
3. #include <sys/syscall.h>
4. #include <unistd.h>
5.
6. int main(int argc, char const *argv[])
7. {
8.     // 调用 hello_world
9.     printf("调用 hello_world\n");
```



```

10.     long int callreturn = syscall(326);
11.     printf("系统调用返回值为 %ld \n", callreturn);
12.
13.     return 0;
14. }

```

#### Test copyfile:

```

1. #include <stdio.h>
2. #include <linux/kernel.h>
3. #include <sys/syscall.h>
4. #include <unistd.h>
5.
6. int main(int argc, char const *argv[])
7. {
8.     // 调用 copy
9.     printf("调用 copy\n");
10.    long int callreturn = syscall(327,argv[1],argv[2]);
11.    printf("系统调用返回值为 %ld \n", callreturn);
12.    return 0;
13. }

```

#### Copyfile:

```

1. asmlinkage long sys_copyfile(const char __user* source, const char __us
    er* target)
2. {
3.     int from_fd, to_fd;
4.     int bytes_read,bytes_write;
5.     char buffer[1024];
6.     char *ptr;
7.
8.     /* 解除内核对用户地址的访问检查 */
9.     mm_segment_t old_fs = get_fs();
10.    set_fs(KERNEL_DS);
11.
12.    /* 打开源文件与目标文件 */
13.    from_fd = sys_open(source, O_RDONLY, S_IRUSR);
14.    if (from_fd <= 0){
15.        printk("source path error!\n");
16.        set_fs(old_fs);
17.        return -1;
18.    }
19.    to_fd = sys_open(target, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
20.
21.    if (to_fd <= 0){
22.        printk("target path error!\n");
23.        set_fs(old_fs);
24.        return -1;
25.    }
26.
27.    ptr = buffer;
28.    while(1){
29.        bytes_read = read(from_fd, ptr, 1024);
30.        if (bytes_read < 0)
31.            return -1;
32.        bytes_write = write(to_fd, ptr, bytes_read);
33.        if (bytes_write < 0)
34.            return -1;
35.        ptr += bytes_read;
36.    }
37.    return 0;
38. }

```

```

23.         return -1;
24.     }
25.
26.     /* 拷贝文件 */
27.     while((bytes_read=sys_read(from_fd, buffer, 1024)))
28.     {
29.         /* 一个致命的错误发生了 */
30.         if(bytes_read==-1) break;
31.         else if(bytes_read>0)
32.         {
33.             ptr=buffer;
34.             while((bytes_write=sys_write(to_fd, ptr, bytes_read)))
35.             {
36.                 /* 一个致命错误发生了 */
37.                 if(bytes_write==-1)break;
38.                 /* 写完了所有读的字节 */
39.                 else if(bytes_write==bytes_read) break;
40.                 /* 只写了一部分,继续写 */
41.                 else if(bytes_write>0)
42.                 {
43.                     ptr+=bytes_write;
44.                     bytes_read-=bytes_write;
45.                 }
46.             }
47.             /* 写的时候发生的致命错误 */
48.             if(bytes_write==-1)break;
49.         }
50.     }
51.
52.     sys_close(from_fd);
53.     sys_close(to_fd);
54.
55.     /* 读写错误 */
56.     if(bytes_read==-1||bytes_write==-1){
57.         printk("read/write error!\n");
58.         return -1;
59.     }
60.     set_fs(old_fs);
61.     return 1;
62. }

```

## 3 实验三 增加设备驱动程序

### 3.1 实验目的

学习并且掌握向 Linux 系统添加设备驱动程序的方法。

### 3.2 实验内容

通过模块的方法，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：演示字符设备的读与写。

### 3.3 实验设计

#### 3.3.1 开发环境

1. 硬件环境：
  - a) 中央处理器 CPU: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GH
  - b) 物理内存 RAM: 16GB
2. 编译环境：
  - c) 虚拟机软件: VMware Workstation 16 Pro
  - d) 虚拟机系统版本: Ubuntu14.04 操作系统
  - e) 内核: Linux4.4.10
  - f) 虚拟机核数: 8
  - g) 虚拟机内存: 4.00 GB
  - h) 虚拟机磁盘: 60 GB
  - i) 编译器版本: GCC 7.5.0
  - j) 调试器版本: GNU gdb 8.1.0

#### 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间的接口。设备驱动程序为应用程序屏蔽了硬件的细节，在应用程序看来，硬件设备只是一个设备文件，应用程序可以像操作普通文件一样对硬件设备进行操作。设备驱动程序是内核的一部分，它至少需要完成 4 个基本功能，分别是：

- 1、对设备初始化和释放，把数据从内核传送到硬件和从硬件读取数据。
- 2、读取应用程序传送给设备文件数据。
- 3、回送应用程序请求数据。
- 4、检测和处理设备出现的错误。

Linux 支持三中不同类型的设备：字符设备（character devices）、块设备（block devices）和网络设备（network interfaces），本次实验将实现字符设备的驱动，演示内核缓冲区的读写操作。添加新的设备驱动的过程其实是编写函数填充 `file_operations` 的各个域的过程。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。用户进程利用系统调用在对设备文件进行操作时，系统调用通过设备文件的主设备号找到相应的设备驱动程序，然后读取这个数据结构相应的函数指针，接着把控制权交给该函数。

为实现设备驱动的基本功能，至少需要实现 `read`、`write`、`open` 以及 `release` 这四个子函数的编写，其中 `open` 函数使用 `kmalloc` 内核空间分配函数分配缓冲区，`read` 函数负责从缓冲区中读取数据，`write` 函数负责向缓冲区写入数据，`close` 函数则是使用 `kfree` 函数释放内核缓冲区。同时对于可卸载的内核模块，至少还需要模块初始化函数以及模块卸载函数这两个基本的模块用于模块的挂载以及卸载。模块初始化函数需要使用系统功能调用函数 `register_chrdev` 将模块挂载到内核上，模块卸载函数则是使用系统功能调用函数 `unregister_chrdev` 将模块卸载。最后，将完成的源码文件编译，并挂载到内核上即可开始编写测试程序，测试当前的功能

## 3.4 实验调试

### 3.4.1 实验步骤

#### 1. 编写字符驱动程序：

本驱动程序利用内核中的一片缓冲区实现数据的写入与读取。首先，按照要求编写 `read`、`write`、`open` 以及 `release` 这四个子函数，并将之与 `file_operations` 的指定域进行绑定；然后，编写驱动设备的初始化与卸载模块，使用 `module_init` 函数以及 `module_exit` 函数将函数与功能绑定；最后，将编写完成的字符驱动源码复制指定目录下，准备开始编译。字符驱动程序见附录。

#### 2. 模块编译：

源码文件复制到当内核源码目录中的 `drivers/misc` 目录下；然后，进入 `drivers/misc` 目录，修改当前目录下 `Makefile` 文件，添加一行命令“`obj-m += my_drive.o`”；最后，在当前目录下输入命令“`sudo make -C /usr/src/linux SUBDIRS=$PWD modules`”开始编译，如果编译成功将得到 `.ko` 文件。

总结步骤如下：

#### 3. 设备挂载与创建：

在当前目录下输入命令“`sudo insmod ./my_drive.ko`”；然后，输入命令“`cat /proc/devices`”查看系统给此设备分配的主设备号，如图 3-1 所示，当前我的设备

my\_drive 的主设备号为 240; 完成之后, 输入命令“mknod /dev/my\_drive C 240 0”创建虚拟设备, 其中第一项参数为当前创建的虚拟设备的路径以及名称, 第二个参数 C 表示当前创建的设备为字符设备, 第三个参数 240 为设备的主设备号, 第四个参数 0 为设备的从设备号 (可自行分配); 最后, 输入命令 “mknod /dev/my\_drive C 240 0” 创建特殊文件, 输入命令 “chmod 666 /dev/my\_drive” 改变文件权限。

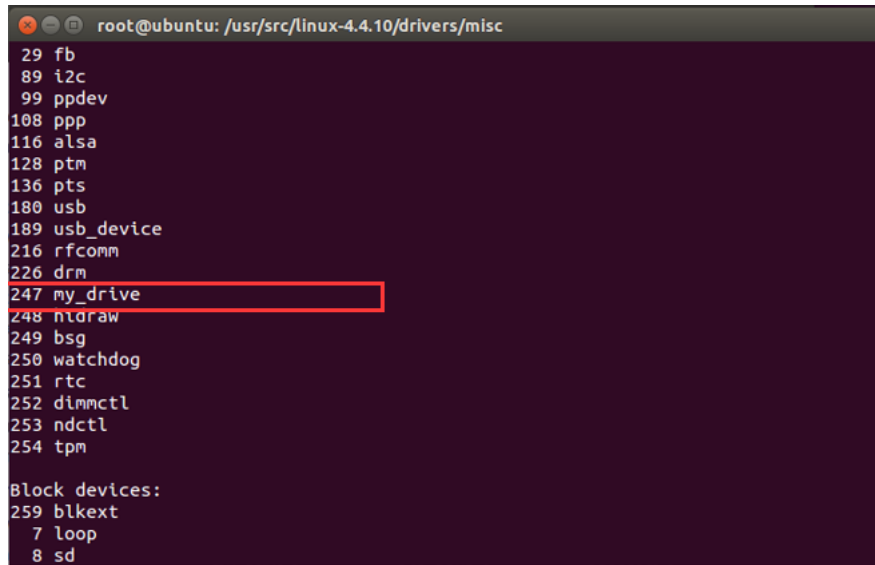


图 3-1 新增设备驱动名称

#### 4. 设备测试:

编写测试程序, 使用 open 系统功能调用打开设备, 系统会自动将控制权交给 file\_operations 的 open 函数指针, 执行之前实现的 open 子函数, 如果当前设备打开失败, 则会返回负值; 然后, 使用 read 函数对设备缓冲区进行读, 最后关闭设备。

测试程序代码如下:

```
1. #include <stdio.h>
2. #include <sys/types.h>
3. #include <sys/stat.h>
4. #include <fcntl.h>
5. #include <stdlib.h>
6. #include <unistd.h>
7. int main()
8. {
9.     int testdev;
10.    int i;
11.    char buf[10];
12.    testdev = open("/dev/my_drive", O_RDWR);
13.    if ( testdev == -1 )
14.    {
15.        printf("Can't open file \n");
16.        exit(0);
17.    }
```

```

18.     read(testdev,buf,10);
19.     for (i = 0; i < 10;i++)
20.         printf("%d\n",buf[i]);
21.     close(testdev);
22. }

```

## 5. 测试成功后卸载的模块

```

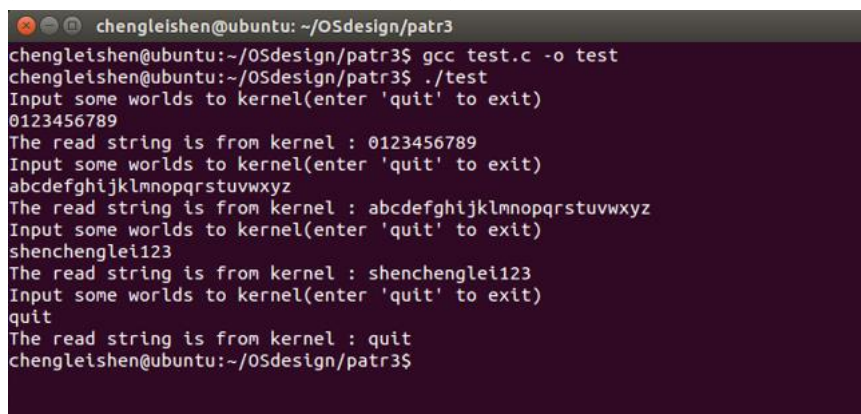
rmmod alex-drive
rm /dev/alex

```

以上步骤中如果权限不够请在命令前加上 sudo.

## 3.4.2实验调试及心得

使用 GCC 命令“gcc -o test test.c”编译事先编写完成的测试程序 test.c 源文件，然后在命令行输入命令“./test”开始进行测试，如图 3-3 所示，测试程序将输入的字符使用 write 函数写入字符设备的缓冲区中，然后使用 read 函数从字符设备的缓冲区中读取字符，最后显示在终端屏幕上，输入 quit 关闭设备。图 3-3 新增设备驱动测试结果图 实验心得：通过实验三的实验过程，我初步掌握了如何实现一个简单的字符设备驱动，了解到了设备驱动的基本运行原理，设备驱动的打开关闭及基本的读写等其他操作的实现方式，同时，掌握了如何将编写完成的驱动设备进行挂载和卸载等。本次实验的主要难点在于掌握设备驱动 file\_operations 结构体在设备驱动编写中的作用以及如何将编写的模块进行编译与挂载等。在本次实验中，我并没有遇到特别严重和困难的问题，主要的问题都是在基本的操作方面，这一个实验的实现相较于实验二来说确实算得上顺利。



```

chengleishen@ubuntu: ~/OSdesign/patr3
chengleishen@ubuntu:~/OSdesign/patr3$ gcc test.c -o test
chengleishen@ubuntu:~/OSdesign/patr3$ ./test
Input some worlds to kernel(enter 'quit' to exit)
0123456789
The read string is from kernel : 0123456789
Input some worlds to kernel(enter 'quit' to exit)
abcdefghijklmnopqrstuvwxyz
The read string is from kernel : abcdefghijklmnopqrstuvwxyz
Input some worlds to kernel(enter 'quit' to exit)
shenchenglei123
The read string is from kernel : shenchenglei123
Input some worlds to kernel(enter 'quit' to exit)
quit
The read string is from kernel : quit
chengleishen@ubuntu:~/OSdesign/patr3$

```

图 3-3 新增设备驱动测试结果

## 附录 实验代码

```

1. #include <linux/kernel.h>
2. #include <linux/module.h>
3. #include <linux/fs.h>
4. #include <linux/init.h>
5. #include <linux/types.h>

```

```

6. #include <linux/errno.h>
7. #include <linux/uaccess.h>
8. #include <linux/kdev_t.h>
9. #include <linux/cdev.h>
10. #include <linux/slab.h>
11.
12. #define MAX_SIZE 1024
13.
14. int my_open(struct inode *inode, struct file *file);
15. int my_release(struct inode *inode, struct file *file);
16. ssize_t my_read(struct file *fileP, char *buf, size_t count, loff_t *ppos);
17. ssize_t my_write(struct file *fileP, const char *buf, size_t count, loff_t *p
    pos);
18.
19. char *data = NULL;
20. int device_num;//设备号
21. char* devName = "my_drive";//设备名
22.
23. struct file_operations pStruct =
24. {
25.     owner:THIS_MODULE,
26.     open:my_open,
27.     release:my_release,
28.     read:my_read,
29.     write:my_write,
30. };
31.
32. /* 注册 */
33. int _my_init_module(void)
34. {
35.     device_num = register_chrdev(0, devName, &pStruct);
36.     if (device_num < 0)
37.     {
38.         printk("failed to register my drive.\n");
39.         return -1;
40.     }
41.     printk("my drive has been registered!\n");
42.     printk("id: %d\n", device_num);
43.     return 0;
44. }
45.
46. /* 注销 */
47. void _my_cleanup_module(void)
48. {

```

```

49.     unregister_chrdev(device_num, devName);
50.     printk("unregister successful.\n");
51. }
52.
53.
54. /* 打开 */
55. int my_open(struct inode *inode, struct file *file)
56. {
57.     try_module_get(THIS_MODULE);
58.     printk("module_refcount(module):%d\n", module_refcount(THIS_MODULE));
59.
60.     data = (char*)kmalloc(sizeof(char) * MAX_SIZE, GFP_KERNEL);
61.     if (!data) return -ENOMEM;
62.     memset(data, 0, MAX_SIZE);
63.     printk("my_drive open successful!\n");
64.     return 0;
65. }
66.
67. /* 关闭 */
68. int my_release(struct inode *inode, struct file *file)
69. {
70.     module_put(THIS_MODULE);
71.     printk("module_refcount(module):%d\n", module_refcount(THIS_MODULE));
72.     printk("Device released!\n");
73.     if (data)
74.     {
75.         kfree(data);
76.         data = NULL;
77.     }
78.     return 0;
79. }
80. /* 读数据 */
81. ssize_t my_read(struct file *fileP, char *buf, size_t count, loff_t *ppos)
82. {
83.     if (!buf) return -EINVAL;
84.     if (count > MAX_SIZE) count = MAX_SIZE;
85.     if (count < 0 ) return -EINVAL;
86.
87.     if (copy_to_user(buf, data, count) == EFAULT)
88.         return -EFAULT;
89.
90.     printk("user read data from device!\n");
91.     return count;
92. }

```



```

93.
94. /* 写数据 */
95. ssize_t my_write(struct file *fileP, const char *buf, size_t count, loff_t *p
    pos)
96. {
97.     if (!buf) return -EINVAL;
98.     if (count > MAX_SIZE) count = MAX_SIZE;
99.     if (count < 0 ) return -EINVAL;
100.
101.     memset(data, 0, MAX_SIZE);
102.     if (copy_from_user(data, buf, count) == EFAULT)
103.         return -EFAULT;
104.
105.     printk("user write data to device\n");
106.     return count;
107. }
108. module_init(_my_init_module);
109. module_exit(_my_cleanup_module);
110. MODULE_AUTHOR("lql");
111. MODULE_LICENSE("GPL");

```

## 4 实验四 使用 QT 实现系统监视器

### 4.1 实验目的

了解 Linux 系统中 /proc 文件的特点和使用方法，掌握使用 /proc 下的文件查询并监控系统中进程运行情况，同时，学会利用 GTK/QT 等 Linux 图形资源库完成对系统资源的图形化显示与控制。

用户以及程序均可以通过 /proc 得到系统的信息，并可以改变内核的某些参数。由于系统的信息均是动态改变的，会经常的进行更新，所以用户或应用程序读取 /proc 获取文件的时候，/proc 文件系统是动态的从系统的内核中获取信息并且提供给用户或应用程序的。

用户或者应用程序若需要获取系统信息的时候，只需要进行相应的文件操作。首先相应的文件 (/proc) 文件打开，然后将所需要的信息写入缓冲区当中，然后将缓冲区的内容加入到 GTK 相应的控件当中，最后将所有的控件进行组合的显示。

### 4.2 实验内容

学习并掌握通过读取 proc 文件系统获取系统各种信息的方法，使用 Linux 下图形库 GTK/QT 进行图形化界面的开发，让系统信息以比较容易理解的方式显示出来，其需要完成的功能具体包括：

- a) 获取并显示主机名
- b) 获取并显示系统启动的时间
- c) 显示系统到目前为止持续运行的时间
- d) 显示系统的版本号
- e) 显示 cpu 的型号和主频大小
- f) 通过 pid 或进程名查询进程，显示该进程详细信息，提供杀掉该进程功能
- g) 显示系统所有进程信息，包括 pid, ppid, 占用内存大小，优先级等
- h) cpu 使用率的图形化显示(2 分钟内的历史纪录曲线)
- i) 内存和交换分区使用率的图形化显示(2 分钟内的历史纪录曲线)
- j) 在状态栏显示当前时间
- k) 在状态栏显示当前 cpu 使用率
- l) 在状态栏显示当前内存使用情况
- m) 用新进程运行一个其他程序

## 4.3 实验设计

### 4.3.1 开发环境

1. 硬件环境:
  - a) 中央处理器 CPU: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
  - b) 物理内存 RAM: 16GB
2. 编译环境:
  - d) 虚拟机软件: VMware Workstation 16 Pro
  - e) 虚拟机系统版本: Ubuntu14.04 操作系统
  - f) 内核: Linux4.4.10
  - g) 虚拟机核数: 8
  - h) 虚拟机内存: 4.00 GB
  - i) 虚拟机磁盘: 60 GB
  - j) 编译器版本: GCC 7.5.0
  - k) 调试器版本: GNU gdb 8.1.0

### 4.3.2 实验设计

本次实验要求使用图形界面实现资源管理器，需要使用到 Linux 的图形库，由于 QT 中的图形界面非常容易上手，且信号和槽的概念十分容易理解，因此本次实验使用 QT 进行资源管理器的编写。

Linux 系统中的系统信息保存在 `/proc` 文件系统中，用户和应用程序可以通过 `/proc` 文件系统就可以得到系统的信息，也可以通过 `/proc` 文件系统改变内核的某些参数。由于系统的信息是动态改变的，所以用户或应用程序读取 `proc` 文件时，`proc` 文件系统是动态从系统内核读出所需信息并提交的。要显示系统信息，只需打开相对应的 `proc` 文件系统中的文件，读取并按照规定要求显示到 QT 的控件上即可实现系统信息的输出，最后将控件进行组合就可以得到最终的控制器。

由于本次实验需要实现的功能较多，为使得布局美观，首先需要对整体的框架以及界面进行设计，实现设计各个控件的位置与相互作用关系，然后分模块对不同的功能进行实现，整体布局如图 4-1 所示：



图 4-1 整体布局

其中具体操作如下：

1、使用 QT 创建基于 Widget 的带 ui 的窗口项目，进入 UI 设计界面进行 UI 设计，将基本的窗口大小调整为 600\*400

2、插入 Widget 部件，将此部件变型为 QTabWidget 部件（此部件可以实现标签页的切换），插入 5 个标签页，分别命名为系统信息、处理器信息、内存信息、进程控制以及说明，基本窗口各个页面功能如下：

- （1）系统信息：显示功能 1-5 的内容，包括主机名、系统启动时间、系统运行时间、版本号、CPU 的型号与主频；
- （2）处理器信息：显示功能 8 的内容，CPU 使用率 2 分钟内纪录曲线；
- （3）内存信息：显示功能 9 的内容，内存使用率 2 分钟曲线；
- （4）磁盘信息：与内存信息在一个界面，显示磁盘使用率 2 分钟曲线；
- （5）进程控制：显示功能 7 的各个进程的信息，并通过排序的方法，实现功能 6 进程的查找与杀死功能；
- （6）状态栏：显示功能 9-11 的内容，分别在窗口的最下端始终显示当前的 时间、CPU 使用率以及内存使用情况
- （7）简易文件：随手写的一个比较有意思小功能，可以创建文件，保存文件，类似一个小记事本。

3. 按照规划将各个控件放置到设计好的位置上，系统信息页面的设计比较简单，需要使用两个 QFrame 部件作为分块显示部件，再向 Frame 上添加 Label 作为显示信息的部件，具体布局如图 4-1 所示；其他几个页面的设计也采用相同的方式。

其中，由于需要在处理器信息以及内存信息页面画曲线，因此需要创建 QVBoxLayot 类的部件作为显示的背板；进程控制页面使用 QTableWidget 部件显示各个进程的信息（没选择 QListWidget 的原因主要是 QListWidget 没有办法按照特定的值将表格排序，因此选择 QTableWidget）。

4. 分模块分功能实现填充各个部件，将系统信息显示到指定部件上，并通过事件以及槽函数实现对进程控制操作及信息刷新，各功能具体设计过程如下：

（1）显示主机名：打开文件/proc/sys/kernel/hostname，该文件的内容即是本系统的主机名，使用 readAll 函数读取主机信息并转化成 QString 后使用

setText 方法将主机名信息显示在 Label 上:

```
1. QFile tempFile;
2.   QByteArray allArray;
3.   QString tempStr;
4.   /* 设置主机名信息 */
5.   tempFile.setFileName("/proc/sys/kernel/hostname");
6.   tempFile.open(QIODevice::ReadOnly);
7.   allArray = tempFile.readAll();
8.   tempStr = QString(allArray);
9.   tempFile.close();
10.  ui->host_msglabel->setText(tempStr.mid(0,tempStr.length()-1));
```

(2) 显示系统启动的时间: 可以读取/proc/uptime 文件获取 uptime, 但是为了简化程序, 使用 Linux 的 sysinfo 函数获取 uptime, 将当前时间减去运行时间 即是系统开始运行的时间。使用 localtime 函数将开始运行的时间从秒转化为可读的时间, 并用 setText 方法将系统启动时间输出, 代码:

```
1.  /* 设置启动时间信息 */
2.  ptm=localtime(&boot_time);
3.  sprintf(time,"%d.%d.%d %02d:%02d:%02d",ptm->tm_year+1900,
4.          ptm->tm_mon+1,ptm->tm_mday,ptm->tm_hour,ptm->tm_min,ptm->tm_
5.          sec);
6.  ui->start_msglabel->setText(QString(time));
```

(3) 显示系统运行时间: 使用 Linux 的 sysinfo 函数获取 uptime, 将运行的秒数转化为“天/时/分/秒”的形式输出到标签上:

```
1. /* 读取时间信息 */
2. struct sysinfo info;
3. time_t cur_time=0;
4. time_t boot_time=0;
5. struct tm *ptm=NULLptr;
6. if(sysinfo(&info)) return;
7. time(&cur_time);
8. boot_time=cur_time-info.uptime;
9. char time[30];
```

(4) 显示系统版本号: 打开文件/proc/sys/kernel/osrelease, 该文件的内容即是本系统版本号, 使用相同的方式将信息显示在 Label:

```
1. /*显示系统版本号信息*/
2. tempFile.setFileName("/proc/sys/kernel/osrelease");
3. tempFile.open(QIODevice::ReadOnly);
4. allArray = tempFile.readAll();
5. tempStr = QString(allArray);
```

```

6.    tempFile.close();
7.    ui->ed_msglabel->setText(tempStr.mid(0,tempStr.length()-1));

```

(5) 显示 cpu 的型号和主频大小：打开文件 `/proc/cpuinfo` 查看文件的组织形式,如图 4-2 所示可以发现 cpu 的型号名称处于 `model name` 以及 `stepping` 之间, 因此, 使用 `indexOf` 函数查找到 `model name` 以及 `stepping` 即可找到 cpu 的型号信息; 同理, 找到 cpu 的主频信息, 使用 `setText` 方法将主机名信息显示在 Label 上即可:

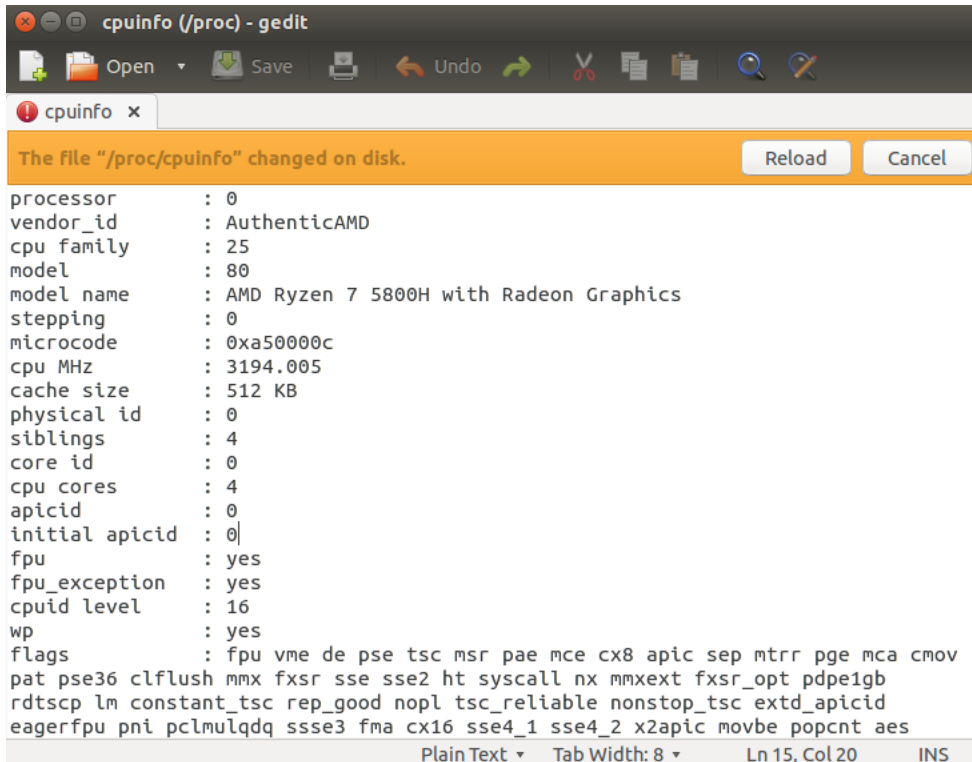


图 4-2 CPU 信息内容显示

```

1. /* 打开 cpuinfo 文件 */
2.    tempFile.setFileName("/proc/cpuinfo");
3.    tempFile.open(QIODevice::ReadOnly);
4.    allArray = tempFile.readAll();
5.    tempStr = QString(allArray);
6.    tempFile.close();
7.
8.    /* 设置 cpu 型号信息 */
9.    int from=tempStr.indexOf("model name");
10.   int to=tempStr.indexOf("stepping");
11.   ui->cpu_msglabel->setText(tempStr.mid(from+13,to-from-14));
12.
13.   /*读取 cpu 主频信息*/
14.   from=tempStr.indexOf("cpu MHz");
15.   to=tempStr.indexOf("cache size");

```

```
16.    ui->cpu_fremsglabel->setText(tempStr.mid(from+11,to-from-12)+" MHz
    ");
```

(6) 显示进程信息：显示进程信息存在两种选择，第一种是使用 ListWidget 进行显示，第二种是使用 TableWidget 进行显示，由于本次实验需要使用各个信息进行排序，如果使用 ListWidget 比较麻烦，因此使用 TableWidget 进行显示。为了显示所有进程的信息，首先需要使用 QDir 读取 /proc 目录下的所有文件夹，使用 toInt 函数将文件夹的名称转换成数字，如果转换失败则表示进程读取 25 完成；如果当前文件夹的名称为纯数字，表示当前文件夹内的文件保存着进程信息，打开当前文件夹下的 stat 文件读取当前进程的信息，通过查阅资料可知，进程名使用小括号包裹，ppid 的值为第 3 个数字，优先级为第 17 个数字，内存大小为第 22 个数字，将信息读取并显示到 TabelWidget 上；使用循环持续读取进程信息显示到表格中即可，代码如下：

```
1 1 (systemd) S 0 1 1 0 -1 4194560 177470 17839783 136 8106 508 1000 36039
16575 20 0 1 0 8 173203456 2558 18446744073709551615 1 1 0 0 0 0 671173123
4096 1260 0 0 0 17 3 0 0 13 0 0 0 0 0 0 0 0 0 0
```

图 4-3 进程信息表

```
1. while(true)
2.     {
3.         /* 获取进程 PID */
4.         pro_id = qsList[find_begin++];
5.         /* 进程文件读取完成 */
6.         int temp_num = pro_id.toInt(NULL, 10);
7.         if(temp_num == 0) break;
8.
9.         /* 打开进程状态文件 */
10.        QFile tempFile("/proc/" + pro_id + "/stat");
11.        tempFile.open(QIODevice::ReadOnly);
12.        tempStr = tempFile.readAll();
13.
14.        int begin_index = tempStr.indexOf("(");
15.        int end_index = tempStr.indexOf(")");
16.        pro_Name = tempStr.mid(begin_index+1, end_index-begin_index-1)
17.        ;
18.        parent_pro_id = tempStr.section(" ", 3, 3);
19.        pro_Priority = tempStr.section(" ", 17, 17);
20.        pro_Mem = tempStr.section(" ", 22, 22);
21.        number_of_pro++;
22.        pro_status=tempStr.section(' ',2,2);
23.        switch(pro_status.at(0).toLatin1()){
24.            case 'S':number_of_sleep++;break;
25.            case 'Z':break;
26.            case 'R':number_of_run++;break;
```

```

26.     }
27.
28.     /* 增加表项 */
29.     ui->tableWidget->insertRow(0);
30.     QTableWidgetItem* pItem = new QTableWidgetItem();
31.     pItem->setData(Qt::EditRole, temp_num);
32.     ui->tableWidget->setItem(0,0,pItem);
33.     ui->tableWidget->setItem(0,1,new QTableWidgetItem(pro_Name));
34.     ui->tableWidget->setItem(0,2,new QTableWidgetItem(parent_pro_i
        d));
35.     ui->tableWidget->setItem(0,3,new QTableWidgetItem(pro_status))
        ;
36.     ui->tableWidget->setItem(0,4,new QTableWidgetItem(pro_Priority
        ));
37.     ui->tableWidget->setItem(0,5,new QTableWidgetItem(pro_Mem));
38.     tempFile.close();
39.     }

```

(7) cpu 使用率的图形化显示：本次实验使用 QChart 控件显示 cpu 使用率历史曲线，首先，初始化曲线画板，然后，每秒刷新曲线即可实现 cpu 使用率历史曲线的图形化显示，具体过程如下。

首先，对画板进行初始化，创建画板并创建一条 120 个点的 曲线（每秒刷新一下，两分钟曲线需要 120 个点），将曲线的每个点的利用率初始化为 0；然后，创建坐标轴并将坐标轴添加到画板；最后，将画板添加到 Layout 布局中即可。

### CPU 使用曲线绘制：

```

1. /*创建画板*/
2.     cpu_chart = new QChart;
3.     mem_chart = new QChart;
4.     swap_chart = new QChart;
5.
6.     /*创建曲线*/
7.     cpu_series = new QLineSeries;
8.     mem_series = new QLineSeries;
9.     swap_series = new QLineSeries;
10.
11.     cpu_chart->addSeries(cpu_series);
12.     mem_chart->addSeries(mem_series);
13.     swap_chart->addSeries(swap_series);
14.

```



```

15.     for(int i=0;i<cpu_series_Size;++i)
16.     {
17.         cpu_series->append(i,0);
18.         mem_series->append(i,0);
19.         swap_series->append(i,0);
20.     }
21.
22.     /*创建 cpu 坐标轴*/
23.     QValueAxis *axisX = new QValueAxis;
24.     axisX->setRange(0,cpu_series_Size);
25.     axisX->setTickCount(cpu_series_Size/8);
26.     axisX->setLabelsVisible(false);
27.
28.     QValueAxis *axisY = new QValueAxis;
29.     axisY->setRange(0, 1);
30.     axisY->setTickCount(5);
31.     axisY->setLabelFormat("%.2f");
32.
33.     cpu_chart->setAxisX(axisX,cpu_series);
34.     cpu_chart->setAxisY(axisY,cpu_series);
35.     cpu_chart->legend()->hide();

```

然后，需要每秒对曲线进行刷新，首先，使用 `QVector` 将曲线的点 转化成由点组成的向量，删除下标为 0 的点；然后，使用 `QFile` 打开 `/proc/stat` 文件，读取并计算 `cpu` 的运行信息。

经过查阅资料可知，`stat` 文件中除了第 5 项为 `cpu` 运行信息，其他各项都是 `cpu` 空闲信息，将当前的运行时间减去上一秒的运行时间，然后除以总运行时间即可得出当前阶段的 `cpu` 使用率；将得出的数值添加到曲线的末端，并将曲线重新绘制到画板上即可实现曲线的更新与移动。

### CPU 使用率刷新：

```

1. void Widget::updateCpuUsg()
2. {
3.     /* 将曲线向前移动一个单位 */
4.     QVector<QPointF> oldPoints = cpu_series->pointsVector();
5.     QVector<QPointF> points;
6.     for(int i=1;i<oldPoints.count();++i)
7.         points.append(QPointF(i-1 ,oldPoints.at(i).y()));
8.
9.     /*打开/proc/stat 文件*/
10.    QFile tempFile("/proc/stat");
11.    tempFile.open(QIODevice::ReadOnly);
12.    QByteArray allArray = tempFile.readLine();
13.    QString tempStr = QString(allArray);

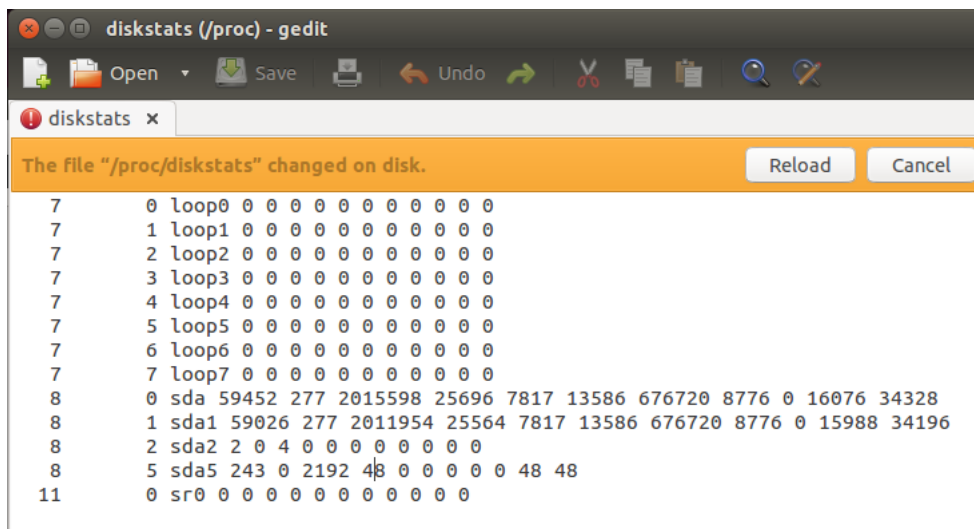
```

```

14.     tempFile.close();
15.     QStringList strList = tempStr.split(" ");
16.     /*读取 cpu 运行信息*/
17.     double total_time = 0;
18.     double use_time = 0;
19.     for(int i=strList.length()-1;i>1;i--)
20.         total_time += strList[i].toInt();
21.     use_time = total_time - strList[5].toDouble();
22.     double usage = (use_time - old_use_time)/(total_time - old_total_
        time);
23.     old_total_time = total_time;
24.     old_use_time = use_time;
25.
26.     points.append(QPointF(oldPoints.count()-1 ,usage));
27.     cpu_series->replace(points);
28.     ui->label->setText("CPU 使用
        率: " + (new QString("%1"))->arg(100*usage).mid(0,4) + "%");
29. }

```

(8) 内存和磁盘使用率的图形化显示：与 CPU 曲线绘制方式相同，唯一不同的是需要读取的文件为/proc/meminfo 和/proc/diskstats，数据的处理方式也有所不同，打开文件查看文件的组织形式，内存使用情况为前两行数据，按照之前的方式读取并显示即可。



7	0	loop0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	loop1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	2	loop2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	3	loop3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	4	loop4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	5	loop5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	6	loop6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	loop7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	sda	59452	277	2015598	25696	7817	13586	676720	8776	0	16076	34328						
8	1	sda1	59026	277	2011954	25564	7817	13586	676720	8776	0	15988	34196						
8	2	sda2	2	0	4	0	0	0	0	0	0	0	0	0					
8	5	sda5	243	0	2192	48	0	0	0	0	0	0	48	48					
11	0	sr0	0	0	0	0	0	0	0	0	0	0	0	0					

图 4-4 磁盘信息表

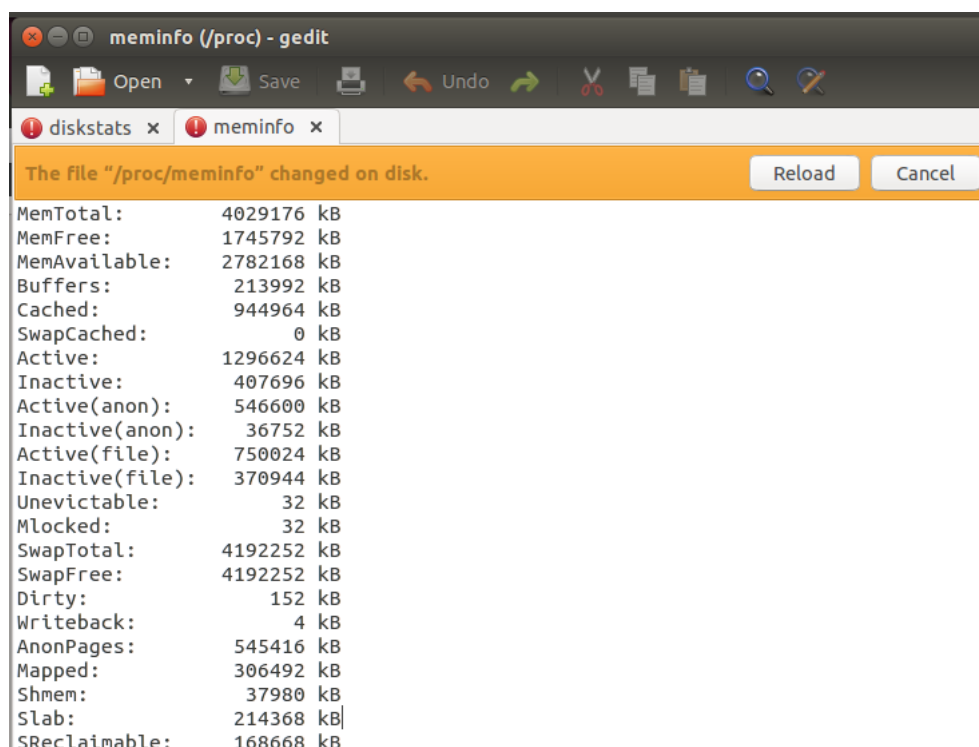


图 4-5 内存信息表

(9) 状态栏显示当前时间：在计算运行时间时，同时计算当前时间并显示。

(10) 状态栏显示当前 cpu 使用率：在绘制 cpu 使用率曲线时，同时将计算出的 cpu 使用率显示到对应的标签中即可。

(11) 状态栏显示当前内存使用情况：绘制内存使用情况曲线时，将使用的内存除以总内存即可得到当前的内存使用率。

## 4.4 实验调试

### 4.4.1 实验步骤

1. 按照实验设计的流程设计并实现资源管理器。
2. 调试程序并修改问题

### 4.4.2 实验调试及心得

实验调试：

1. 测试结果如图 4-6 所示，正确显示基本系统信息，包括主机名 ubuntu、系统的启动时间、运行时间、CPU 的型号主频以及系统内核的版本号；图 4-6 资源管理器系统信息页面效果图。



图 4-6 系统信息效果图

2. cpu 历史使用率曲线的测试结果如图 4-7 所示，由于 Linux 系统的特点，cpu 的使用率并不高，但是可以看到 cpu 的使用率随着时间发生变化，实现了 cpu 使用率的图形化显示功能。同理，内存使用率以及交换区使用率曲线如图 4-7 所示，可以发现成功实现了功能。



图 4-7 资源管理器处理器信息页面效果图

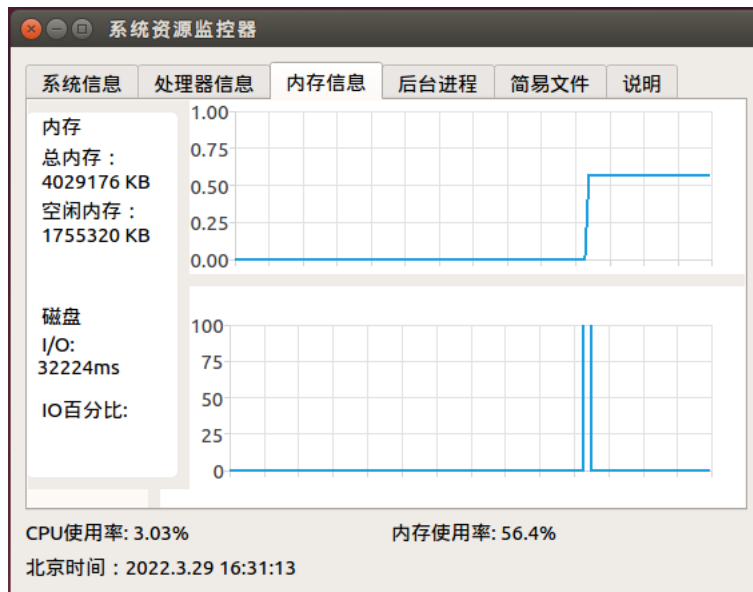


图 4-8 资源管理器内存信息页面效果图

3. 进行进程控制页面的测试，首先打开进程控制界面，如图 4-9 所示，可以看到当前存在 277 个进程，运行中的进程数为 1，睡眠中的进程数为 276；点击 PID 或者进程名表头，可以发现进程按照该值进行排序，程序功能正常。



图 4-9 资源管理器进程控制页面效果图

4. 简易文件测试，如图 4-10 所示，功能和记事本类似，是一个小玩具，不再赘述。



图 4-10 简易文件效果图

### 实验心得:

实验四需要实现一个功能复杂的系统资源管理器，与之前的项目不同，这个项目需要完成一个具有一定功能的较为复杂的实际项目，是一个比较接近于现实项目设计与实现的场景。本次实验的主要目的就是掌握 `/proc` 文件的特点和使用方法，以及熟悉图形化编程的流程。通过本次实验，我充分了解了 `proc` 文件系统中各个文件的作用以及基本的组织形式，初步掌握了 `QT` 的图形化编程的流程。本次实验中我也遇到了很多的问题，由于本次实验涉及到图形化编程，对于图形化编程的不熟悉在编写初期给我制造了相当多障碍，通过查阅各种资料，我很快掌握了基本的图形化编程方法。本次实验我的收获也很多，不仅了解了如何读取系统信息，更掌握了 `QT` 的使用方法，我觉得这会对未来有很大帮助！

## 附录 实验代码

补上文未展示代码:

简易文件:

```
1. void Widget::on_createfile_clicked()
2. {
3.     ui->editfiletext->clear();
4. }
5.
6. void Widget::on_openfile_clicked()
7. {
8.     QString fileName=QFileDialog::getOpenFileName(this, tr("打开一个文件"),
9.                                                     "/home/chengleishen/OSdesign",
10.                                                     "*.txt");
```

```

11.     if(fileName.isEmpty())
12.     {
13.         QMessageBox::warning(this, "warning", "请选择一个文件");
14.     }
15.     else
16.     {
17.         QFile file(fileName); //创建文件对象
18.         file.open(QIODevice::ReadOnly);
19.         QByteArray ba = file.readAll();
20.         ui->editfiletext->setText(QString(ba));
21.     }
22.
23. }
24.
25. void Widget::on_savefile_clicked()
26. {
27.     QString fileName=QFileDialog::getSaveFileName(this, tr("选择一个文件"),
28.                                                     "/home/chengleishen/OSdesign",
29.                                                     "*.txt");
30.     if(fileName.isEmpty())
31.     {
32.         QMessageBox::warning(this, "warning", "请选择一个文件");
33.     }
34.     else
35.     {
36.         QFile file(fileName);
37.         file.open(QIODevice::WriteOnly);
38.         QByteArray ba;
39.         ba.append(ui->editfiletext->toPlainText());
40.         file.write(ba);
41.         file.close();
42.     }
43. }

```

# 5 实验五 小型文件系统

## 5.1 实验目的

使用文件系统的知识，设计并实现一个模拟的文件系统，掌握文件系统的基本结构的实现以及加深对文件系统的了解和掌握。

## 5.2 实验内容

### linux 文件系统概述：

文件系统指文件存在的物理空间，linux 系统中每个分区都是一个文件系统，都有自己的目录层次结构。linux 会将这些分属不同分区的、单独的文件系统按一定的方式形成一个系统的总的目录层次结构。一个操作系统的运行离不开对文件的操作，因此必然要拥有并维护自己的文件系统。

linux 文件系统使用索引节点来记录文件信息，作用像 windows 的文件分配表。索引节点是一个结构，它包含了一个文件的长度、创建及修改时间、权限、所属关系、磁盘中的位置等信息。一个文件系统维护了一个索引节点的数组，每个文件或目录都与索引节点数组中的唯一一个元素对应。系统给每个索引节点分配了一个号码，也就是该节点在数组中的索引号，称为索引节点号。

linux 文件系统将文件索引节点号和文件名同时保存在目录中。所以，目录只是将文件的名称和它的索引节点号结合在一起的一张表，目录中每一对文件名称和索引节点号称为一个连接。

对于一个文件来说有唯一的索引节点号与之对应，对于一个索引节点号，却可以有多个文件名与之对应。因此，在磁盘上的同一个文件可以通过不同的路径去访问它。可以用 ln 命令对一个已经存在的文件再建立一个新的连接，而不复制文件的内容。

连接有软连接和硬连接之分，软连接又叫符号连接。它们各自的特点是：  
硬连接：原文件名和连接文件名都指向相同的物理地址。目录不能有硬连接；硬连接不能跨越文件系统（不能跨越不同的分区）文件在磁盘中只有一个拷贝，节省硬盘空间。

由于删除文件要在同一个索引节点属于唯一的连接时才能成功，因此可以防止不必要的误删除。

符号连接：用 ln -s 命令建立文件的符号连接符号连接是 linux 特殊文件的一种，作为一个文件，它的数据是它所连接的文件的路径名。类似 windows 下的快捷方式。可以删除原有的文件而保存连接文件，没有防止误删除功能。

### 本次实验要求：

使用磁盘中的一个文件（大小事先指定）来模拟磁盘，设计并实现一个模拟的文件系统，需要完成如下功能与结构：

1. 设计并实现完善的文件目录项的结构；
2. 实现空白块的管理；
3. 实现文件系统的基本操作，如文件打开删除、目录的创建删除等；



4. 选择支持多用户与树形目录的设计与实现。

## 5.3 实验设计

### 5.3.1 开发环境

1. 硬件环境:
  - a) 中央处理器 CPU: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
  - b) 物理内存 RAM: 16GB
2. 编译环境:
  - c) 虚拟机软件: VMware Workstation 16 Pro
  - d) 虚拟机系统版本: Ubuntu14.04 操作系统
  - e) 内核: Linux4.4.10
  - f) 虚拟机核数: 8
  - g) 虚拟机内存: 4.00 GB
  - h) 虚拟机磁盘: 60 GB
  - i) 编译器版本: GCC 7.5.0
  - j) 调试器版本: GNU gdb 8.1.0

### 5.3.2 实验设计

使用 QT 控制台项目开发文件系统,本次实验需要使用一个大文件模拟磁盘进行文件操作,通过规划设计,计划本次将使用大小为 10MB 的文件模拟磁盘,系统磁盘块的大小为 1024B,总磁盘块数为 10240 块,总大小为 10M,可以使用宏定义的方式进行调整,具体文件系统设计如下:

1. 文件系统总体结构:本文件系统支持多用户与树形目录,基本磁盘块大小为 1024B,磁盘块数为 10240 个(可以通过宏定义调整),磁盘总大小为 10M,文件组织形式采用串联文件,空白块管理使用空白块链。

使用 0 号块为特殊引导块存放空白块的管理信息以及用户控制信息(多用户),每个用户拥有各自独立的根目录,不可以相互访问,用户之间完全隔离,具体结构如图 5-1 所示。

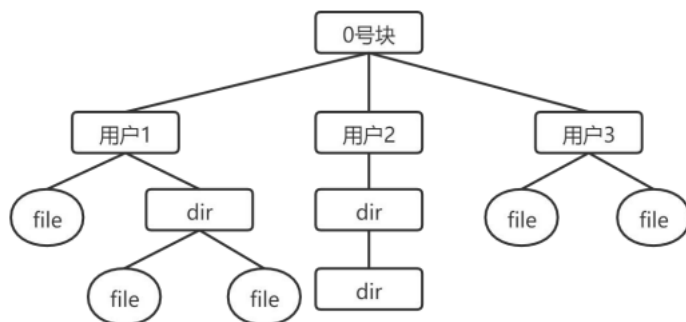


图 5-1 文件系统总体结构示意图

此外，本文件系统支持多项用户、文件以及目录操作，用户操作包括注册、移除、登录以及显示用户信息操作，文件操作包括创建、删除、查看内容以及编辑内容，目录操作包括创建、删除、显示目录下文件以及切换当前目录操作。本系统的所有路径相关操作均支持绝对路径和相对路径操作，且可以解析复杂的文件路径名，操作体验接近于真实的系统。

2. 具体文件组织形式：普通文件的组织形式如图 5-2 所示，采用串联文件形式进行组织，每个文件块的头 4 个字节为串联指针，指向下一个文件块，最后一个文件块的串联指针指向 0 号块（NULL）；为简化系统设计与实现，本文件系统规定，每个目录文件最多使用 1 个磁盘块，而普通文件使用的磁盘块个数没有限制，文件可以为任意大小。

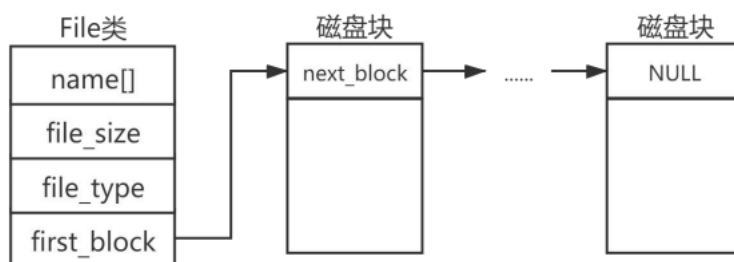


图 5-2 文件组织形式示意图

3. 文件目录项结构：如图 5-2 所示，使用 File 类对文件目录项进行封装，每个文件目录项的大小为 32B，文件名占 20 字节，然后文件大小 file\_size、文件类型 file\_type（e file\_type 的数值为 DIR\_TYPE 宏定义表示目录文件，为 FILE\_TYPE 宏定义表示普通文件）以及文件块指针 first\_block 各 4 个字节。文件目录的基本结构如图 5-3 所示，由于目录最多使用一个磁盘块，因此每一级目录中最多可以容纳 32 个普通文件或目录文件。

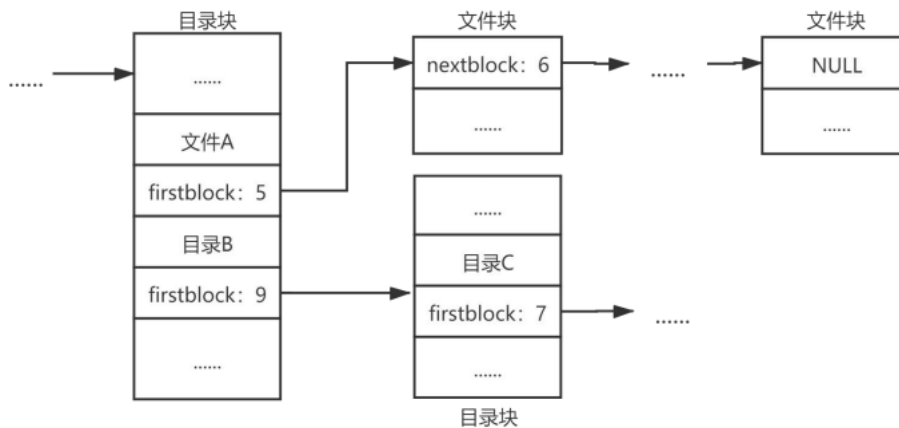


图 5-3 文件目录结构示意图

4. 空白块管理：使用空白块链的方式对空白块进行管理，将所有空白块以链表的形式进行串联，每个空白块的头 4 个字节作为串联指针，指向下一个空白块，最后一个空白块指向 NULL（0 号块）。

使用 0 号块作为特殊引导块管理空白块信息，0 号块的前 4 个字节为空白块链头指针，指向第一个空白块；0 号块的次 4 个字节为空白块的个数，用于管理当前空白块；之后存放用户控制块 Usr 的信息，用户控制块定义详见其他数据结构中（2）部分定义。

5. 文件系统的初始化与磁盘创建：每次进行文件系统初始化时，首先使用 r+的模式使用 fopen 函数打开磁盘模拟文件；如果文件不存在，则需要使用 a+的方式创建磁盘文件，并将磁盘文件全部初始化为 0；如果文件存在，则直接使用 r+的模式使用 fopen 函数打开磁盘模拟文件即可进行操作。

6. 文件系统基本操作：文件系统需要支持一些基本操作，如将指定块从磁盘读到内存中，将内存中指定块的内容重新写入磁盘块中等操作，本文件系统定义了 7 种基本操作，具体定义如图 5-4 所示，包括磁盘指定块读写，空白块的获取与释放，文件目录项指针设置以及 0 号块的更新功能。

```

92  /* 文件系统基本操作 */
93  void read_from_Disc(int block, char* buf); /* 从磁盘读取数据 */
94  void write_to_Disc(int block, char* buf); /* 向磁盘写入数据 */
95  int get_block(); /* 获取空白块 */
96  int release_block(int block_num, int File_type); /* 释放数据块 */
97  int set_File_Pointer(File**file, int block_num, int index, Block &block); /* 设置文件指针 */
98  int get_zero_block();
99  int update_zero_block();
100
101  /* 文件操作 */
102  File_control_block* open_file(string &filepath, int mode);
103  int read_from_file(File_control_block* file, int size, char *buf, int pos);
104  int write_to_file(File_control_block* file, int size, const char* buf, int pos);
105  int clear_file(File_control_block* file, int mode);
106  File_control_block* close_file(File_control_block* file);
107
108  /* 目录操作 */
109  int create_dir(string &filename, int mode);
110  int remove_dir(string &filename);
111  int delete_dir_dfs(int block, int file_size);
112  int list_files();
113  int cd_dir(string &filepath);
114
115  /* 用户操作 */
116  int log_in(string &user, string &pass);
117  int log_out();
118  int Register(string &user, string &pass);
119  int Remove(string &user, string &pass);
120  int show_Usr();
121

```

图 5-4 文件系统操作定义代码图

7. 文件系统拓展操作：文件系统的拓展操作包括文件操作、目录操作以及用户操作这三类操作，由于本次实现的文件系统拓展操作数量较多，因此无法完全在此进行详细描述，本次报告仅做简述，详细参见代码。

(1) 用户操作：用户操作包括注册、移除、登录以及显示用户信息，以登录操作为例，当系统检测到用户输入“login”命令时，会调用 log\_in 函数进行登录，使用循环的方式查询用户信息，然后如图 5-5 所示，对用户名以及密码进行比对，当两者都相同时，表示登录成功需要将 0 号块的引导信息加载到内存中，需要设置包括根目录、当前目录以及空白块指针等数据。

```

978  /* 用户登录函数 */
979  int File_system::log_in(string &user, string &pass)
980  {
981      int usr_num = 0;
982      char buf[BLOCK_SIZE];
983      /* 读取引导块信息 */
984      read_from_Disc(0, buf);
985      Ustr* currUstr = (Ustr*)&buf[8];
986      /* 查找当前是否存在该用户 */
987      for(;usr_num<MAX_USR_NUM;usr_num++)
988      {
989          if(!currUstr[usr_num].vaild) return USR_NOT_EXIST;
990          if(!strcmp(currUstr[usr_num].username, user.c_str()))
991          {
992              cout << "用户名正确! " << endl;
993              if(!strcmp(currUstr[usr_num].passname, pass.c_str()))
994              {
995                  cout << "密码正确! " << endl;
996                  use_num = usr_num;
997                  /* 设置当前的根目录 读取引导块 */
998                  read_from_Disc(0, zero_block.buffer);
999                  currUstr = (Ustr*)&zero_block.buffer[8];
1000                  rootDir = (File*)&(currUstr[usr_num]);
1001
1002                  /* 设置当前目录以及临时目录的目录块 */
1003                  currentDir=rootDir;
1004                  currentDir_num=0;
1005                  currentDir_index=usr_num;
1006                  tempDir=rootDir;
1007                  tempDir_num=0;
1008                  tempDir_index=usr_num;
1009
1010                  /* 加载空白块处理逻辑 */
1011                  read_from_Disc(0, buf);
1012                  bk_fhead = *((int*)buf); /* 空白块头指针 */
1013                  bk_nfree = ((int*)buf)[1]; /* 空白块数 */
1014                  usr_name = currUstr[usr_num].username;
1015                  return USR_EXIST;
1016              }
1017          }
1018      }
1019      return USR_NOT_EXIST;
1020  }

```

图 5-5 用户登录函数部分代码图 37

(2) 文件操作：文件操作包括创建（touch）、删除（rm）、查看内容（cat）以及编辑内容（gedit），文件操作支持复杂路径的解析，由于文件操作的实现比较复杂，为提高系统的拓展性，本文件系统将常见的文件操作封装成了 4 个基本函数 open\_file、close\_file、read\_from\_file 以及 write\_to\_file。以 open\_file 文件打开函数为例，open\_file 函数使用使用文件路径进行文件打开，使用 mode 参数区分创建模式。当 mode 为 READ\_WRITE 时，表示仅查找当前文件，不存在时返回异常；当 mode 为 CREATE 时，表示文件不存在时需要创建文件与文件路径上的目录文件，创建异常分为 3 种，分别是文件名冲突、空白块耗尽以及目录已满。

其他函数如 close\_file、read\_from\_file 等的实现详见代码注释，函数实现在代码中有详细的注释，在此不再赘述。

(3) 目录操作：目录操作包括创建（mkdir）、删除（rmdir）、显示目录下文件（ls）以及切换当前目录操作（cd），目录操作的函数实现在代码中有详细

的注释，参见工程文件。

## 5.4 实验调试

### 5.4.1 实验步骤

终端，使用命令“g++ -o disc disc.cpp”将文件系统源文件 disc.cpp 编译 为可执行文件，然后输入命令“./disc”打开可执行文件，如图 5-6 所示，进入文 件系统用户操作界面，在此 界面可进行登录、注册等操作，具体功能测试如下：

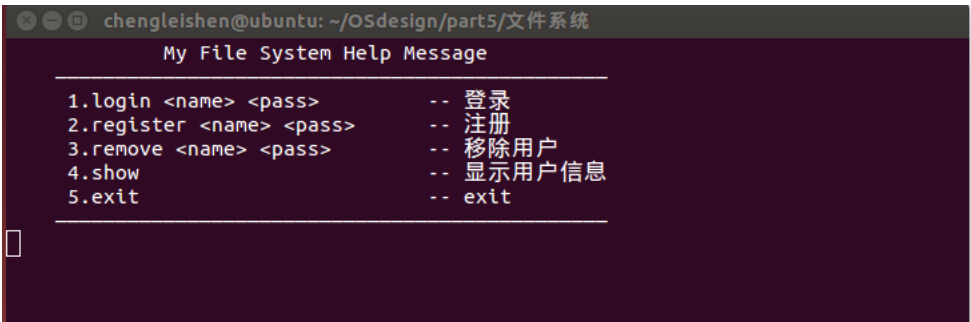


图 5-6 文件系统初始用户操作界面图

**1. 测试用户注册功能：**如图 5-7 所示，输入命令“register chenglei 123456”进行用户注册，系统提示“用户创建成功！”表示当前用户创建成功；这时如果进行重复注册，系统会提示“用户已存在！”。

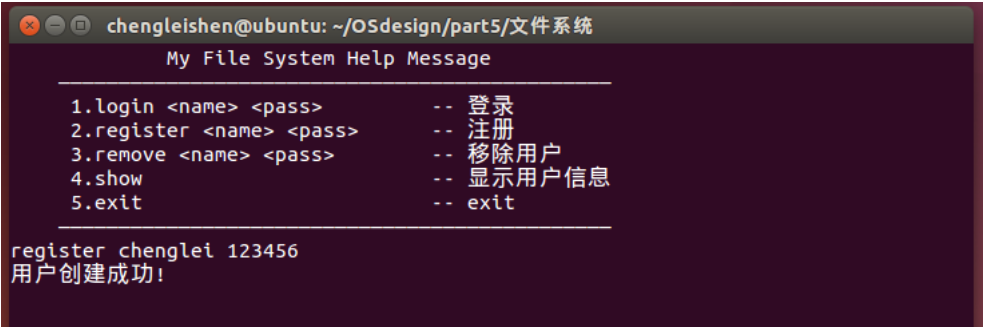


图 5-7 文件系统用户注册功能测试界面图

**2. 测试用户登录功能：**如图 5-8 所示，如果输入的用户名不存在，系统提 示“登录失败！”；如果当前用户存在，但是密码不正确，系统提示“用户名正确！登录失败！”；如果 输入的用户名和密码都正确，系统提示“用户名正确！ 密码正确！”然后进入文件系统的文 件及目录操作界面。当正常进入操作界面后，屏幕会显示文件系统信息提示界面，并显示 “chenglei@ubuntu:\$”表示当前用户为 chenglei， 目录为当前用户的根目录。

```
chengleishen@ubuntu: ~/OSdesign/part5/文件系统
1.login <name> <pass>      -- 登录
2.register <name> <pass>   -- 注册
3.remove <name> <pass>     -- 移除用户
4.show                     -- 显示用户信息
5.exit                     -- exit

login chenglei 123456
用户名正确！
密码正确！

My File System Help Message

1.touch <filename>        -- 新建文件
2.rm <filename>            -- 删除文件
3.cat <filename>           -- 查看文件
4.gedit <filename>        -- 编辑文件
5.mkdir <dirname>          -- 新建目录
6.rmdir <filename>         -- 移除目录
7.ls                       -- 显示目录文件
8.cd <dirname>             -- 进入目录
9.clear                   -- 清除屏幕内容
10.help                   -- 显示帮助
11.exit                   -- 注销用户

chenglei@ubuntu:$
```

图 5-8 文件系统用户登录功能测试界面图

3. 文件创建功能测试：如图 5-11 所示。

```
chengleishen@ubuntu: ~/OSdesign/part5/文件系统
My File System Help Message

1.touch <filename>        -- 新建文件
2.rm <filename>            -- 删除文件
3.cat <filename>           -- 查看文件
4.gedit <filename>        -- 编辑文件
5.mkdir <dirname>          -- 新建目录
6.rmdir <filename>         -- 移除目录
7.ls                       -- 显示目录文件
8.cd <dirname>             -- 进入目录
9.clear                   -- 清除屏幕内容
10.help                   -- 显示帮助
11.exit                   -- 注销用户

chenglei@ubuntu:$mkdir test2.0
chenglei@ubuntu:$cd test2.0
chenglei@ubuntu:test2.0/$touch hhh.txt
chenglei@ubuntu:test2.0/$ls
chenglei/test2.0/:
FILE hhh.txt    0 B
chenglei@ubuntu:test2.0/$
```

图 5-9 文件系统文件创建功能测试界面图

输入命令“mkdir test2.0”创建文件夹，输入命令“cd test2.0”进入文件夹，输入命令“touch hhh.txt”使用 touch 命令创建文件 hhh.txt，使用命令“ls”查看当前目录下的文件，新文件 hhh.txt 创建成功，文件类型为 FILE，文件大小为 0；

4. 文件编辑与查看功能测试：如图 5-12 所示，使用命令“gedit hhh.txt”编辑文件，输入以回车结尾的字符串，然后使用命令“cat hhh.txt”查看文件 hhh.txt 的内容，可以发现文本内容正确；重新使用 gedit 命令编辑文件，可以发现文件内容被覆盖，文件编辑与查看功能实现成功。

```
chengleishen@ubuntu: ~/OSdesign/part5/文件系统
5.mkdir <dirname>          -- 新建目录
6.rmdir <filename>         -- 移除目录
7.ls                       -- 显示目录文件
8.cd <dirname>             -- 进入目录
9.clear                   -- 清除屏幕内容
10.help                   -- 显示帮助
11.exit                   -- 注销用户

chenglei@ubuntu:$mkdir test2.0
chenglei@ubuntu:$cd test2.0
chenglei@ubuntu:test2.0/$touch hhh.txt
chenglei@ubuntu:test2.0/$ls
chenglei/test2.0/:
FILE hhh.txt  0 B
chenglei@ubuntu:test2.0/$gedit hhh.txt

请输入需要输入的文本(\n结尾):
hello, my name is shenglei! today is 3/29
chenglei@ubuntu:test2.0/$cat hhh.txt
hello, my name is shenglei! today is 3/29
chenglei@ubuntu:test2.0/$ls
chenglei/test2.0/:
FILE hhh.txt  41 B
chenglei@ubuntu:test2.0/$
```

图 5-10 文件系统文件查看与编辑功能测试界面图

**5. 文件删除功能测试:** 如图 5-11 所示。

输入命令“rm hhh.txt”删除文件 hhh.txt, 系统发出提示“文件删除成功!”, 使用 ls 命令查看文件, 文件 hhh.txt 删除成功;

```
chenglei@ubuntu:test2.0/$rm hhh.txt
文件删除成功!
chenglei@ubuntu:test2.0/$ls
chenglei/test2.0/:
chenglei@ubuntu:test2.0/$
```

图 5-11 文件系统文件删除功能测试界面图

**6. 目录移除功能测试:** 如图 5-12 所示。

输入命令“rmdir test2.0”, 系统发出提示“目录删除成功!”, 使用命令 ls 查看文件, 目录 test2.0 删除成功;

```
chenglei@ubuntu:test2.0/$cd ..
chenglei@ubuntu:$rmdir test2.0
目录删除成功!
chenglei@ubuntu:$ls
chenglei/:
FILE test  0 B
DIR test1 64 B
FILE source.txt 0 B
chenglei@ubuntu:$
```

图 5-12 文件系统目录删除功能测试界面图

**8. 用户移除功能测试:** 如图 5-13 所示, 输入命令“remove chenglei 123456”, 当密码不正确时, 系统提示移除失败.如果密码正确, 系统提示“用户名正确! 密码正确!”。



```
chengleishen@ubuntu: ~/OSdesign/part5/文件系统
My File System Help Message
1.login <name> <pass>      -- 登录
2.register <name> <pass>   -- 注册
3.remove <name> <pass>     -- 移除用户
4.show                     -- 显示用户信息
5.exit                     -- exit

show
free block num: 10229

Name: scl
root: 1
Name: scl1
root: 2
Name: chenglei
root: 7

remove chenglei 1234
用户名正确！
移除失败！

remove chenglei 123456
用户名正确！
密码正确！

show
free block num: 10233

Name: scl
root: 1
Name: scl1
root: 2
```

图 5-13 文件系统用户移除

## 5.4.2 实验调试及心得

### 实验心得：

实验五需要使用单一大文件模拟磁盘实现小型文件系统，本项目是一个综合性的项目，不仅需要扎实的编程基础，还需要对操作系统文件系统方面的知识有比较深入的了解。

在进行本次文件系统的设计时我为了简化实验并未采用 UNIX 的，索引文件，i 节点表管理方式，而是使用串联文件以及空白块链的方式实现文件系统。

由于文件系统比较复杂，我很难在短时间内实现，但网络上资源很多，在了解基本原理，并构造出思路后，我借鉴了一些优秀博主的代码块，并加以完善。但 debug 的过程同样漫长，经过不懈努力，最终完成了这个项目，初步实现了文件系统的基本功能。通过这次实验，我充分了解到了不同文件系统组织形式之间的不同，加深了我对文件系统的理解！

## 附录 实验代码

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <iostream>
5. #include <string>
```

```

6.
7. #define DISC_NAME          "DISE"
8. #define BLOCK_SIZE         1024
9. #define STORAGE_SIZE       10240
10.
11. #define MAX_NAME_LENGTH    10
12. #define MAX_PSW_LENGTH     10
13.
14. #define MAX_USR_NUM         32
15. #define MAX_FILE_NUM        32
16. #define MAX_USR_NUM         32
17.
18. #define USR_EXIST           1
19. #define USR_NOT_EXIST       3
20. #define USR_FULL            2
21. #define USR_CREATE_SUCCESS  0
22. #define USR_CREATE_FAIL     -1
23.
24. #define FILE_TYPE           0
25. #define DIR_TYPE            1
26.
27. #define READ_WRITE           1
28. #define CREATE               2
29. #define SEARCH               3
30. #define nullptr              NULL
31. using namespace std;
32.
33. /* 磁盘块类(1024B) */
34. struct Block{
35.     char buffer[BLOCK_SIZE];
36. };
37. struct Usr {
38.     char username[MAX_NAME_LENGTH];
39.     char passname[MAX_PSW_LENGTH];
40.     int  root_size; /* 根目录的大小 */
41.     int  vaild;     /* 有效性 */
42.     int  root_block; /* root_block 指向根目录块 */
43. };
44. struct File_control_block {
45.     int block_num;
46.     int file_index;
47. };
48.
49.

```

```

50. struct File {
51.     char name[MAX_NAME_LENGTH*2];
52.     int file_size;
53.     int file_type;
54.     int first_block; /* first_block 指向文件第一个块 */
55. };
56.
57.
58. class File_system
59. {
60. private:
61.     int use_num; /* 当前用户编号 */
62.     FILE* fdisc; /* 磁盘文件指针 */
63.
64.     /* 目录管理 */
65.     File* rootDir; /* 根目录 */
66.     File* currentDir; /* 当前目录 */
67.     File* tempDir; /* 临时目录-用于路径解析 */
68.
69.     Block zero_block; /* 0 号引导块 */
70.     Block temp_block; /* 临时块 */
71.
72.     int currentDir_num; /* 当前目录块号 */
73.     int currentDir_index; /* 当前目录编号 */
74.     int tempDir_num; /* 临时目录块号 */
75.     int tempDir_index; /* 临时目录编号 */
76.
77.     /* 文件控制块 */
78.     File_control_block file_control;
79.     File* currentFile;
80.
81.     /* 空白块管理 */
82.     int bk_fhead; /* 空白块头指针 */
83.     int bk_nfree; /* 空白块数 */
84.
85. public:
86.     string usr_name;
87.     string dir_path;
88.
89.     File_system();
90.     ~File_system();
91.
92.     /* 文件系统基本操作 */
93.     void read_from_Disc(int block, char* buf); /* 从磁盘读取数据 */

```

```

94.     void write_to_Disc(int block, char* buf);    /* 向磁盘写入数据 */
95.     int get_block();                            /* 获取空白块 */
96.     int release_block(int block_num,int File_type); /* 释放数据
    块 */
97.     int set_File_Pointer(File**file, int block_num, int index, Block
    &block); /* 设置文件指针 */
98.     int get_zero_block();
99.     int update_zero_block();
100.
101.     /* 文件操作 */
102.     File_control_block* open_file(string &filepath, int mode);
103.     int read_from_file(File_control_block* file, int size, char *b
    uf, int pos);
104.     int write_to_file(File_control_block* file, int size, const ch
    ar* buf, int pos);
105.     int clear_file(File_control_block* file, int mode);
106.     File_control_block* close_file(File_control_block* file);
107.
108.     /* 目录操作 */
109.     int create_dir(string &filename, int mode);
110.     int remove_dir(string &filename);
111.     int delete_dir_dfs(int block, int file_size);
112.     int list_files();
113.     int cd_dir(string &filepath);
114.
115.     /* 用户操作 */
116.     int log_in(string &user, string &pass);
117.     int log_out();
118.     int Register(string &user, string &pass);
119.     int Remove(string &user, string &pass);
120.     int show_Usr();
121.
122.     /* 拓展操作 */
123.     void touch(string &filepath);
124.     void rm(string &filepath);
125.     void cat(string &filepath);
126.     void gedit(string &filepath);
127.     void mkdir(string &filepath);
128.     void rmdir(string &filepath);
129.     void show_help();
130. };
131.
132. File_system::~File_system()
133. {

```

```

134.     fclose(fdisc);
135. }
136.
137. File_system::File_system()
138. {
139.     /* 创建磁盘文件
140.      * 0 号块为引导块 */
141.     fdisc = nullptr;
142.     if(!(fdisc = fopen(DISC_NAME,"r+"))){
143.         fdisc = fopen(DISC_NAME,"a+");
144.         /* 如果磁盘文件未分配空间 */
145.         char buf[BLOCK_SIZE];
146.         memset(buf, 0, BLOCK_SIZE);
147.         /* 文件初始化与空白块成链 */
148.         for(int i=0;i<STORAGE_SIZE;i++){
149.             if(i==STORAGE_SIZE-1) *((int*)buf) = 0;
150.             else *((int*)buf) = i+1;
151.             fwrite(buf, sizeof(char), BLOCK_SIZE, fdisc);
152.         }
153.         fclose(fdisc);
154.         fdisc = fopen(DISC_NAME,"r+");
155.         /* 设置空白块的块数 */
156.         fread(buf, sizeof(char), BLOCK_SIZE, fdisc);
157.         ((int*)buf)[1] = STORAGE_SIZE - 1;
158.         fseek(fdisc, 0, SEEK_SET);
159.         fwrite(buf, sizeof(char), BLOCK_SIZE, fdisc);
160.     }
161.     fseek(fdisc, 0, SEEK_SET);
162.
163. }
164.
165. /* 文件系统基本操作 */
166.
167. /* 从磁盘读取数据
168.  * block: 磁盘块号
169.  * buf: 磁盘块数据缓冲区
170.  */
171. void File_system::read_from_Disc(int block, char* buf)
172. {
173.     fseek(fdisc, block*BLOCK_SIZE, SEEK_SET);
174.     fread(buf, sizeof(char), BLOCK_SIZE, fdisc);
175. }
176.
177. /* 向磁盘写入数据

```

```

178.     * block: 磁盘块号
179.     * buf: 磁盘块数据缓冲区
180.     */
181. void File_system::write_to_Disc(int block, char* buf)
182. {
183.     fseek(fdisc, block*BLOCK_SIZE, SEEK_SET);
184.     fwrite(buf, sizeof(char), BLOCK_SIZE, fdisc);
185. }
186.
187. /* 获取空白块
188.  * 返回值 int: 正常返回可用的空白块号, 异常返回 0
189.  */
190. int File_system::get_block()
191. {
192.     int temp_bk=bk_fhead;
193.     if(bk_nfree==0) return 0;
194.     else
195.     {
196.         char temp_buf[BLOCK_SIZE];
197.         read_from_Disc(temp_bk, temp_buf);
198.         bk_fhead = *((int*)temp_buf);
199.         bk_nfree--;
200.         return temp_bk;
201.     }
202. }
203.
204. /* 释放数据块
205.  * block_num: 文件的首个磁盘块号
206.  * File_type: 文件类型
207.  */
208. int File_system::release_block(int block_num,int File_type)
209. {
210.     Block block;
211.     if(File_type==DIR_TYPE)
212.     {
213.         read_from_Disc(block_num, block.buffer);
214.         *((int*)block.buffer)=bk_fhead;
215.         write_to_Disc(block_num, block.buffer);
216.         bk_fhead=block_num;
217.         bk_nfree++;
218.         return 0;
219.     }
220.     while(block_num!=0)
221.     {

```

```

222.         read_from_Disc(block_num, block.buffer);
223.         int tempnum=((int*)block.buffer);
224.         *((int*)block.buffer)=bk_fhead;
225.         write_to_Disc(block_num, block.buffer);
226.         bk_fhead=block_num;
227.         block_num=tempnum;
228.         bk_nfree++;
229.     }
230.     return 0;
231. }
232.
233. /* 设置文件指针
234.  * file: 文件双重指针
235.  * block_num: 文件块号
236.  * index: 文件索引
237.  * block: 缓冲区
238.  */
239. int File_system::set_File_Pointer(File**file, int block_num, int i
    ndex, Block &block)
240. {
241.     /* 从磁盘加载当前文件目录块 */
242.     read_from_Disc(block_num, block.buffer);
243.     if(!block_num) /* 当前块为 0 号引导块 */
244.         *file = &((File*)&block.buffer[8])[index];
245.     else *file = &((File*)block.buffer)[index];
246.     return 0;
247. }
248.
249. /* 获取 0 号引导块
250.  * 将 0 号引导块加载到内存
251.  */
252. int File_system::get_zero_block()
253. {
254.     read_from_Disc(0, zero_block.buffer);
255.     return 0;
256. }
257.
258. /* 更新 0 号引导块
259.  * 更新空白块情况
260.  */
261. int File_system::update_zero_block()
262. {
263.     char buf[BLOCK_SIZE];
264.     /* 更新引导块 */

```

```

265.     fseek(fdisc, 0, SEEK_SET);
266.     read_from_Disc(0, buf);
267.     *((int*)buf) = bk_fhead; /* 空白块头指针 */
268.     ((int*)buf)[1] = bk_nfree; /* 空白块数 */
269.     write_to_Disc(0, buf);
270.     return 0;
271. }
272.
273.
274.
275. /* 文件操作 */
276.
277. /* 打开文件函数
278.  * filepath: 文件路径, 仅支持相对路径
279.  * mode: 打开方式控制变量
280.  *     READ_WRITE: 对文件进行读写, 文件不存在时返回异常
281.  *     CREATE: 创建文件 (可读写), 不存在时创建文件与目录
282.  * 正常返回值 int: 若当前文件存在, 或创建成功则返回对应的编号
283.  * 异常返回 int: block_num 为-1 表示目录名与文件名冲突,
284.  *               为-2 表示当前空白块耗尽, 为-3 表示目录已满
285.  */
286. File_control_block* File_system::open_file(string &filepath, int m
ode)
287. {
288.     /* 设置临时目录为当前目录 */
289.     tempDir_num=currentDir_num;
290.     tempDir_index=currentDir_index;
291.     set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_bl
ock);
292.
293.     while (true)
294.     {
295.         /* 如果开头为/ */
296.         if(filepath[0]=='/')
297.         {
298.             /* 设置临时目录为根目录 */
299.             tempDir_num=0;
300.             tempDir_index=use_num;
301.             set_File_Pointer(&tempDir, tempDir_num, tempDir_index,
temp_block);
302.             filepath = filepath.substr(1);
303.         }
304.         else if(filepath[0]=='.'&&filepath[1]=='/') /* 如果开头
为./ */

```



```

305.         filepath = filepath.substr(2);
306.
307.         /* 如果无下级目录 */
308.         if(filepath.find('/')==filepath.npos)
309.         {
310.             /* filelist 指向文件目录项 */
311.             Block filelist_block;
312.             read_from_Disc(tempDir->first_block, filelist_block.buffer);
313.             File* filelist = (File*)filelist_block.buffer;
314.             int file_num = tempDir->file_size/32;
315.
316.             /* 查询是否已经存在该文件 */
317.             for(int i=0;i<file_num;i++)
318.             {
319.                 if(!strcmp(filelist[i].name, filepath.c_str()))
320.                 {
321.                     File_control_block* temp_control = new File_control_block;
322.                     if(filelist[i].file_type==FILE_TYPE)
323.                     {
324.                         /* 返回文件控制块信息 */
325.                         temp_control->block_num=tempDir->first_block;
326.                         temp_control->file_index=i;
327.                         return temp_control;
328.                     }
329.                     else
330.                     {
331.                         /* 当前文件名称与目录冲突 */
332.                         temp_control->block_num=-1;
333.                         temp_control->file_index=0;
334.                         return temp_control;
335.                     }
336.                 }
337.             }
338.             if(mode==READ_WRITE)
339.             {
340.                 /* READ_WRITE 状态下, 当前文件不存在, 返回异常 */
341.                 File_control_block* temp_control = new File_control_block;
342.                 temp_control->block_num=0;
343.                 temp_control->file_index=0;
344.                 return temp_control;

```

```

345.         }
346.         if(tempDir->file_size>=1024)
347.         {
348.             /* 当前目录已满 */
349.             File_control_block* temp_control = new File_control_block;
350.             temp_control->block_num=-3;
351.             temp_control->file_index=0;
352.             return temp_control;
353.         }
354.
355.         /* 从磁盘申请空白块 */
356.         int newblock = get_block();
357.         if(!newblock)
358.         {
359.             /* 当前空间已满 */
360.             File_control_block* temp_control = new File_control_block;
361.             temp_control->block_num=-2;
362.             temp_control->file_index=0;
363.             return temp_control;
364.         }
365.         /* 修改文件头文件块的指针 */
366.         Block temp;
367.         read_from_Disc(newblock, temp.buffer);
368.         *((int*)temp.buffer)=0;
369.         write_to_Disc(newblock, temp.buffer);
370.
371.         /* 修改并更新文件目录项 */
372.         tempDir->file_size += 32;
373.         write_to_Disc(tempDir_num, temp_block.buffer);
374.
375.         /* 创建新文件并更新文件块 */
376.         File* newFile = &filelist[file_num];
377.         newFile->first_block=newblock;
378.         newFile->file_size = 0;
379.         newFile->file_type = FILE_TYPE;
380.         strcpy(newFile->name,filepath.c_str());
381.         write_to_Disc(tempDir->first_block, filelist_block.buffer);
382.
383.         File_control_block* temp_control = new File_control_block;
384.         temp_control->block_num=tempDir->first_block;

```

```

385.         temp_control->file_index=file_num;
386.         return temp_control;
387.     }
388.
389.     /* 路径切分解析 */
390.     string spilt=filepath.substr(0, filepath.find('/'));
391.     filepath=filepath.substr(filepath.find('/')+1);
392.     tempDir_index=create_dir(spilt, CREATE);
393.     if(tempDir_index < 0)
394.     {
395.         File_control_block* temp_control = new File_control_block;
396.         temp_control->block_num=-4;
397.         temp_control->file_index=0;
398.         return temp_control;
399.     }
400.
401.     /* 更新当前文件目录块 */
402.     write_to_Disc(tempDir_num, temp_block.buffer);
403.     /* 加载下一级文件目录块 */
404.     tempDir_num = tempDir->first_block;
405.     set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_block);
406.     }
407. }
408.
409. /* 读取函数
410.  * file: 文件控制块指针, 指向文件控制块
411.  * size: 读取的数据大小
412.  * buf: 缓冲区
413.  * pos: 读取数据的偏移
414.  * 返回值 int: 正常返回读取数据的大小, 异常返回-1
415.  */
416. int File_system::read_from_file(File_control_block* file, int size
, char *buf, int pos)
417. {
418.     /* 当前文件无法读取 */
419.     if(file->block_num <= 0) return -1;
420.     if(pos < 0 || size < 0 || buf == nullptr) return -1;
421.
422.     /* 读取数据到缓冲区 */
423.     Block block;
424.     read_from_Disc(file->block_num, block.buffer);
425.     File* pfile = &((File*)block.buffer)[file->file_index];

```

```

426.
427.     int tempblock=pfile->first_block;
428.     int tempsize=pfile->file_size;
429.     read_from_Disc(tempblock, block.buffer);
430.
431.     int index=0;
432.     int block_length=BLOCK_SIZE-4;
433.     while(tempsize!=0)
434.     {
435.         if(!size) return index;
436.         if(!block_length){
437.             block_length=BLOCK_SIZE-4;
438.             tempblock=((int*)block.buffer);
439.             read_from_Disc(tempblock, block.buffer);
440.         }
441.         if(!pos){
442.             buf[index] = block.buffer[BLOCK_SIZE-block_length];
443.             index++;
444.             size--;
445.         }
446.         else pos--;
447.         tempsize--;
448.         block_length--;
449.     }
450.     return index;
451. }
452.
453. /* 写入函数
454.  * file: 文件控制块指针, 指向文件控制块
455.  * size: 读取的数据大小
456.  * buf: 缓冲区
457.  * pos: 读取数据的偏移, 超过文件大小默认为文件末尾
458.  * 返回值 int: 正常返回读取数据的大小, 异常返回-1
459.  */
460. int File_system::write_to_file(File_control_block* file, int size,
    const char* buf, int pos)
461. {
462.     /* 当前文件无法写入 */
463.     if(file->block_num <= 0) return -1;
464.     if(pos < 0 || size < 0 || buf == nullptr) return -1;
465.
466.     /* 创建中间缓冲区, 读取原数据 */
467.     Block block;
468.     read_from_Disc(file->block_num, block.buffer);

```

```

469.     File* pfile = &((File*)block.buffer)[file->file_index];
470.     int tempsize=pfile->file_size;
471.     if(pos>tempsize) pos=tempsize; /* 超过文件大小默认为文件末尾 */
472.     int buf_length = ((pos+size) > tempsize) ? (pos+size) : tempsize;
473.     char *tempbuf = new char[buf_length];
474.
475.     read_from_file(file, pos, tempbuf, 0);
476.     for(int i=pos;i<pos+size;i++)
477.         tempbuf[i]=buf[i-pos];
478.     if((pos+size) < tempsize)
479.         read_from_file(file, tempsize, tempbuf+pos+size, pos+size)
480.         ;
481.     /* 申请空白块 */
482.     int tempblock=get_block();
483.     if(!tempblock)
484.     {
485.         delete [] tempbuf;
486.         return -1;
487.     }
488.
489.     /* 加载磁盘块, 将当前块设置为头块 */
490.     read_from_Disc(tempblock, block.buffer);
491.     *((int*)block.buffer)=0;
492.     int head_block=tempblock;
493.
494.     /* 刷新文件 */
495.     tempsize=buf_length;
496.     int block_length=BLOCK_SIZE-4;
497.     while(tempsize)
498.     {
499.         /* 原空白块已满, 申请新空白块 */
500.         if(!block_length)
501.         {
502.             block_length=BLOCK_SIZE-4;
503.             int temp=get_block();
504.             /* 空白块申请失败 */
505.             if(!temp){
506.                 *((int*)block.buffer)=0;
507.                 write_to_Disc(tempblock, block.buffer);
508.                 release_block(head_block, FILE_TYPE);
509.                 delete [] tempbuf;
510.                 return -1;

```

```

511.         }
512.
513.         /* 更新磁盘块 */
514.         *((int*)block.buffer)=temp;
515.         write_to_Disc(tempblock, block.buffer);
516.         /* 加载磁盘块 */
517.         read_from_Disc(temp, block.buffer);
518.         tempblock=temp;
519.     }
520.
521.     block.buffer[BLOCK_SIZE-block_length]=tempbuf[buf_length-tempsize];
522.     block_length--;
523.     tempsize--;
524. }
525.
526. /* 更新磁盘块 */
527. *((int*)block.buffer)=0;
528. write_to_Disc(tempblock, block.buffer);
529.
530. read_from_Disc(file->block_num, block.buffer);
531. pfile = &((File*)block.buffer)[file->file_index];
532.
533. /*释放原有的空白块*/
534. release_block(pfile->first_block, FILE_TYPE);
535.
536. /* 更新文件目录块 */
537. pfile->file_size=buf_length;
538. pfile->first_block=head_block;
539. write_to_Disc(file->block_num, block.buffer);
540. delete [] tempbuf;
541. return 0;
542.
543. }
544.
545. /* 文件关闭函数
546.  * file: 文件控制块指针, 指向文件控制块
547.  * 返回值 int: 正常返回 nullptr, 异常返回-1
548.  */
549. File_control_block* File_system::close_file(File_control_block* file)
550. {
551.     delete file;
552.     return nullptr;

```

```

553. }
554.
555. /* 文件内容清空函数
556.  * file: 文件控制块指针, 指向文件控制块
557.  * mode: mode 为 1 时表示删除文件, 不保留任何块; mode 为 2 时表示清除内容,
      保留头块
558.  * 返回值 int: 正常返回 0, 异常返回 -1
559.  */
560. int File_system::clear_file(File_control_block* file, int mode)
561. {
562.     Block block;
563.     read_from_Disc(file->block_num, block.buffer);
564.     File* pfile = &((File*)block.buffer)[file->file_index];
565.
566.     /*释放原有的空白块*/
567.     release_block(pfile->first_block, FILE_TYPE);
568.
569.     if(mode==1) return 0;
570.     /* 从磁盘申请空白块 */
571.     int newblock = get_block();
572.     if(!newblock) return -1;
573.     /* 修改文件头文件块的指针 */
574.     Block temp;
575.     read_from_Disc(newblock, temp.buffer);
576.     *((int*)temp.buffer)=0;
577.     write_to_Disc(newblock, temp.buffer);
578.
579.     /* 更新文件目录块 */
580.     pfile->file_size=0;
581.     pfile->first_block=newblock;
582.     write_to_Disc(file->block_num, block.buffer);
583.     return 0;
584. }
585.
586.
587. /* 目录操作 */
588.
589. /* 目录创建函数, 使用之前需要设置 tempDir
590.  * filename: 目录名
591.  * mode: 打开方式控制变量
592.  *     SEARCH: 查找当前目录, 不存在时返回异常-1
593.  *     CREATE: 创建目录, 不存在时创建目录
594.  * 正常返回值 int: 若当前文件存在, 或创建成功则返回对应的目录项编号

```

```

595.     * 异常返回 int: 返回-1 表示目录名与文件名冲突, 返回-2 表示当前空白块耗尽,
      返回-3 表示目录已满
596.     */
597.     int File_system::create_dir(string &filename, int mode)
598.     {
599.         int file_num = tempDir->file_size/32;
600.         /* filelist 指向文件目录项 */
601.         Block filelist_block;
602.         read_from_Disc(tempDir->first_block, filelist_block.buffer);
603.         File* filelist = (File*)filelist_block.buffer;
604.
605.         filelist[0].file_size = tempDir_index;
606.         filelist[0].first_block = tempDir_num;
607.         Block block; File* temp_file;
608.         set_File_Pointer(&temp_file, tempDir_num, 0, block);
609.         if(!tempDir_num)
610.         {
611.             filelist[1].file_size = tempDir_index;
612.             filelist[1].first_block = tempDir_num;
613.         }
614.         else {
615.             filelist[1].file_size = temp_file->file_size;
616.             filelist[1].first_block = temp_file->first_block;
617.         }
618.
619.         write_to_Disc(tempDir->first_block, filelist_block.buffer);
620.
621.         read_from_Disc(tempDir->first_block, filelist_block.buffer);
622.         filelist = (File*)filelist_block.buffer;
623.
624.         /* 查询是否已经存在该目录 */
625.         for(int i=0;i<file_num;i++)
626.         {
627.             if(!strcmp(filelist[i].name, filename.c_str()))
628.             {
629.                 if(filelist[i].file_type==DIR_TYPE) return i;
630.                 else return -1;
631.             }
632.         }
633.
634.         /* SEARCH 状态下, 当前目录不存在, 返回异常 */
635.         if(mode==SEARCH) return -1;
636.
637.         /* CREATE 状态下, 当前目录不存在

```



```

638.      * 当前目录不存在，需要创建当前目录
639.      * 若当前目录已满，则创建失败
640.      * 否则，将创建当前目录
641.      */
642.      if(tempDir->file_size>=1024) return -3;
643.
644.      /* 从磁盘申请空白块 */
645.      int newblock = get_block();
646.      if(!newblock) return -2;
647.
648.      /* 修改并更新文件目录项 */
649.      tempDir->file_size += 32;
650.      write_to_Disc(tempDir_num, temp_block.buffer);
651.
652.
653.      /* 创建新目录并更新目录块 */
654.      File* newDir = &filelist[file_num];
655.      newDir->file_type = DIR_TYPE;
656.      newDir->first_block = newblock;
657.      newDir->file_size = 64;
658.      strcpy(newDir->name, filename.c_str());
659.      write_to_Disc(tempDir->first_block, filelist_block.buffer);
660.
661.      /* 修正当前目录的内容，添加 .与..目录 */
662.      Block temp;
663.      read_from_Disc(newblock, temp.buffer);
664.      File* pdir = (File*)temp.buffer; /* 使用文件指针 pdir 进行文件操
        作 */
665.
666.      strcpy(pdir[0].name, ".");
667.      pdir[0].file_size = file_num;
668.      pdir[0].file_type = DIR_TYPE;
669.      pdir[0].first_block = tempDir->first_block;
670.
671.      strcpy(pdir[1].name, "..");
672.      pdir[1].file_size = filelist[0].file_size;
673.      pdir[1].file_type = DIR_TYPE;
674.      pdir[1].first_block = filelist[0].first_block;
675.      write_to_Disc(newblock, temp.buffer);
676.
677.      return file_num;
678.  }
679.
680.  /* 目录移除函数，使用之前需要设置 tempDir

```

```

681.     * filename: 目录名
682.     * 返回值 int: 返回 0, 删除成功; 返回 -1, 目录不存在
683.     */
684.     int File_system::remove_dir(string &filename)
685.     {
686.         /* 使用 create_dir 函数查找当前目录是否存在 */
687.         int file_num = create_dir(filename, SEARCH);
688.         if(file_num < 0) return -1;
689.
690.         /* 加载对应的文件目录块 */
691.         Block block; File* temp;
692.         int temp_num = tempDir_num;
693.         int temp_index = tempDir_index;
694.         set_File_Pointer(&temp, tempDir->first_block, file_num, block)
        ;
695.
696.         /* 递归删除当前目录下的所有文件 */
697.         delete_dir_dfs(temp->first_block, temp->file_size);
698.
699.         /* 加载对应的文件目录块 */
700.         tempDir_num=temp_num;
701.         tempDir_index=temp_index;
702.         set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_block);
703.
704.         /* 修改文件目录块 */
705.         tempDir->file_size-=32;
706.         write_to_Disc(tempDir_num, temp_block.buffer);
707.
708.         read_from_Disc(tempDir->first_block, block.buffer);
709.         File* curr_filelist=(File*)block.buffer;
710.         while (file_num<tempDir->file_size/32)
711.         {
712.             curr_filelist[file_num]=curr_filelist[file_num+1];
713.             file_num++;
714.         }
715.         write_to_Disc(tempDir->first_block, block.buffer);
716.         return 0;
717.     }
718.
719.     /* 目录递归删除函数, 删除该目录下的所有子文件
720.     * block: 目录名
721.     * file_size: 目录文件的大小
722.     * 返回值 int: 返回 0, 目录删除成功; 返回 -1, 目录删除失败

```

```

723.  */
724.  int File_system::delete_dir_dfs(int block, int file_size)
725.  {
726.      /* filelist 指向文件目录项 */
727.      Block File_block;
728.      read_from_Disc(block, File_block.buffer);
729.      File* filelist = (File*)File_block.buffer;
730.
731.      /* 循环查询每个文件目录项 */
732.      int max_file_num = file_size/32;
733.      for(int i=2;i<max_file_num;i++)
734.      {
735.          /* 如果当前为文件，则释放所有空白块 */
736.          if(filelist[i].file_type==FILE_TYPE)
737.              release_block(filelist[i].first_block, FILE_TYPE);
738.          /* 如果当前为目录，则递归删除 */
739.          if(filelist[i].file_type==DIR_TYPE)
740.              delete_dir_dfs(filelist[i].first_block, filelist[i].fi
le_size);
741.      }
742.
743.      /* 释放当前目录块 */
744.      *((int*)File_block.buffer)=bk_fhead;
745.      bk_fhead=block;
746.      bk_nfree++;
747.      write_to_Disc(block, File_block.buffer);
748.      return 0;
749.  }
750.
751.  /* 显示目录下文件函数
752.   * 显示当前目录下的所有文件
753.   */
754.  int File_system::list_files()
755.  {
756.      /* 加载当前目录 */
757.      set_File_Pointer(&tDir, currentDir_num, currentDir_index, temp
_block);
758.
759.      Block tempblock;
760.      read_from_Disc(currentDir->first_block, tempblock.buffer);
761.
762.      cout << usr_name << "/" << dir_path << ":" << endl;
763.      for(int i=2;i<currentDir->file_size/32;i++)
764.      {

```

```

765.         if(((File*)tempblock.buffer)[i].file_type==FILE_TYPE) cout
            << "FILE ";
766.         else cout << "DIR ";
767.         cout << ((File*)tempblock.buffer)[i].name << " ";
768.         cout << ((File*)tempblock.buffer)[i].file_size;
769.         cout << " B " << endl;
770.     }
771.     return 0;
772. }
773.
774. /* 切换当前目录函数
775.  * filepath: 切换到 filepath 指向的目录, 支持绝对路径以及相对路径
776.  * 返回值 int: 返回 0 成功, 返回-1 失败
777.  */
778. int File_system::cd_dir(string &filepath)
779. {
780.     string path = dir_path;
781.     /* 从磁盘中加载当前文件目录块 */
782.     tempDir_num=currentDir_num;
783.     tempDir_index=currentDir_index;
784.     set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_block);
785.
786.     while (true) {
787.         /* 如果开头为/ */
788.         if(filepath[0]=='/')
789.         {
790.             /* 设置临时目录为根目录 */
791.             tempDir_num=0;
792.             tempDir_index=use_num;
793.             set_File_Pointer(&tempDir, tempDir_num, tempDir_index,
                temp_block);
794.             filepath = filepath.substr(1);
795.             if(filepath.length()==0)
796.             {
797.                 currentDir_num=0;
798.                 currentDir_index=use_num;
799.                 set_File_Pointer(&tempDir, currentDir_num, currentDir
                    _index, temp_block);
800.                 dir_path = "";
801.                 return 0;
802.             }
803.         }

```

```

804.         else if(filepath[0]=='.'&&filepath[1]=='/') /* 如果开头
           为./ */
805.             filepath = filepath.substr(2);
806.         else if(filepath[0]=='.'&&filepath[1]!='.') /* 如果开头为.
           当级目录 */
807.             {
808.                 filepath = filepath.substr(1);
809.                 continue;
810.             }
811.
812.         /* 如果无下级目录 */
813.         if(filepath.find('/')==filepath.npos)
814.             {
815.                 /* 使用 create_dir 函数查找当前目录是否存在 */
816.                 int file_num = create_dir(filepath, SEARCH);
817.                 if(file_num < 0) return -1;
818.
819.                 /* 设置为当前目录 */
820.                 currentDir_num=tempDir->first_block;
821.                 currentDir_index=file_num;
822.                 set_File_Pointer(&tDir, currentDir_num, currentDir_ind
                   ex, temp_block);
823.
824.                 /* 如果当前..上级目录 */
825.                 string tempstr = currentDir->name;
826.                 if(tempstr=="..")
827.                     {
828.                         /* 如果需要切换的为根目录 */
829.                         if(!currentDir->first_block)
830.                             {
831.                                 currentDir_num=0;
832.                                 currentDir_index=use_num;
833.                                 set_File_Pointer(&tDir, currentDir_num, curren
                                   tDir_index, temp_block);
834.                             }
835.                         else
836.                             {
837.                                 currentDir_index = currentDir->file_size;
838.                                 currentDir_num = currentDir->first_block;
839.                                 read_from_Disc(currentDir_num, temp_block.buff
                                   er);
840.                                 currentDir = &((File*)temp_block.buffer)[curren
                                   ntDir_index];
841.                             }

```

```

842.
843.         /* 修正当前目录 */
844.         path=path.substr(0,path.length()-1);
845.         path=path.substr(0,path.find_last_of("/")+1);
846.     }
847.     else path += (tempstr + "/");
848.
849.     dir_path = path;
850.     return 0;
851. }
852.
853. /* 路径切分解析 */
854. string spilt=filepath.substr(0, filepath.find('/'));
855. filepath=filepath.substr(filepath.find('/')+1);
856. tempDir_index=create_dir(spilt, SEARCH);
857. if(tempDir_index < 0) return -1;
858.
859. /* 更新当前文件目录块 */
860. write_to_Disc(tempDir_num, temp_block.buffer);
861. /* 加载下一级文件目录块 */
862. tempDir_num = tempDir->first_block;
863. set_File_Pointer(&tempDir, tempDir_num, tempDir_index, tem
    p_block);
864.
865. /* 如果当前..上级目录 */
866. string tempstr = tempDir->name;
867. if(tempstr=="..")
868. {
869.     /* 如果需要切换的为根目录 */
870.     if(!tempDir->first_block)
871.     {
872.         tempDir_num=0;
873.         tempDir_index=use_num;
874.         set_File_Pointer(&tempDir, tempDir_num, tempDir_in
            dex, temp_block);
875.     }
876.     else
877.     {
878.         tempDir_index = tempDir->file_size;
879.         tempDir_num = tempDir->first_block;
880.         read_from_Disc(tempDir_num, temp_block.buffer);
881.         tempDir = &((File*)temp_block.buffer)[tempDir_inde
            x];
882.     }

```

```

883.             path=path.substr(0,path.length()-1);
884.             path=path.substr(0,path.find_last_of("/") +1);
885.         }
886.         else path += (tempstr + "/");
887.     }
888. }
889.
890.
891. /* 用户操作 */
892.
893. /* 用户注册函数 */
894. int File_system::Register(string &user, string &pass)
895. {
896.     int usr_num = 0;
897.     char buf[BLOCK_SIZE];
898.
899.     /* 读取引导块信息 */
900.     read_from_Disc(0, buf);
901.     Usr* currUsr = (Usr*)&buf[8];
902.     bk_fhead = *((int*)buf);
903.     bk_nfree = ((int*)buf)[1];
904.
905.     /* 查找当前是否存在该用户 */
906.     for(;usr_num<MAX_USR_NUM;usr_num++)
907.     {
908.         if(!currUsr[usr_num].vaild) break;
909.         if(!strcmp(currUsr[usr_num].usrname, user.c_str()))
910.         {
911.             cout << "当前用户已存在！" << endl;
912.             return USR_EXIST;
913.         }
914.     }
915.     if(usr_num==MAX_USR_NUM) return USR_FULL;
916.
917.     /* 创建用户 */
918.     int root = get_block();
919.     if(!root) return USR_CREATE_FAIL;
920.     update_zero_block();
921.
922.     read_from_Disc(0, buf);
923.     currUsr = (Usr*)&buf[8];
924.
925.     currUsr[usr_num].root_block = root;
926.     currUsr[usr_num].root_size = 64;

```

```

927.     currUsr[usr_num].vaild=1;
928.     strcpy(currUsr[usr_num].usrname,user.c_str());
929.     strcpy(currUsr[usr_num].passname,pass.c_str());
930.     write_to_Disc(0, buf);
931.
932.     /* 添加 .与..目录 */
933.     read_from_Disc(currUsr[usr_num].root_block, buf);
934.     File* pdir = (File*)buf; /* 使用文件指针 pdir 进行文件操作 */
935.     strcpy(pdir[0].name,".");
936.     pdir[0].file_size = usr_num;
937.     pdir[0].file_type = DIR_TYPE;
938.     pdir[0].first_block = 0;
939.
940.     strcpy(pdir[1].name,"..");
941.     pdir[1].file_size = usr_num;
942.     pdir[1].file_type = DIR_TYPE;
943.     pdir[1].first_block = 0;
944.
945.     /* 将修改结果写回 */
946.     write_to_Disc(root, buf);
947.     cout << "用户创建成功!" << endl;
948.     return USR_CREATE_SUCCESS;
949. }
950.
951. /* 用户移除函数 */
952. int File_system::Remove(string &user, string &pass)
953. {
954.     int num = log_in(user, pass);
955.     if(num != USR_EXIST)
956.     {
957.         cout << "移除失败! " << endl;
958.         return -1;
959.     }
960.     delete_dir_dfs(rootDir->first_block, rootDir->file_size);
961.     log_out();
962.
963.     char buf[BLOCK_SIZE];
964.     /* 读取引导块信息 */
965.     read_from_Disc(0, buf);
966.     Usr* currUsr = (Usr*)&buf[8];
967.     int i=use_num;
968.     for(;i<MAX_USR_NUM;i++)
969.     {
970.         if(!currUsr[i+1].vaild) break;

```



```

971.         currUsr[i] = currUsr[i+1];
972.     }
973.     currUsr[i].vaild = 0;
974.     write_to_Disc(0, buf);
975.     return 0;
976. }
977.
978. /* 用户登录函数 */
979. int File_system::log_in(string &user, string &pass)
980. {
981.     int usr_num = 0;
982.     char buf[BLOCK_SIZE];
983.     /* 读取引导块信息 */
984.     read_from_Disc(0, buf);
985.     Usr* currUsr = (Usr*)&buf[8];
986.     /* 查找当前是否存在该用户 */
987.     for(;usr_num<MAX_USR_NUM;usr_num++)
988.     {
989.         if(!currUsr[usr_num].vaild) return USR_NOT_EXIST;
990.         if(!strcmp(currUsr[usr_num].usrname, user.c_str()))
991.         {
992.             cout << "用户名正确！" << endl;
993.             if(!strcmp(currUsr[usr_num].passname, pass.c_str()))
994.             {
995.                 cout << "密码正确！" << endl;
996.                 use_num = usr_num;
997.                 /* 设置当前的根目录 读取引导块 */
998.                 read_from_Disc(0, zero_block.buffer);
999.                 currUsr = (Usr*)&zero_block.buffer[8];
1000.                 rootDir = (File*)&currUsr[usr_num];
1001.
1002.                 /* 设置当前目录以及临时目录的目录块 */
1003.                 currentDir=rootDir;
1004.                 currentDir_num=0;
1005.                 currentDir_index=usr_num;
1006.                 tempDir=rootDir;
1007.                 tempDir_num=0;
1008.                 tempDir_index=usr_num;
1009.
1010.                 /* 加载空白块处理逻辑 */
1011.                 read_from_Disc(0, buf);
1012.                 bk_fhead = *((int*)buf); /* 空白块头指针 */
1013.                 bk_nfree = ((int*)buf)[1]; /* 空白块数 */
1014.                 usr_name = currUsr[usr_num].usrname;

```

```

1015.             return USR_EXIST;
1016.         }
1017.     }
1018. }
1019. return USR_NOT_EXIST;
1020. }
1021.
1022. /* 用户登出函数 */
1023. int File_system::log_out()
1024. {
1025.     /* 更新引导块 */
1026.     update_zero_block();
1027.     usr_name = "";
1028.     dir_path = "";
1029.     return 0;
1030. }
1031.
1032. /* 用户显示函数 */
1033. int File_system::show_usr()
1034. {
1035.     int usr_num = 0;
1036.     char buf[BLOCK_SIZE];
1037.     /* 读取引导块信息 */
1038.     read_from_Disc(0, buf);
1039.
1040.     cout << "free block num: " << ((int*)buf)[1] << endl << endl;
1041.     Usr* currUsr = (Usr*)&buf[8];
1042.     /* 查找当前是否存在该用户 */
1043.     for(;usr_num<MAX_USR_NUM;usr_num++)
1044.     {
1045.         if(!currUsr[usr_num].vaild) break;
1046.         cout << "Name: " << currUsr[usr_num].username << endl;
1047.         cout << "root: " << currUsr[usr_num].root_block << endl;
1048.     }
1049.     return 0;
1050. }
1051. }
1052.
1053.
1054. /* 拓展操作 */
1055.
1056. void File_system::touch(string &filepath)
1057. {
1058.     File_control_block* file = open_file(filepath, CREATE);

```

```

1059.     if(file->block_num==-1) cout << "文件名与目录名冲突!" << endl;
1060.     else if(file->block_num==-2) cout << "空白块耗尽!" << endl;
1061.     else if(file->block_num==-3) cout << "目录已满!" << endl;
1062.     else if(file->block_num==-4) cout << "路径目录创建失
        败!" << endl;
1063. }
1064.
1065. void File_system::rm(string &filepath)
1066. {
1067.     File_control_block* file = open_file(filepath, READ_WRITE);
1068.     if(file->block_num<=0) cout << "当前文件不存在!" << endl;
1069.     else {
1070.         clear_file(file, 1);
1071.         Block block;
1072.         /* 加载对应的文件目录块 */
1073.         read_from_Disc(file->block_num, block.buffer);
1074.         File* filelist = (File*)block.buffer;
1075.
1076.         /* 修改文件目录块 */
1077.         tempDir_num=filelist[0].first_block;
1078.         tempDir_index=filelist[0].file_size;
1079.
1080.         set_File_Pointer(&tempDir, tempDir_num, tempDir_index, tem
            p_block);
1081.         tempDir->file_size-=32;
1082.         write_to_Disc(tempDir_num, temp_block.buffer);
1083.
1084.         int file_num = file->file_index;
1085.         while (file_num<tempDir->file_size/32)
1086.         {
1087.             filelist[file_num]=filelist[file_num+1];
1088.             file_num++;
1089.         }
1090.         write_to_Disc(file->block_num, block.buffer);
1091.         cout << "文件删除成功!" << endl;
1092.     }
1093. }
1094.
1095. void File_system::cat(string &filepath)
1096. {
1097.     File_control_block* file = open_file(filepath, READ_WRITE);
1098.     if(file->block_num<=0) cout << "当前文件不存在!" << endl;
1099.     else {
1100.         Block block;

```

```

1101.         read_from_Disc(file->block_num, block.buffer);
1102.         File* pfile = &((File*)block.buffer)[file->file_index];
1103.
1104.         int size = pfile->file_size;
1105.         char *buf = new char[size];
1106.         read_from_file(file, size, buf, 0);
1107.         for(int i=0;i<size;i++)
1108.             cout << buf[i];
1109.         cout << endl;
1110.         delete[] buf;
1111.     }
1112. }
1113.
1114. void File_system::gedit(string &filepath)
1115. {
1116.     File_control_block* file = open_file(filepath, READ_WRITE);
1117.     if(file->block_num<=0) cout << "当前文件不存在!" << endl;
1118.     else {
1119.         /* 显示原有文本内容 */
1120.         cat(filepath);
1121.
1122.         /* 读入新的文本 */
1123.         int size = 0; char c;
1124.         char *buf = nullptr;
1125.         cout << "请输入需要输入的文本(\\n 结尾):" << endl;
1126.         while((c=getchar())!='\\n')
1127.         {
1128.             if(!(size % BLOCK_SIZE))
1129.             {
1130.                 if(!buf) buf = new char[BLOCK_SIZE];
1131.                 else {
1132.                     int buf_num = size/BLOCK_SIZE;
1133.                     char *tempbuf = new char[buf_num*BLOCK_SIZE];
1134.
1135.                     for(int i=0;i<size;i++)
1136.                         tempbuf[i] = buf[i];
1137.                     delete [] buf;
1138.                     buf = tempbuf;
1139.                 }
1140.                 buf[size++] = c;
1141.             }
1142.             clear_file(file, 2);
1143.             write_to_file(file, size, buf, 0);

```

```

1144.     }
1145. }
1146.
1147. void File_system::mkdir(string &filepath)
1148. {
1149.     /* 从磁盘中加载当前文件目录块 */
1150.     tempDir_num=currentDir_num;
1151.     tempDir_index=currentDir_index;
1152.     set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_block);
1153.
1154.     while (true) {
1155.         /* 如果开头为/ */
1156.         if(filepath[0]=='/')
1157.         {
1158.             /* 设置临时目录为根目录 */
1159.             tempDir_num=0;
1160.             tempDir_index=use_num;
1161.             set_File_Pointer(&tempDir, tempDir_num, tempDir_index,
temp_block);
1162.             filepath = filepath.substr(1);
1163.         }
1164.         else if(filepath[0]=='.'&&filepath[1]=='/') /* 如果开头
为./ */
1165.             filepath = filepath.substr(2);
1166.
1167.         /* 如果无下级目录 */
1168.         if(filepath.find('/')==filepath.npos)
1169.         {
1170.             int file_num = create_dir(filepath, CREATE);
1171.             if(file_num==-1) cout << "目录名与文件名冲
突!" << endl;
1172.             else if(file_num==-2) cout << "空白块耗尽!" << endl;
1173.             else if(file_num==-3) cout << "目录已满!" << endl;
1174.             return;
1175.         }
1176.
1177.         /* 路径切分解析 */
1178.         string spilt=filepath.substr(0, filepath.find('/'));
1179.         filepath=filepath.substr(filepath.find('/')+1);
1180.         tempDir_index=create_dir(spilt, CREATE);
1181.         if(tempDir_index < 0){
1182.             cout << "路径目录创建失败!" << endl;
1183.             return;

```

```

1184.     }
1185.     /* 更新当前文件目录块 */
1186.     write_to_Disc(tempDir_num, temp_block.buffer);
1187.     /* 加载下一级文件目录块 */
1188.     tempDir_num = tempDir->first_block;
1189.     set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_block);
1190. }
1191. }
1192.
1193. void File_system::rmdir(string &filepath)
1194. {
1195.     /* 从磁盘中加载当前文件目录块 */
1196.     tempDir_num=currentDir_num;
1197.     tempDir_index=currentDir_index;
1198.     set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_block);
1199.
1200.     while (true) {
1201.         /* 如果开头为/ */
1202.         if(filepath[0]=='/')
1203.         {
1204.             /* 设置临时目录为根目录 */
1205.             tempDir_num=0;
1206.             tempDir_index=use_num;
1207.             set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_block);
1208.             filepath = filepath.substr(1);
1209.         }
1210.         else if(filepath[0]=='.'&&filepath[1]=='/') /* 如果开头为./ */
1211.             filepath = filepath.substr(2);
1212.
1213.         /* 如果无下级目录 */
1214.         if(filepath.find('/')==filepath.npos)
1215.         {
1216.             int status = remove_dir(filepath);
1217.             if(status < 0) cout << "目录删除失败!" << endl;
1218.             else cout << "目录删除成功!" << endl;
1219.             return;
1220.         }
1221.
1222.         /* 路径切分解析 */
1223.         string spilt=filepath.substr(0, filepath.find('/'));

```

```

1224.         filepath=filepath.substr(filepath.find('/')+1);
1225.         tempDir_index=create_dir(spilt, READ_WRITE);
1226.         if(tempDir_index < 0){
1227.             cout << "路径目录不存在!" << endl;
1228.             return;
1229.         }
1230.         /* 更新当前文件目录块 */
1231.         write_to_Disc(tempDir_num, temp_block.buffer);
1232.         /* 加载下一级文件目录块 */
1233.         tempDir_num = tempDir->first_block;
1234.         set_File_Pointer(&tempDir, tempDir_num, tempDir_index, temp_block);
1235.     }
1236.
1237. }
1238.
1239. void File_system::show_help()
1240. {
1241.     cout << "                My File System Help Message
1242.     " << endl;
1243.     cout << "     _____
1244.     -" << endl;
1245.     cout << "     1.touch <filename>                -- 新建文件 " << endl;
1246.     cout << "     2.rm <filename>                    -- 删除文件 " << endl;
1247.     cout << "     3.cat <filename>                  -- 查看文件 " << endl;
1248.     cout << "     4.gedit <filename>                -- 编辑文件 " << endl;
1249.     cout << "     5.mkdir <dirname>                -- 新建目录 " << endl;
1250.     cout << "     6.rmdir <filename>               -- 移除目录 " << endl;
1251.     cout << "     7.ls                             -- 显示目录文件 " << endl;
1252.     cout << "     8.cd <dirname>                   -- 进入目录 " << endl;
1253.     cout << "     9.clear                           -- 清除屏幕内容 " << endl;
1254.     cout << "    10.help                           -- 显示帮助 " << endl;
1255.     cout << "    11.exit                           -- 注销用户 " << endl;

```

```

1254.     cout << "      _____
      -" << endl;
1255. }
1256.
1257.
1258. bool all_blank(string &str)
1259. {
1260.     for(int i=0;i<str.length();i++)
1261.         if(str[i]!=' ') return false;
1262.     return true;
1263. }
1264.
1265. int main()
1266. {
1267.     system("clear");
1268.     File_system *file_system = new File_system();
1269.
1270.     while (true){
1271.         string cmd, op, sname, dname;
1272.         getline(cin, cmd, '\n');
1273.         system("clear");
1274.         cout << "                My File System Help Message      " <<
            endl;
1275.         cout << "      _____
      -" << endl;
1276.         cout << "          1.login <name> <pass>          -- 登
录      " << endl;
1277.         cout << "          2.register <name> <pass>        -- 注
册      " << endl;
1278.         cout << "          3.remove <name> <pass>        -- 移除用
户      " << endl;
1279.         cout << "          4.show                          -- 显示用户信
息      " << endl;
1280.         cout << "          5.exit                          -- exit
      " << endl;
1281.         cout << "      _____
      -" << endl;
1282.
1283.         getline(cin, cmd, '\n');
1284.         /* 如果只输入空格 */
1285.         if(all_blank(cmd)) continue;
1286.         /* 删除两端空格 */
1287.         cmd = cmd.substr(cmd.find_first_not_of(' '),

```



```

1288.                cmd.find_last_not_of(' ') - cmd.find_firs
        t_not_of(' ') + 1);
1289.        if(cmd.length() == 0) continue;
1290.
1291.        /* 切分指令 */
1292.        if(cmd.find(" ") == cmd.npos) op = cmd;
1293.        else{
1294.            op = cmd.substr(0, cmd.find(" "));
1295.            dname = cmd.substr(cmd.rfind(" ") + 1);
1296.
1297.            /* 判断是否为双输入指令 */
1298.            sname = cmd.substr(cmd.find(" "), cmd.rfind(" ")-cmd.f
        ind(" ")+1);
1299.            if(!all_blank(sname)){
1300.                sname = sname.substr(sname.find_first_not_of(' '),
1301.
                                sname.find_last_not_of(' ')-s
        name.find_first_not_of(' ')+1);
1302.            }
1303.        }
1304.
1305.        if(op == "login"){
1306.            if(file_system->log_in(sname, dname)!=USR_EXIST){
1307.                cout << "登录失败! " << endl;
1308.                continue;
1309.            }
1310.        }
1311.        else if(op == "register"){ file_system->Register(sname, dn
        ame); continue;}
1312.        else if(op == "remove"){ file_system->Remove(sname, dname)
        ; continue;}
1313.        else if(op == "show"){ file_system->show_usr(); continue;}
1314.        else if(op == "exit"){ break;}
1315.        else continue;
1316.        file_system->show_help();
1317.        file_system->dir_path = "";
1318.
1319.
1320.        while (true)
1321.        {
1322.            setbuf(stdin, nullptr);
1323.            bool is_cp = false;

```

```

1324.         cout << file_system->usr_name << "@ubuntu:" << file_sy
            stem->dir_path << "$";
1325.
1326.         getline(cin, cmd, '\n');
1327.
1328.         /* 如果只输入空格 */
1329.         if(all_blank(cmd)) continue;
1330.         /* 删除两端空格 */
1331.         cmd = cmd.substr(cmd.find_first_not_of(' '),
1332.             cmd.find_last_not_of(' ') - cmd.find_
            first_not_of(' ') + 1);
1333.         if(cmd.length() == 0) continue;
1334.
1335.         /* 切分指令 */
1336.         if(cmd.find(" ") == cmd.npos) op = cmd;
1337.         else{
1338.             op = cmd.substr(0, cmd.find(" "));
1339.             dname = cmd.substr(cmd.rfind(" ") + 1);
1340.
1341.             /* 判断是否为 cp 指令 */
1342.             sname = cmd.substr(cmd.find(" "), cmd.rfind(" ") - c
            md.find(" ")+1);
1343.             if(!all_blank(sname)){
1344.                 is_cp = true;
1345.                 sname = sname.substr(sname.find_first_not_of('
            '),
1346.                     sname.find_last_not_of('
            ')-sname.find_first_not_of(' ')+1);
1347.             }
1348.         }
1349.
1350.         if(op == "touch") file_system->touch(dname);
1351.         if(op == "rm") file_system->rm(dname);
1352.         if(op == "cat") file_system->cat(dname);
1353.         if(op == "gedit")file_system->gedit(dname);
1354.         if(op == "mkdir") file_system->mkdir(dname);
1355.         if(op == "rmdir") file_system->rmdir(dname);
1356.         if(op == "ls") file_system->list_files();
1357.         if(op == "cd")
1358.             if(file_system->cd_dir(dname)<0)
1359.                 cout << "当前目录不存在!" << endl;
1360.         if(op == "clear") { system("clear"); file_system->show
            _help(); }
1361.         if(op == "help") file_system->show_help();

```

```
1362.         if(op == "exit") { file_system->log_out();break; }
1363.         file_system->update_zero_block();
1364.     }
1365. }
1366. delete file_system;
1367. return 0;
1368. }
```