



华中科技大学

操作系统原理课程设计报告

姓 名： 范 唯
学 院： 计算机科学与技术
专 业： 计算机科学与技术
班 级： CS1703
学 号： U201714670
指导教师： 谢 夏

分数	
教师签名	

2020 年 6 月 30 日

目 录

1	实验一 Linux 编程的相关知识	1
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	2
1.4	实验调试	3
1.4.1	实验步骤	3
1.4.2	实验调试及心得	4
	附录 实验代码	4
2	实验二 新增系统调用实现.....	13
2.1	实验目的	13
2.2	实验内容	13
2.3	实验设计	13
2.3.1	开发环境	13
2.3.2	实验设计	13
2.4	实验调试	13
2.4.1	实验步骤	13
2.4.2	实验调试及心得	16
	附录 实验代码	17
3	实验三 增加设备驱动程序.....	19
3.1	实验目的	19
3.2	实验内容	19
3.3	实验设计	19
3.3.1	开发环境	19
3.3.2	实验设计	19
3.4	实验调试	21
3.4.1	实验步骤	21
3.4.2	实验调试及心得	23
	附录 实验代码	24
4	实验四 使用 QT 实现系统监控器.....	27
4.1	实验目的	27
4.2	实验内容	27
4.3	实验设计	27
4.3.1	开发环境	27
4.3.2	实验设计	27
4.4	实验调试	28
4.4.1	实验步骤	28

4.4.2	实验调试及心得	32
附录	实验代码	32
5	实验五 小型模拟文件系统.....	49
5.1	实验目的	49
5.2	实验内容	49
5.3	实验设计	50
5.3.1	开发环境	50
5.3.2	实验设计	50
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	52
附录	实验代码	56

1 实验一 Linux 编程的相关知识

1.1 实验目的


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

1.2 实验内容

- (1) 编一个 C 程序，其内容为实现文件拷贝的功能。
- (2) 基本要求:使用系统调用 *open/read/write...*；选择:容错、*cp*。
- (3) 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)
- (4) 基本要求:三个独立子进程各自窗口显示；

1.3 实验设计

Vmware 安装虚拟机 Ubuntu 知乎教程链接

 <https://zhuanlan.zhihu.com/p/38797088>

VMware 虚拟机扩展 Ubuntu 系统磁盘空间教程

 https://blog.csdn.net/daemon_2017/article/details/80660372

1.3.1 开发环境

- Windows10
- VMware Workstation 15 Pro
- 虚拟机 Ubuntu19.10 (60G)
- 内核 Linux 5.4.21
- 核数: 8
- 内存: 4

1.3.2 实验设计

(task1) 编写 Get_Copy_Get 文件复制程序 这是一个样例, 不涉及并发.

```
1.  int main(int argc, char **argv) {
2.  //缓冲区大小
3.      int from_fd, to_fd;
4.      int bytes_read, bytes_write;
5.      char buffer[BUFFER_SIZE];
6.      char *ptr;
7.      if(argc!=3) //三个参数
8.      {
9.          fprintf(stderr, "Usage:%s fromfile tofile\n\a", argv[0]);
10.         return(-1);
11.     }
12.     /* 打开源文件 */
13.     if((from_fd=open(argv[1], O_RDONLY))==-1)
14.     {
15.         fprintf(stderr, "Open %s Error:%s\n", argv[1], strerror(errno));
16.         return(-1);
17.     }
18.     /* 创建目的文件 */
19.     if((to_fd=open(argv[2], O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR))==-1) {
20.         fprintf(stderr, "Open %s Error:%s\n", argv[2], strerror(errno));
21.         //设定一个缓冲区
22.         return(-1);
23.     }
24.     /* 以下代码是一个经典的拷贝文件的代码 */
25.     while(bytes_read=read(from_fd, buffer, BUFFER_SIZE)) {
26.         /* 一个致命的错误发生了 */
27.         if((bytes_read==-1)&&(errno!=EINTR)) break;
28.         else if(bytes_read>0)
29.         {
30.             ptr=buffer;
31.             while(bytes_write=write(to_fd, ptr, bytes_read))
32.             {
33.                 /* 一个致命错误发生了 */
34.                 if((bytes_write==-1)&&(errno!=EINTR)) break;
35.                 /* 写完了所有读的字节 */
36.                 else if(bytes_write==bytes_read) break;
37.                 /* 只写了一部分, 继续写 */
38.                 else if(bytes_write>0)
39.                 {
40.                     ptr+=bytes_write;
```

```

41.             bytes_read==bytes_write;
42.         }
43.     }
44. /* 写的时候发生的致命错误 */ if(bytes_write==-1)break;
45.     }
46.     }
47.     close(from_fd);
48.     close(to_fd);
49.
50.     return(1);
51. }

```

(task2) 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）

因为 GTK 之前没有接触过, 而且有一点点古老, 所以我选择了 Qt5. 我一共做了四个并发进程, 通过 fork() 来实现. 分别监视系统世界、CPU 使用率、内存使用率、磁盘使用率。

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，整个流程还是比较清晰与简单的。知识在做前端时会遇到一些奇怪的格式问题。在后续的实验中所做的资源管理器其实和它实现过程也十分相似。

1.4 实验调试

1.4.1 实验步骤

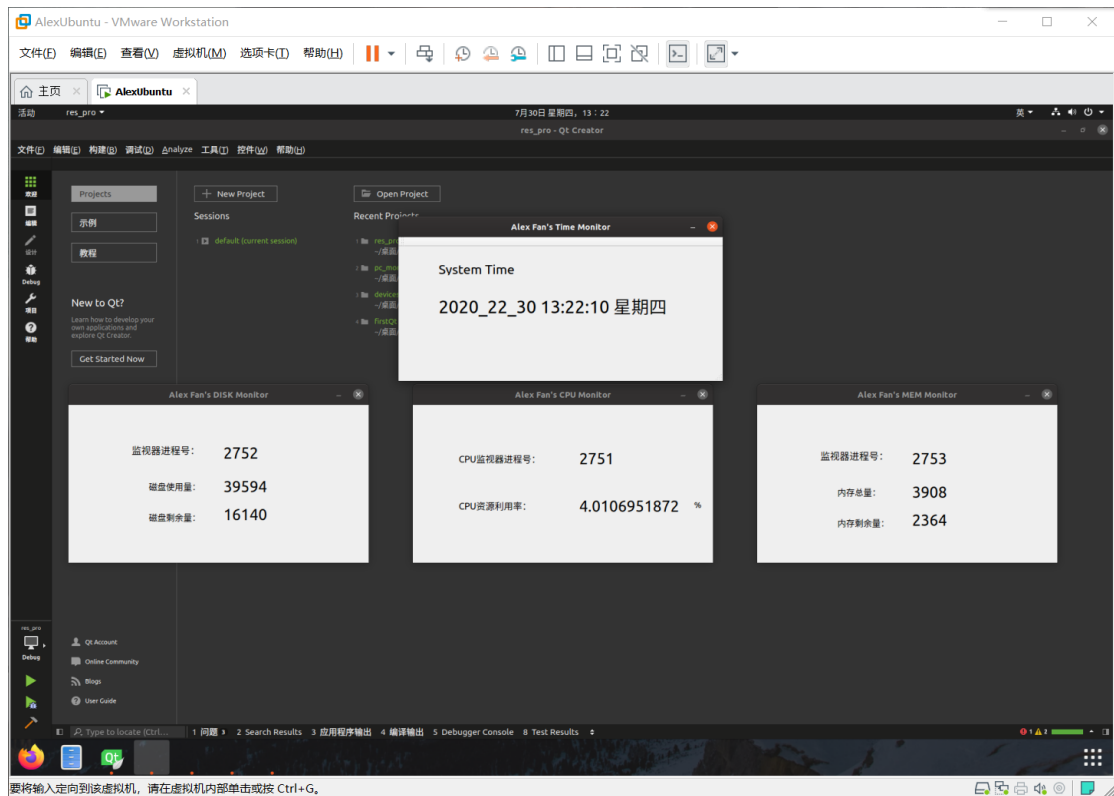
(1) task1:

由于在代码当中已经将需要复制的文本名写死，所以不需要在终端输入需要复制的文本，直接执行编译出来的可执行文件即可。执行结果如下图 1-1 所示，打开文件即可判断两个文件是否相同。

(2) task2:

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，整个流程还是比较清晰与简单的。知识在做前端时会遇到一些奇怪的格式问题。在后续的实验中所做的资源管理器其实和它实现过程也十分相似。具体步骤如下：

1. 从系统资源文件中获取 CPU MEM TIME 等信息
2. 每秒刷新一次，并通过 slot 传输到 QT
3. 编写 main 界面
4. 编写 CPU 信息界面、编写 MEM 信息界面、编写 TIME 信息界面
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

- (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) Mian.cpp

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

```

(11) {
(12)     int pid;
(13)     if((pid = fork()) == 0){
(14)         QApplication a(argc,argv);
(15)         res_pro w;
(16)         w.setWindowTitle("Alex Fan's CPU Monitor");
(17)         w.show();
(18)         a.exec();
(19)         exit(0);
(20)     }
(21)     if((pid = fork()) == 0){
(22)         QApplication a(argc,argv);
(23)         res_disk w;
(24)         w.setWindowTitle("Alex Fan's DISK Monitor");
(25)         w.show();
(26)         a.exec();
(27)         exit(0);
(28)     }
(29)     if((pid = fork()) == 0){
(30)         QApplication a(argc,argv);
(31)         res_mem w;
(32)         w.setWindowTitle("Alex Fan's MEM Monitor");
(33)         w.show();
(34)         a.exec();
(35)         exit(0);
(36)     }
(37)     QApplication a(argc, argv);
(38)     MainWindow w;
(39)     w.setWindowTitle("Alex Fan's Time Monitor");
(40)     w.show();
(41)
(42)     return a.exec();
(43) }

```

(2) mainwindow.cpp

```

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

```



```

10.     this->move(675,200);
11.     QTimer*timer = new QTimer (this);
12.     connect(timer,SIGNAL(timeout()),this,SLOT(timerUpdate()));
13.     timer->start(1000);
14. }
15.
16. MainWindow::~MainWindow()
17. {
18.     delete ui;
19. }
20.
21. void MainWindow::timerUpdate(void)
22. {
23.     QDateTime time = QDateTime::currentDateTime();
24.     QString str = time.toString("yyyy_mm_dd hh:mm:ss dddd");
25.     ui->Time->setText(str);
26. }

```

(3) mainwindow.h

```

1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3. #include <QTimer>
4. #include <QDateTime>
5. #include <QMainWindow>
6. #include <res_pro.h>
7. namespace Ui {
8. class MainWindow;
9. }
10.
11. class MainWindow : public QMainWindow
12. {
13.     Q_OBJECT
14.
15. public:
16.     explicit MainWindow(QWidget *parent = nullptr);
17.     ~MainWindow();
18.
19. private:
20.     Ui::MainWindow *ui;
21.
22. public slots:
23.     void timerUpdate(void);
24.
25.

```

```
26. };
```

(4)mainwindow.ui

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <ui version="4.0">
3.   <class>MainWindow</class>
4.   <widget class="QMainWindow" name="MainWindow">
5.     <property name="geometry">
6.       <rect>
7.         <x>0</x>
8.         <y>0</y>
9.         <width>565</width>
10.        <height>257</height>
11.      </rect>
12.    </property>
13.    <property name="windowTitle">
14.      <string>MainWindow</string>
15.    </property>
16.    <widget class="QWidget" name="centralWidget">
17.      <widget class="QLabel" name="Time">
18.        <property name="geometry">
19.          <rect>
20.            <x>70</x>
21.            <y>70</y>
22.            <width>411</width>
23.            <height>71</height>
24.          </rect>
25.        </property>
26.        <property name="font">
27.          <font>
28.            <pointsize>23</pointsize>
29.          </font>
30.        </property>
31.        <property name="text">
32.          <string>Time</string>
33.        </property>
34.      </widget>
35.      <widget class="QLabel" name="timetext">
36.        <property name="geometry">
37.          <rect>
38.            <x>70</x>
39.            <y>30</y>
40.            <width>151</width>
41.            <height>21</height>
```

```

42.     </rect>
43. </property>
44. <property name="font">
45.     <font>
46.         <pointsize>17</pointsize>
47.     </font>
48. </property>
49. <property name="text">
50.     <string>System Time</string>
51. </property>
52. </widget>
53. </widget>
54. <widget class="QMenuBar" name="menuBar">
55.     <property name="geometry">
56.         <rect>
57.             <x>0</x>
58.             <y>0</y>
59.             <width>565</width>
60.             <height>28</height>
61.         </rect>
62.     </property>
63. </widget>
64. <widget class="QToolBar" name="mainToolBar">
65.     <attribute name="toolBarArea">
66.         <enum>TopToolBarArea</enum>
67.     </attribute>
68.     <attribute name="toolBarBreak">
69.         <bool>false</bool>
70.     </attribute>
71. </widget>
72. <widget class="QStatusBar" name="statusBar"/>
73. </widget>
74. <layoutdefault spacing="6" margin="11"/>
75. <resources/>
76. <connections/>
77. </ui>

```

(5) res_disk.cpp

```

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(100,500);
14.
15.     int pid = getpid();
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.
48.                 Disk_Free = lst[3].toDouble()/1024.0;
49.                 ui->Disk_Free->setText(QString::number(Disk_Free,'f',0));
49.             }

```

```

50.         //qDebug("Disk Used:%.01fMB Free:%.01fMB",lst[2].toDouble()/
        1024.0,lst[3].toDouble()/1024.0);
51.     }
52. }
53. }
54. res_disk::~res_disk()
55. {
56.     delete ui;
57. }

```

(6) res_mem.cpp

```

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(1300,500);
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.     timer->start(1000);
27.
28. }
29. void res_mem::Update()
30. {
31.     QProcess process;
32.     process.start("free -m");           //使用 free 完成获取

```

```

33.     process.waitForFinished();
34.     process.readLine();
35.     QString str = process.readLine();
36.     str.replace("\n", "");
37.     str.replace(QRegExp("( ){1,}"), " "); //将连续空格替换为单个空格 用于分割
38.     auto lst = str.split(" ");
39.     if(lst.size() > 6)
40.     {
41.         Mem_Used = lst[1].toDouble();
42.         ui->Mem_Monitor->setText(QString::number(Mem_Used, 'f', 0));
43.         Mem_Free = lst[6].toDouble();
44.         ui->Mem_Free->setText(QString::number(Mem_Free, 'f', 0));
45.         qDebug("Mem Total:%.01fMB Free:%.01fMB", lst[1].toDouble(), lst[6].toDouble());
46.     }
47. }
48. res_mem::~res_mem()
49. {
50.     delete ui;
51. }

```

(7) res_pro.cpp

```

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(700, 500);
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].toDouble
        ();
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__) * 100.0;
45.
46.             ui->CPU_Monitor->setText(QString::number(usage, 'f', 10));
47.             //qDebug("cpu usage: %.2lf%%", (use - m_cpu_use__) / (total - m_cp
            u_total__) * 100.0);
48.             m_cpu_total__ = total;
49.             m_cpu_use__ = use;
50.         }
51.     }
52. }
53.
54. res_pro::~res_pro()
55. {
56.     delete ui;
57. }

```

2 实验二 新增系统调用实现

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，并且编译，生成该新的内核。使用新的内核进行启动。

(2) 对新增的系统函数进行调用。

2.3 实验设计

2.3.1 开发环境

- Windows10
- VMware Workstation 15 Pro
- 虚拟机 Ubuntu19.10 (60G)
- 内核 Linux 5.4.21
- 核数: 8
- 内存: 4

2.3.2 实验设计

- (1) 从官网下载新的内核
- (2) 向内核中加入自动新加入的内核
- (3) 编译新的内核
- (4) 编写测试程序调用内核，验证新的系统调用是否加入成功

2.4 实验调试

2.4.1 实验步骤

以下内容基于 Ubuntu19.10 + 内核 Linux5.4.21, 如果是 4.x.x 版本的内核, 参考以下下文链接:

基于 Linux4 代内核的新增系统调用步骤:

https://blog.csdn.net/qq_41175905/article/details/80529245

1. 配置虚拟机

Vmware 安装虚拟机 Ubuntu 知乎教程链接

<https://zhuanlan.zhihu.com/p/38797088>

VMware 虚拟机扩展 Ubuntu 系统磁盘空间教程

https://blog.csdn.net/daemon_2017/article/details/80660372

2. 更换 Ubuntu 源为清华源

更换镜像源链接

<https://blog.csdn.net/yumeil998/article/details/83214433>

注意你的 Ubuntu 版本, 选择适配的镜像源, 可加快安装包的速度.

3. 一键配置所有所需要的包

换好镜像源后直接在命令行里配置就行了.

```
apt-get install libncurses-dev flex bison openssl libssl-dev dkms libelf-dev libudev-dev libpci-dev libiberty-dev autoconf
```

后续如果缺其他包就安装系统提示下载就好了

4. 内核下载

别去 www.kernel.org 下了, 速度实在不忍直视

直接通过下述链接下载配置.

kernel 镜像源链接

<http://ftp.sjtu.edu.cn/sites/ftp.kernel.org/pub/linux/kernel/>

选择合适的版本挑一个下载即可.

5. 安装内核

- ctrl+alt+t 快速打开 CMD, 并且输入 `sudo su` 获得管理员权限.
- 将下载好的内核 move 到 `/usr/src` 目录下, 使用命令

1. `sudo mv **/**/**/linux*** /usr/src/`
2. `/**`代表你刚下载的 `linux` 内核的路径

- 进入/usr/src/ 解压文件

1. `sudo tar -xvf linux****`
2. `//linux***`代表你刚下载的 `linux` 内核的文件名,可以用 `tab` 自动补齐.

- 进入解压出来的内核文件中的 `kernel` 目录

```
cd linux***/kernel
```

- 修改 `sys.c` 文件

```
sudo gedit sys.c
```

```
//在最末尾加上你想加的新的系统调用
```

```
//举个例子, helloworld 程序
```

```
SYSCALL_DEFINE1(helloworld, int, number) {  
    printk("Hello, world!");  
    return number+1  
}
```

```
//对系统调用的编写 注意此处与前版本内核的不同
```

```
//上述程序中, 1 代表这个函数只有一个参数, 有两个参数就换成 2
```

```
//helloworld 是新增系统调用函数名
```

```
//int 是参数类型, number 是形参.
```

```
//函数功能就是打印 Hello, world!, 并且返回 1+number. 可以通过 dmesg 看到打印的句子.
```

- 增加系统调用号

```
//从 kernel 返回上一级目录
```

```
cd /arch/x86/entry/syscalls
```

```
sudo gedit syscall_64.tbl
```

```
//增加一个新的系统调用
```

```
//例如 335      64  helloworld      __x64_sys_helloworld
```

- 增加系统调用头文件

```
//返回/usr/src/linux***目录
```

```
cd include/linux
```

```
sudo gedit syscalls.h
```

```
//例如 asmlinkage long sys_helloworld(int number);
```

- 最后按照上述步骤编写你的文件 `copy` 和 `P、V` 操作系统调用程序吧

6. 编译内核

`sudo make mrproper` //删除之前编译的残余文件. 只在第一次编译时执行即可, 以免每次编译把重复文件重复编译.

`sudo make clean` //可执行可不执行, 因为上一个命令似乎是涵盖了 `clean` 的

`sudo make menuconfig`//在 `general setup` 哪里给你的内核换个名字, 比如 `AlexKernel`

`sudo make -j4` //根据你给虚拟机分配的核心数来适配. 我分了 4 核心.

`sudo make modules_install` // 安装内核模块

`sudo make install` //安装内核

7. 更换系统默认启动内核

更换启动内核链接

https://blog.csdn.net/cf_wu95/article/details/85984956

8. 重启, 编写你的测试 c 文件

PS:不嫌麻烦的可以直接看官方文档。新增系统调用官方文档

<https://www.kernel.org/doc/html/latest/process/adding-syscalls.html>

2.4.2 实验调试及心得

以下是我的测试 c 文件, 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, 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。

```
#include<stdio.h>
#include<unistd.h>
#include<errno.h>
#include<sys/syscall.h>
#include<linux/kernel.h>
int main(int argc, char *argv[]){
    long int a=syscall(335,5);
    printf("System call sys_mycall return %ld\n",a);
    return 0;
}
```

附录 实验代码

```
1. SYSCALL_DEFINE2(mcf, const char __user*, source, const char __user*, target){
2.     long ret = 0;
3.     char buffer[1000];
4.     // 转化为内核存储空间
5.     char * source_kd = strndup_user(source, PAGE_SIZE);
6.     char * target_kd = strndup_user(target, PAGE_SIZE);
7.     // 异常处理
8.     if (IS_ERR(source_kd)) {
9.         printk("source path error1");
10.        ret = PTR_ERR(source);
11.        goto error;
12.    }
13.    if (IS_ERR(target_kd)) {
14.        printk("target path error1");
15.        ret = PTR_ERR(target);
16.        goto error;
17.    }
18.    printk("source path : %s, target path : %s", source_kd, target_kd);
19.    mm_segment_t old_fs = get_fs();
20.    set_fs(KERNEL_DS);
21.    // 建立文件句柄
22.    struct file* source_fd = filp_open(source_kd, O_RDONLY, S_IRUSR);
23.    struct file* target_fd = filp_open(target_kd, O_WRONLY | O_CREAT, S_IRUS
        R|S_IWUSR);
24.    // 异常处理
25.    if (IS_ERR(source_fd)) {
26.        ret = PTR_ERR(source);
27.        printk("source file path error2\n");
28.        goto error;
29.    }
30.    if (IS_ERR(target_fd)) {
31.        ret = PTR_ERR(target);
32.        printk("target file path error2\n");
33.        goto error;
34.    }
35.    // 读取文件
36.    int read_num = 0;
37.    // vfs_read(source_fd, buffer, sizeof(buffer), &(source_fd->f_pos));
38.    // vfs_write(target_fd, buffer, sizeof(buffer), &(target_fd->f_pos))
    ;
39.    int count = 0;
```

```
40.     while ((read_num = vfs_read(source_fd,buffer, sizeof(buffer), &source_fd
    ->f_pos)) > 0) {
41.         vfs_write(target_fd, buffer, read_num, &target_fd->f_pos);
42.         count += 1;
43.         // 测试用,防止死循环
44.         if (count > 100) {
45.             goto error;
46.         }
47.     }
48.     set_fs(old_fs);
49.
50. error:
51.     return ret;
52. }
```

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

- Windows10
- VMware Workstation 15 Pro
- 虚拟机 Ubuntu19.10 (60G)
- 内核 Linux 5.4.21
- 核数：8
- 内存：4

3.3.2 实验设计

系统调用是操作系统内核和应用程序之间的接口，设备驱动程序是操作系统内核和机器硬件之间的接口。设备驱动程序为应用程序屏蔽了硬件的细节，这样在应用程序看来，硬件设备只是一个设备文件，应用程序可以象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，它完成以下的功能：

1. 对设备初始化和释放。
2. 把数据从内核传送到硬件和从硬件读取数据。
3. 读取应用程序传送给设备文件的数据和回送应用程序请求的数据。

检测和处理设备出现的错误。在 Linux 操作系统下有三类主要的设备文件类型，一是字符设备，二是块设备，三是网络设备。字符设备和块设备的主要区别是：在对字符设备发出读/写请求时，实际的硬件 I/O 一般就紧接着发生了，块设备则不然，它利用一块系统内存作缓冲区，当用户进程对设备请求能满足用户的要求，就返回请求的数据，如果不能，就调用请求函数来进行实际的 I/O 操作。块设备是主要针对磁盘等慢速设备设计的，以免耗费过多的 CPU 时间来等待。

(1) 编写一个简单程序，向内核中加入简单的内核，仅简单的输出一个语句，并且通过命令 `dmesg` 来查看是否有输出该语句，进而判断模块是否成功的插入。则主体的函数如下：

```
1. #include <linux/init.h>    //所有模块都会需要这个头文件
2. #include <linux/module.h> //下面的宏需要
3.
4. static int __init hello_init(void){
5.     printk(KERN_INFO "module init success\n");
6.     return 0;
7. }
8.
9. static void __exit hello_exit(void){
10.    printk(KERN_INFO "module exit success\n");
11. }
12.
13. module_init(hello_init);
14. module_exit(hello_exit);
15.
16. MODULE_LICENSE("GPL"); //开源协议
17. MODULE_AUTHOR("hyq");
18. MODULE_DESCRIPTION("功能描述");
```

(2) 因为插入的模块只能够使用内核提供的函数，所以这里输出语句的函数不是 `printf` 而是内核提供的输出函数 `printk`

(3) 在终端中输入 `make` 指令，执行 `makefile` 中的语句，`makefile` 文件内容如下所示：

```
obj-m := hello.o
PWD := $(shell pwd)
KVER := $(shell uname -r)
KDIR := /lib/modules/$(KVER)/build/

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

clean:
    rm -rf *.o *.mod.c *.mod.o *.ko *.synvers|*.order *.a
```

图 3-1 为模块插入的 `makefile` 文件

3.4 实验调试

3.4.1 实验步骤

1. 编写字符驱动程序, 功能是向虚拟出的虚拟设备传入一连串的数字字符.
(以下是概要代码, 详情请前往 GitHub 仓库阅览)

2. 字符设备驱动代码

```
1. #include <linux/modules.h>
2. #include <linux/version.h>
3. #include <linux/types.h>
4. #include <linux/fs.h>
5. #include <linux/mm.h>
6. #include <linux/errno.h>
7. #include <asm/segment.h>
8. //以上是我们需要的所有头文件
9. long int ALEX_MAJOR = 0;
10. static ssize_t read_alex(struct file *flip, char *buf, size_t count, loff_t*f_pos)
11. {
12.     int left;
13.     for(left = count ; left > 0 ; left--)
14.     {
15.         put_user(9,buf);
16.         buf++;
17.     }
18.     return count;
19. } //read 操作定义
20. static ssize_t write_alex(struct file *flip,const char *file,size_t count, l
    off_t*f_pos)
21. {
22.     return count;
23. } //write 操作的定义
24.
25. static int open_alex(struct inode *inode,struct file *file )
26. {
27.
28.     return 0;
29. } //open 操作的定义
30.
31. static int release_alex(struct inode *inode,struct file *file )
32. {
```



```

33.
34. }//release 操作的定义
35.
36. struct file_operations alex_fops={
37.     .read = read_alex,
38.     .write=write_alex,
39.     .open=open_alex,
40.     .release=release_alex
41. };//操作顺序定义
42.
43. static int init_mymodule(void)
44. {
45.     int result;
46.     result = register_chrdev(0, "alex", &alex_fops);
47.     if (result < 0) {
48.         printk(KERN_INFO "alex: can't get major number\n");
49.         return result;
50.     }
51.     if (ALEX_MAJOR == 0) ALEX_MAJOR = result; /* dynamic */
52.     return 0;
53. }//初始化驱动程序
54. static void cleanup_mymodule(void)
55. {
56.     unregister_chrdev(ALEX_MAJOR,"alex");
57. } //终止驱动程序
58.
59. MODULE_LICENSE("GPL");
60. module_init(init_mymodule);
61. module_exit(cleanup_mymodule);

```

3. 编译步骤

- a. 将上述程序复制到/usr/src/linux**/drivers/misc 下
- b. 在 Makefile 文件中加入一句
obj-m +=alex_driver.o
- c. 执行
make -C /usr/src/linux SUBDIR=\$PWD modules
得到.ko 文件
- d. 挂载
insmod ./alex_driver.ko
- e. 查看系统分配的设备号
cat /proc/devices
发现出现了一个设备叫 alex, 记下它前面的设备号, 我的是 241.
- f. 创建新的虚拟设备文件

```
mknod /dev/alex c 241 0
chmod 666 /dev/alex
```

执行结束后, 会在/dev 中看到一个新的设备文件 alex

4. 编写测试程序

```
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/alex",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

通过编写出来的主程序需要复杂的指令才能生成模块并且插入, 需要编写好 makefile 文件。通过查阅资料以及查看老师的指导文档, 最终尝试出正确的 makefile 文件的写法。

附录 实验代码

1. alex_drive.c

```
1. #include <linux/module.h>
2. #include <linux/version.h>
3. #include <linux/types.h>
4. #include <linux/fs.h>
5. #include <linux/mm.h>
6. #include <linux/errno.h>
7. #include <asm/segment.h>
8.
9.
10. long int ALEX_MAJOR = 0;
11. static ssize_t read_alex(struct file *flip, char *buf, size_t count, loff_t*f_
    pos)
12. {
13.     int left;
14.     for(left = count ; left > 0 ; left--)
15.     {
16.         put_user(9,buf);
17.         buf++;
18.     }
19.     return count;
20. }
21. static ssize_t write_alex(struct file *flip, const char *file, size_t count, l
    off_t*f_pos)
22. {
23.     return count;
24. }
25.
26. static int open_alex(struct inode *inode, struct file *file )
27. {
28.
29.     return 0;
30. }
31.
32. static int release_alex(struct inode *inode, struct file *file )
33. {
34.
35. }
36.
37. struct file_operations alex_fops={
38.     .read = read_alex,
```

```

39.     .write=write_alex,
40.     .open=open_alex,
41.     .release=release_alex
42. };
43.
44.
45.
46.
47. static int init_mymodule(void)
48. {
49.     int result;
50.     result = register_chrdev(0, "alex", &alex_fops);
51.     if (result < 0) {
52.         printk(KERN_INFO "alex: can't get major number\n");
53.         return result;
54.     }
55.     if (ALEX_MAJOR == 0) ALEX_MAJOR = result; /* dynamic */
56.     return 0;
57. }
58.
59.
60. static void cleanup_mymodule(void)
61. {
62.     unregister_chrdev(ALEX_MAJOR, "alex");
63. }
64.
65. MODULE_LICENSE("GPL");
66. module_init(init_mymodule);
67. module_exit(cleanup_mymodule);

```

2. alex_test.c (测试文件)

```

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/alex", 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. }
```

4 实验四 使用 QT 实现系统监控器

4.1 实验目的

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

- (1) 了解 /proc 文件的结构，内容，以及使用的方法
- (2) 能够从 /proc 文件当中获取所需要的系统信息
- (3) 用图形化界面将获取出来的系统内容进行集中的显示

4.3 实验设计

4.3.1 开发环境

- Windows10
- VMware Workstation 15 Pro
- 虚拟机 Ubuntu19.10 (60G)
- 内核 Linux 5.4.21
- 核数: 8
- 内存: 4

4.3.2 实验设计

同 1.1 中的题目其实十分相似，只是不用强制要求设计并发进程了，但是一定要制作图形界面，这里我们仍然选用 Qt5 来做。

- 1. 获取并显示主机名

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

4.4 实验调试

4.4.1 实验步骤

具体实现方法如下:

1. 构造函数:

```
MainWindow::MainWindow(QWidget *parent) : //构造函数, 初始化ui, 计时器
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    usage = 0;
    ui->cpu_rate->setText(QString::number(usage, 'f', 10));

    QTimer* timer = new QTimer(this);

    connect( timer, SIGNAL( timeout() ), this, SLOT( timer_update_currentTabInfo() ) );
    timer_update_currentTabInfo();
    showinfo(0);
    connect(timer, SIGNAL(timeout()), this, SLOT(Update()));
    timer->start(1000);
}
```

初始化 UI 界面, 并打开计时器用于统计运行时间

2. 刷新函数:

```

void MainWindow::Update()
{
    QProcess process;
    process.start("cat /proc/stat");
    process.waitForFinished();
    QString str = process.readLine();
    str.replace("\n", "");
    str.replace(QRegExp("( ){1,}"), " ");
    auto lst = str.split(" ");
    if(lst.size() > 3)
    {
        double use = lst[1].toDouble() + lst[2].toDouble() + lst[3].toDouble();
        double total = 0;
        for(int i = 1; i < lst.size(); ++i)
            total += lst[i].toDouble();
        if(total - m_cpu_total__ > 0)
        {
            usage = (use - m_cpu_used__) / (total - m_cpu_total__) * 100.0;
            ui->cpu_rate->setText(QString::number(usage, 'f', 10));
            // qDebug("cpu usage: %.2lf%%", (use - m_cpu_used__) / (total - m_cpu_total__) * 100.0);
            m_cpu_total__ = total;
            m_cpu_used__ = use;
        }
    }
}

```

用于隔指定的时间来重新获取系统信息文件。用于实时更新控制台上的信息。

3. 内存信息显示函数:

```

void MainWindow::timer_update_currentTabInfo(){
    QFile tempFile;
    QDateTime time;
    QString tempStr;
    int pos;
    ui->label_CurrentTime->setText(time.currentDateTime().toString("yyyy")+"."++"\
    time.currentDateTime().toString("M")+"."++"\
    time.currentDateTime().toString("d")+" "++"\
    time.currentDateTime().toString("h")+":"++"\
    time.currentDateTime().toString("m")+":"++"\
    time.currentDateTime().toString("s"));

    tempFile.close(); //关闭stat文件
    tempFile.setFileName("/proc/meminfo"); //打开内存信息文件
    if ( !tempFile.open(QIODevice::ReadOnly) )
    {
        QMessageBox::warning(this, tr("warning"), tr("The meminfo file can not open!"));
        return ;
    }

    QString memTotal;
    QString memFree;
    QString memUsed;
    int nMemTotal, nMemFree, nMemUsed;

```

用于实时刷新内存信息，并且实时更新内存的占用率。

4. 基本配置信息函数:


```

void MainWindow::showinfo(int index){
    QString tempStr; //读取文件信息字符串
    QFile tempFile; //用于打开系统文件
    int pos; //读取文件的位置
    if(index == 0){//SystemInfo
        //int ok;
        tempFile.setFileName("/proc/cpuinfo"); //打开CPU信息文件
        if ( !tempFile.open(QIODevice::ReadOnly) )
        {
            QMessageBox::warning(this, tr("warning"), tr("The cpuinfo file can not open"));
            return;
        }
        char hostname[30];
        gethostname(hostname,30);
        ui->label_hostname->setText(hostname);

        struct sysinfo info;
        time_t cur_time=0;
        time_t boot_time=0;
        struct tm *ptm=NULLptr;
        if(sysinfo(&info)) return;
        time(&cur_time);
        boot_time=cur_time-info.uptime;
        ptm=localtime(&boot_time);
        char boottime_buf[30];
        sprintf(boottime_buf,"%d.%d.%d %02d:%02d:%02d",ptm->tm_year+1900,ptm->tm_mon+1,ptm->tm_mday,ptm->tm_hour,ptm->tm_min,ptm->tm_sec);
        ui->label_boottime->setText(QString(boottime_buf));
    }
}

```

用于显示系统的基本配置信息，包括电脑的硬件信息。

5. 关机按钮

```

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

```

用于关闭计算机，本质上是调用了系统内置的关机函数。

最后效果图如下：



4.4.2 实验调试及心得

(1) 状态栏需要动态刷新，需要在 QT 界面动态的将信息获取并且在控件当中显示出来。

(2) 不同的编译内核文件存放相关信息的文件不一定相同，需要查询好自己当前内核的/proc 文件的存放位置以及信息存放的方式

(3) QT 与 javafx 类似，在 GUI 编程的思路类似

附录 实验代码

1.mainwindow.cpp

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3.
4. #include <QFile>
5. #include <QMessageBox>
6. #include <QDir>
7. #include <QListWidget>
8. #include <QListWidgetItem>
9. #include <QStringList>
10. #include <QTimer>
11. #include <QDateTime>
12. #include <QProcess>
13. #include <QPainter>
14. #include <QStringList>
15. #include <QTableWidget>
16.
17. MainWindow::MainWindow(QWidget *parent) : //构造函数，初始化 ui，计时器
18.     QMainWindow(parent),
19.     ui(new Ui::MainWindow)
20. {
21.     ui->setupUi(this);
22.     usage = 0;
23.     ui->cpu_rate->setText(QString::number(usage, 'f', 10));
24.
25.     QTimer* timer=new QTimer(this);
26.
27.     connect( timer, SIGNAL( timeout() ), this, SLOT( timer_update_currentTab
        Info() ) );
28.     timer_update_currentTabInfo();
29.     showinfo(0);
```

```

30.     connect(timer,SIGNAL(timeout()),this,SLOT(Update()));
31.     timer->start(1000);
32. }
33.
34.
35.
36. void MainWindow::Update()
37. {
38.     QProcess process;
39.     process.start("cat /proc/stat");
40.     process.waitForFinished();
41.     QString str = process.readLine();
42.     str.replace("\n","");
43.     str.replace(QRegExp("( ){1,}")," ");
44.     auto lst = str.split(" ");
45.     if(lst.size() > 3)
46.     {
47.         double use = lst[1].toDouble() + lst[2].toDouble() + lst[3].toDouble
            ();
48.         double total = 0;
49.         for(int i = 1;i < lst.size();++i)
50.             total += lst[i].toDouble();
51.         if(total - m_cpu_total__ > 0)
52.         {
53.             usage = (use - m_cpu_used__) / (total - m_cpu_total__) * 100.0;
54.
55.             ui->cpu_rate->setText(QString::number(usage,'f',10));
56.             //qDebug("cpu usage:%.2lf%%",(use - m_cpu_use__) / (total - m_cp
            u_total__) * 100.0);
57.             m_cpu_total__ = total;
58.             m_cpu_used__ = use;
59.         }
60. }
61. void MainWindow::timer_update_currentTabInfo(){
62.     QFile tempFile;
63.     QDateTime time;
64.     QString tempStr;
65.     int pos;
66.     ui->label_CurrentTime->setText(time.currentDateTime().toString("yyyy")+
        ". "+\
67.                                     time.currentDateTime().toString("M")+". "+
        \

```

```

68.         time.currentDateTime().toString("d")+ " "+
        \
69.         time.currentDateTime().toString("h")+":"+
        \
70.         time.currentDateTime().toString("m")+":"+
        \
71.         time.currentDateTime().toString("s"));
72.
73.
74.     tempFile.close(); //关闭 stat 文件
75.     tempFile.setFileName("/proc/meminfo"); //打开内存信息文件
76.     if ( !tempFile.open(QIODevice::ReadOnly) )
77.     {
78.         QMessageBox::warning(this, tr("warning"), tr("The meminfo file can not open!"), QMessageBox::Yes);
79.         return ;
80.     }
81.     QString memTotal;
82.     QString memFree;
83.     QString memUsed;
84.     int nMemTotal, nMemFree, nMemUsed;
85.
86.     while (1)
87.     {
88.         tempStr = tempFile.readLine();
89.         pos = tempStr.indexOf("MemTotal");
90.         if (pos != -1)
91.         {
92.             memTotal = tempStr.mid(pos+10, tempStr.length()-13);
93.             memTotal = memTotal.trimmed();
94.             nMemTotal = memTotal.toInt()/1024;
95.         }
96.         else if (pos = tempStr.indexOf("MemFree"), pos != -1)
97.         {
98.             memFree = tempStr.mid(pos+9, tempStr.length()-12);
99.             memFree = memFree.trimmed();
100.            nMemFree = memFree.toInt()/1024;
101.            break;
102.        }
103.    }
104.
105.    nMemUsed = nMemTotal - nMemFree;
106.
107.    memUsed = QString::number(nMemUsed, 10);

```

```

108.     memFree = QString::number(nMemFree, 10);
109.     memTotal = QString::number(nMemTotal, 10);
110.
111.     ui->progressBar_RAM->setValue(nMemUsed*100/nMemTotal);
112.
113.     int index=ui->tabWidget_INFO->currentIndex();
114.     if(index==0){
115.         struct sysinfo info;
116.         sysinfo(&info);
117.         struct tm *ptm=NULLPTR;
118.         ptm=gmtime(&info.uptime);
119.         char time_buf[30];
120.         sprintf(time_buf, "%02d:%02d:%02d", ptm->tm_hour, ptm->tm_min, ptm->tm_
            sec);
121.         ui->label_runningtime->setText(QString(time_buf));
122.     }
123.
124.     tempFile.close(); //关闭内存信息文件
125. }
126.
127.
128.
129. void MainWindow::showinfo(int index){
130.     QString tempStr; //读取文件信息字符串
131.     QFile tempFile; //用于打开系统文件
132.     int pos; //读取文件的位置
133.     if(index ==0){ //SystemInfo
134.         //int ok;
135.         tempFile.setFileName("/proc/cpuinfo"); //打开 CPU 信息文件
136.         if ( !tempFile.open(QIODevice::ReadOnly) )
137.         {
138.             QMessageBox::warning(this, tr("warning"), tr("The cpuinfo file
                can not open!"), QMessageBox::Yes);
139.             return;
140.         }
141.         char hostname[30];
142.         gethostname(hostname,30);
143.         ui->label_hostname->setText(hostname);
144.
145.         struct sysinfo info;
146.         time_t cur_time=0;
147.         time_t boot_time=0;
148.         struct tm *ptm=NULLPTR;
149.         if(sysinfo(&info)) return;

```

```

150.         time(&cur_time);
151.         boot_time=cur_time-info.uptime;
152.         ptm=localtime(&boot_time);
153.         char boottime_buf[30];
154.         sprintf(boottime_buf,"%d.%d.%d %02d:%02d:%02d",ptm->tm_year+1900,pt
            m->tm_mon+1,ptm->tm_mday,
155.                 ptm->tm_hour,ptm->tm_min,ptm->tm_sec);
156.         ui->label_boottime->setText(QString(boottime_buf));
157.
158.
159.         //循环读取文件内容, 查找需要的信息
160.         while (1)
161.         {
162.             tempStr = tempFile.readLine();
163.             if (pos = tempStr.indexOf("model name"), pos != -1)
164.             {
165.                 pos += 12;
166.                 QString *cpu_type = new QString( tempStr.mid(pos, tempStr.l
                    ength()-12) );
167.                 ui->label_CPUType->setText(*cpu_type);
168.                 break;
169.             }
170.         }
171.         tempFile.close();
172.
173.
174.         tempFile.setFileName("/proc/version");
175.         if ( !tempFile.open(QIODevice::ReadOnly) )
176.         {
177.             QMessageBox::warning(this, tr("warning"), tr("The version file
                can not open!"), QMessageBox::Yes);
178.             return ;
179.         }
180.         tempStr = tempFile.readLine();
181.
182.         int pos1 = tempStr.indexOf("(");
183.         QString *os_type = new QString( tempStr.mid(14, pos1-pos-2) );
184.         ui->label_SystemVersion->setText(*os_type);
185.
186.         tempFile.close(); //关闭操作系统信息文件
187.     }
188.
189.     else if(index==1){//ProgInfo
190.         ui->listWidget_process->clear();

```

```

191.         QDir qd("/proc");
192.         QStringList qsList = qd.entryList();
193.         QString qs = qsList.join("\n");
194.         QString id_of_pro;
195.         bool ok;
196.         int find_start = 3;
197.         int a, b;
198.         int nProPid; //进程 PID
199.         QString proName; //进程名
200.         QString proState; //PPID
201.         QString proPri; //进程优先级
202.         QString proMem; //进程占用内存
203.         QListWidgetItem *title = new QListWidgetItem("PID\t" + QString::from
            mUtf8("名称") + "\t\t" +
204.                                                     QString::fromUtf8("PPI
            D") + "\t" +
205.                                                     QString::fromUtf8("优
            优先级") + "\t" +
206.                                                     QString::fromUtf8("占
            用内存"), ui->listWidget_process);
207.         //循环读取进程
208.         while (1)
209.         {
210.             //获取进程 PID
211.             a = qs.indexOf("\n", find_start);
212.             b = qs.indexOf("\n", a+1);
213.             find_start = b;
214.             id_of_pro = qs.mid(a+1, b-a-1);
215.             nProPid = id_of_pro.toInt(&ok, 10);
216.             if(!ok)
217.             {
218.                 break;
219.             }
220.
221.             //打开 PID 所对应的进程状态文件
222.             tempFile.setFileName("/proc/" + id_of_pro + "/stat");
223.             if ( !tempFile.open(QIODevice::ReadOnly) )
224.             {
225.                 QMessageBox::warning(this, tr("warning"), tr("The pid stat
                    file can not open!"), QMessageBox::Yes);
226.                 return;
227.             }
228.             tempStr = tempFile.readLine();
229.             if (tempStr.length() == 0)

```



```

230.         {
231.             break;
232.         }
233.         a = tempStr.indexOf("(");
234.         b = tempStr.indexOf(")");
235.         proName = tempStr.mid(a+1, b-a-1);
236.         proName.trimmed(); //删除两端的空格
237.         proState = tempStr.section(" ", 3, 3);
238.         proPri = tempStr.section(" ", 17, 17);
239.         proMem = tempStr.section(" ", 22, 22);
240.
241.
242.         if (proName.length() >= 13)
243.         {
244.             QListWidgetItem *item = new QListWidgetItem(id_of_pro + "\t
" +
245.                                                         proName + "\t"
246.                                                         +
247.                                                         proState + "\t"
248.                                                         +
249.                                                         proPri + "\t" +
250.                                                         proMem, ui->lis
tWidget_process);
251.         }
252.         else
253.         {
254.             QListWidgetItem *item = new QListWidgetItem(id_of_pro + "\t
" +
255.                                                         proName + "\t\t
" +
256.                                                         proState + "\t"
257.                                                         +
258.                                                         proPri + "\t" +
259.                                                         proMem, ui->lis
tWidget_process);
260.         }
261.         tempFile.close();
262.     }
263.     tempFile.close(); //关闭该 PID 进程的状态文件

```

```

264.     }
265. }
266.
267. void MainWindow::on_pushButton_shutdown_clicked()
268. {
269.     system("shutdown -h now");
270. }
271.
272. void MainWindow::on_tabWidget_INFO_currentChanged(int index)
273. {
274.     showinfo(index); //显示 tab 中的内容
275.     return ;
276. }
277.
278.
279. MainWindow::~MainWindow()
280. {
281.     delete ui;
282. }

```

2.mainwindow.h

```

1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3.
4. #include <QMainWindow>
5. #include <unistd.h>
6. #include <QTimer>
7. #include <sys/sysinfo.h>
8. #include <QObject>
9. #include <QProcess>
10. #include <QDebug>
11. #include <QWidget>
12. #include <QList>
13.
14. namespace Ui {
15.     class MainWindow;
16. }
17.
18. class MainWindow : public QMainWindow
19. {
20.     Q_OBJECT
21.
22. public:
23.     explicit MainWindow(QWidget *parent = nullptr);

```

```

24.     ~MainWindow();
25.
26. private:
27.     Ui::MainWindow *ui;
28.     QList<float> yList;
29.     QList<float> yList1;
30.
31. private slots:
32.     void showinfo(int index);
33.     void on_pushButton_shutdown_clicked();
34.     void on_tabWidget_INFO_currentChanged(int index);
35.     void timer_update_currentTabInfo();
36. public slots:
37.     void Update();
38. private:
39.     double usage;
40.     double m_cpu_total__ = 0;
41.     double m_cpu_used__ = 0;
42. };
43.
44. #endif // MAINWINDOW_H

```

3.mainwindow.ui

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <ui version="4.0">
3.   <class>MainWindow</class>
4.   <widget class="QMainWindow" name="MainWindow">
5.     <property name="geometry">
6.       <rect>
7.         <x>0</x>
8.         <y>0</y>
9.         <width>403</width>
10.        <height>540</height>
11.      </rect>
12.    </property>
13.    <property name="windowTitle">
14.      <string> Alex PC Monitor</string>
15.    </property>
16.    <widget class="QWidget" name="centralWidget">
17.      <widget class="QTabWidget" name="tabWidget_INFO">
18.        <property name="geometry">
19.          <rect>

```

```

20.     <x>10</x>
21.     <y>0</y>
22.     <width>381</width>
23.     <height>291</height>
24. </rect>
25. </property>
26. <property name="currentIndex">
27.     <number>0</number>
28. </property>
29. <widget class="QWidget" name="tab">
30.     <attribute name="title">
31.         <string>系统信息</string>
32.     </attribute>
33.     <widget class="QFrame" name="frame_2">
34.         <property name="geometry">
35.             <rect>
36.                 <x>10</x>
37.                 <y>130</y>
38.                 <width>361</width>
39.                 <height>121</height>
40.             </rect>
41.         </property>
42.         <property name="frameShape">
43.             <enum>QFrame::StyledPanel</enum>
44.         </property>
45.         <property name="frameShadow">
46.             <enum>QFrame::Raised</enum>
47.         </property>
48.         <widget class="QLabel" name="label_7">
49.             <property name="geometry">
50.                 <rect>
51.                     <x>20</x>
52.                     <y>30</y>
53.                     <width>111</width>
54.                     <height>21</height>
55.                 </rect>
56.             </property>
57.             <property name="text">
58.                 <string>系统版本号: </string>
59.             </property>
60.         </widget>
61.         <widget class="QLabel" name="label_8">
62.             <property name="geometry">
63.                 <rect>

```

```

64.         <x>20</x>
65.         <y>81</y>
66.         <width>91</width>
67.         <height>21</height>
68.     </rect>
69. </property>
70. <property name="text">
71.     <string>CPU 型号: </string>
72. </property>
73. </widget>
74. <widget class="QLabel" name="label_SystemVersion">
75.     <property name="geometry">
76.         <rect>
77.             <x>130</x>
78.             <y>20</y>
79.             <width>341</width>
80.             <height>41</height>
81.         </rect>
82.     </property>
83.     <property name="text">
84.         <string>TextLabel</string>
85.     </property>
86. </widget>
87. <widget class="QLabel" name="label_CPUType">
88.     <property name="geometry">
89.         <rect>
90.             <x>90</x>
91.             <y>79</y>
92.             <width>341</width>
93.             <height>41</height>
94.         </rect>
95.     </property>
96.     <property name="font">
97.         <font>
98.             <pointsize>9</pointsize>
99.         </font>
100.    </property>
101.    <property name="text">
102.        <string>TextLabel</string>
103.    </property>
104. </widget>
105. </widget>
106. <widget class="QFrame" name="frame_5">
107.     <property name="geometry">

```

```

108.      <rect>
109.          <x>10</x>
110.          <y>20</y>
111.          <width>361</width>
112.          <height>111</height>
113.      </rect>
114.  </property>
115.  <property name="frameShape">
116.      <enum>QFrame::StyledPanel</enum>
117.  </property>
118.  <property name="frameShadow">
119.      <enum>QFrame::Raised</enum>
120.  </property>
121.  <widget class="QLabel" name="label_4">
122.      <property name="geometry">
123.          <rect>
124.              <x>20</x>
125.              <y>40</y>
126.              <width>81</width>
127.              <height>21</height>
128.          </rect>
129.      </property>
130.      <property name="text">
131.          <string>启动时间: </string>
132.      </property>
133.  </widget>
134.  <widget class="QLabel" name="label_5">
135.      <property name="geometry">
136.          <rect>
137.              <x>20</x>
138.              <y>70</y>
139.              <width>101</width>
140.              <height>21</height>
141.          </rect>
142.      </property>
143.      <property name="text">
144.          <string>运行时长: </string>
145.      </property>
146.  </widget>
147.  <widget class="QLabel" name="label_6">
148.      <property name="geometry">
149.          <rect>
150.              <x>20</x>
151.              <y>10</y>

```

```

152.         <width>71</width>
153.         <height>21</height>
154.     </rect>
155. </property>
156. <property name="text">
157.     <string>主机名: </string>
158. </property>
159. </widget>
160. <widget class="QLabel" name="label_hostname">
161.     <property name="geometry">
162.         <rect>
163.             <x>130</x>
164.             <y>10</y>
165.             <width>341</width>
166.             <height>21</height>
167.         </rect>
168.     </property>
169.     <property name="text">
170.         <string>AlexFanLinux</string>
171.     </property>
172. </widget>
173. <widget class="QLabel" name="label_boottime">
174.     <property name="geometry">
175.         <rect>
176.             <x>130</x>
177.             <y>40</y>
178.             <width>341</width>
179.             <height>21</height>
180.         </rect>
181.     </property>
182.     <property name="text">
183.         <string>2020.3.1 18:30:10</string>
184.     </property>
185. </widget>
186. <widget class="QLabel" name="label_runningtime">
187.     <property name="geometry">
188.         <rect>
189.             <x>130</x>
190.             <y>70</y>
191.             <width>341</width>
192.             <height>21</height>
193.         </rect>
194.     </property>
195.     <property name="text">

```

```

196.         <string>00: 00</string>
197.     </property>
198. </widget>
199. </widget>
200. </widget>
201. <widget class="QWidget" name="tab_1">
202.     <attribute name="title">
203.         <string>进程信息</string>
204.     </attribute>
205.     <widget class="QListWidget" name="listWidget_process">
206.         <property name="geometry">
207.             <rect>
208.                 <x>10</x>
209.                 <y>20</y>
210.                 <width>391</width>
211.                 <height>281</height>
212.             </rect>
213.         </property>
214.     </widget>
215. </widget>
216. </widget>
217. <widget class="QLabel" name="label_9">
218.     <property name="geometry">
219.         <rect>
220.             <x>20</x>
221.             <y>450</y>
222.             <width>91</width>
223.             <height>21</height>
224.         </rect>
225.     </property>
226.     <property name="text">
227.         <string>当前时间: </string>
228.     </property>
229. </widget>
230. <widget class="QLabel" name="label_10">
231.     <property name="geometry">
232.         <rect>
233.             <x>20</x>
234.             <y>315</y>
235.             <width>91</width>
236.             <height>16</height>
237.         </rect>
238.     </property>
239.     <property name="text">

```



```

240.     <string>CPU 使用率: </string>
241.     </property>
242. </widget>
243. <widget class="QLabel" name="label_11">
244.     <property name="geometry">
245.         <rect>
246.             <x>20</x>
247.             <y>360</y>
248.             <width>91</width>
249.             <height>16</height>
250.         </rect>
251.     </property>
252.     <property name="text">
253.         <string>内存使用率: </string>
254.     </property>
255. </widget>
256. <widget class="QPushButton" name="pushButton_shutdown">
257.     <property name="geometry">
258.         <rect>
259.             <x>269</x>
260.             <y>420</y>
261.             <width>81</width>
262.             <height>81</height>
263.         </rect>
264.     </property>
265.     <property name="font">
266.         <font>
267.             <pointsize>19</pointsize>
268.         </font>
269.     </property>
270.     <property name="text">
271.         <string>关    机</string>
272.     </property>
273. </widget>
274. <widget class="QLabel" name="label_CurrentTime">
275.     <property name="geometry">
276.         <rect>
277.             <x>90</x>
278.             <y>450</y>
279.             <width>341</width>
280.             <height>20</height>
281.         </rect>
282.     </property>
283.     <property name="text">

```

```

284.     <string>Time</string>
285.     </property>
286. </widget>
287. <widget class="QProgressBar" name="progressBar_RAM">
288.     <property name="geometry">
289.         <rect>
290.             <x>140</x>
291.             <y>360</y>
292.             <width>211</width>
293.             <height>21</height>
294.         </rect>
295.     </property>
296.     <property name="value">
297.         <number>24</number>
298.     </property>
299. </widget>
300. <widget class="QLabel" name="cpu_rate">
301.     <property name="geometry">
302.         <rect>
303.             <x>140</x>
304.             <y>315</y>
305.             <width>191</width>
306.             <height>21</height>
307.         </rect>
308.     </property>
309.     <property name="text">
310.         <string>10</string>
311.     </property>
312. </widget>
313. <widget class="QLabel" name="label">
314.     <property name="geometry">
315.         <rect>
316.             <x>240</x>
317.             <y>310</y>
318.             <width>31</width>
319.             <height>31</height>
320.         </rect>
321.     </property>
322.     <property name="text">
323.         <string>%</string>
324.     </property>
325.     <property name="textFormat">
326.         <enum>Qt::PlainText</enum>
327.     </property>

```

```
328.     </widget>
329.     <zorder>label_9</zorder>
330.     <zorder>label_10</zorder>
331.     <zorder>label_11</zorder>
332.     <zorder>pushButton_shutdown</zorder>
333.     <zorder>label_CurrentTime</zorder>
334.     <zorder>progressBar_RAM</zorder>
335.     <zorder>tabWidget_INFO</zorder>
336.     <zorder>cpu_rate</zorder>
337.     <zorder>label</zorder>
338. </widget>
339. </widget>
340. <layoutdefault spacing="6" margin="11"/>
341. <resources/>
342. <connections/>
343. </ui>
```

5 实验五 小型模拟文件系统

5.1 实验目的

设计一个小型模拟文件系统，可以模拟用户登陆、退出，以及一些基本的文件操作。

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。

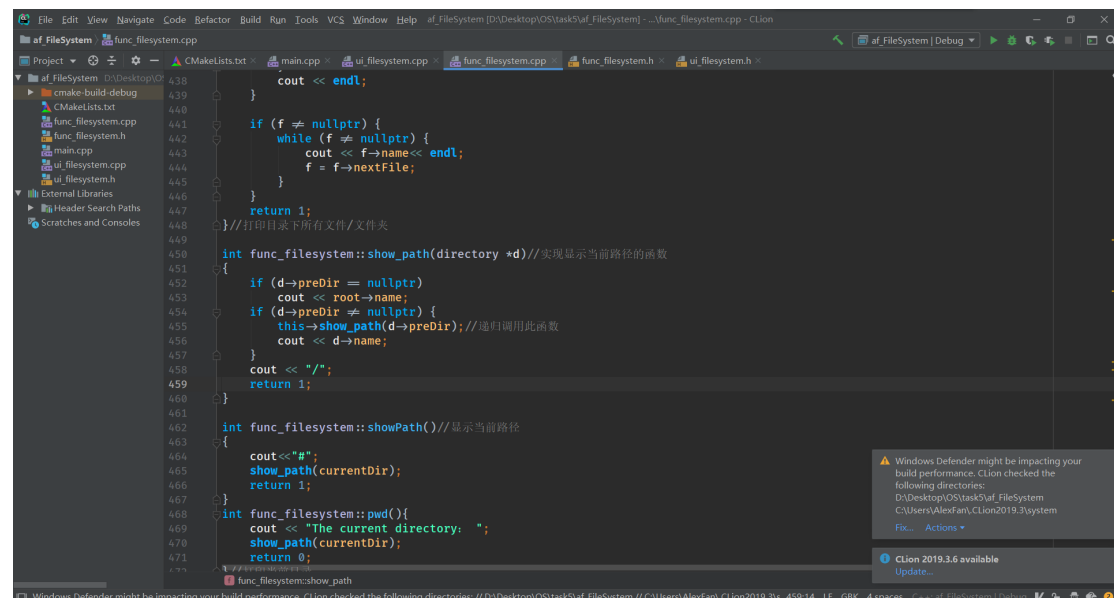
5.3 实验设计

5.3.1 开发环境

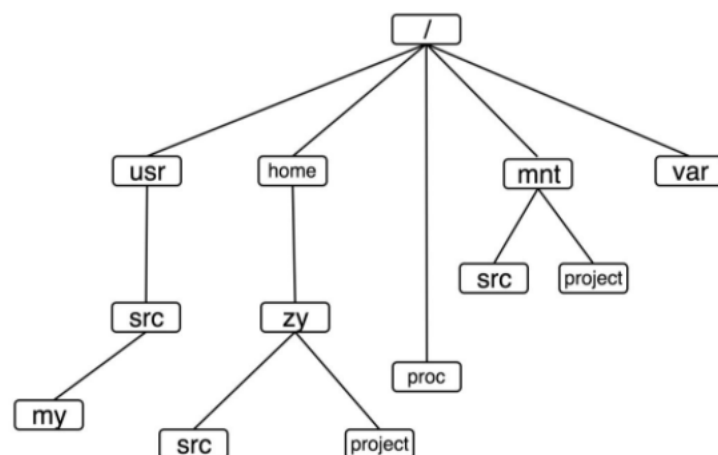
- Windows10
- VMware Workstation 15 Pro
- 虚拟机 Ubuntu19.10 (60G)
- 内核 Linux 5.4.21
- 核数: 8
- 内存: 4

5.3.2 实验设计

工程界面图



在本地机器上请求一块固定大小的磁盘区域，实现如下的文件树，以及多用户。



5.4 实验调试

5.4.1 实验步骤

因为是基于 windows 平台来进行编写的，所以没有使用到 Linux 的系统调用，但是命令的格式都是相同的。主要实现了以下功能。

1. 登陆
2. 注册
3. 帮助菜单
4. 登出

命令清单：

1. 新建文件 —— touch
2. 删除文件 —— rm
3. 复制文件 —— cp
4. 粘贴文件 —— ps
5. 打开文件 —— open
6. 编辑文件 —— vi
7. 新建目录 —— mkdir
8. 删除目录 —— rmdir
9. 展开目录 —— ls
10. 进入目录 —— cd + [目录名]
11. 返回上级 —— cd ..
12. 清屏 —— clear
13. 注销 —— exit

```
D:\Desktop\OS\task5\af_FileSystem\cmake-build-debug\af_FileSystem

Alex Fan File System
1. sign up
2. login
3. help
4. sign out

请选择(请输入数字编号): 3

-----Alex Fan File System 操作手册-----
1.touch + <文件名>      新建文件
2.rm + <文件名>         删除文件
3.cp + <文件名>         复制文件
4.ps                    粘贴文件
5.open + <文件名>       打开文件
6.vi + <文件名>         编辑文件
7.mkdir + <文件名>      新建目录
8.rmdir + <文件名>      删除目录
9.ls                    展开目录
10.cd + <文件名>        进入下级目录
11.cd ..                返回上级目录
12.clear                清屏
13.exit                注销

请按任意键继续...
```

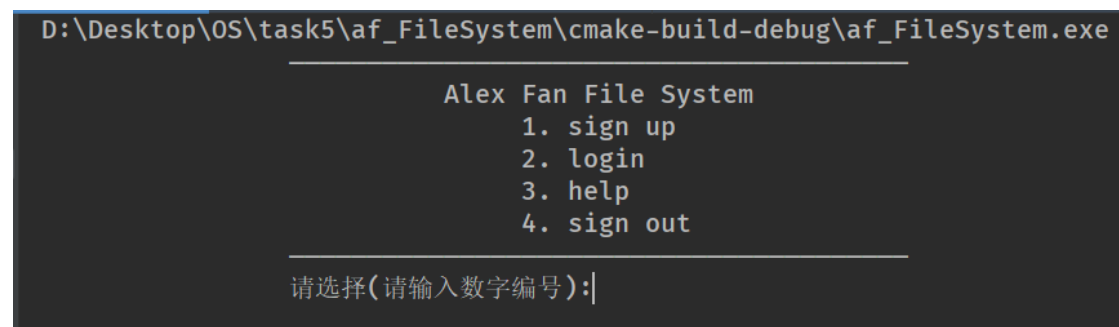
5.4.2 实验调试及心得

1. 入口主函数设计

```
#include "ui_filesystem.h"

int main() {
    start();
    return 0;
}
```

效果图:



2. 注册功能

通过用户输入将用户名和密码写入到 user.ini 文件中。并保存起来。

```
int regist(char username[30], char password[30]) {

    FILE *cfptr; // 文件指针
    if ((cfptr = fopen("user.ini", "a+")) == nullptr) {
        printf("File client.txt could not be opened\n");
        fclose(cfptr);
        return 0;
    } else {
        fprintf(cfptr, "%s %s\n", username, password);
        fclose(cfptr);
        return 1;
    }
}

/* 用户名字密码检验, 是否已经注册 (与文件内数据比较) */
```

效果图如下, 我们注册一个账号名为 alexfan, 密码 123456alex。
提示注册成功...

```
D:\Desktop\OS\task5\af_FileSystem\cmake-build-debug\af_FileSystem.exe

          Alex Fan File System
          1. sign up
          2. login
          3. help
          4. sign out

          请选择(请输入数字编号): 1

1
请输入用户名: alexfan
alexfan
请输入密码: 123456alex
123456alex
请确认密码: 123456alex
123456alex
注册成功..
```

3. 登陆功能

登陆功能的实现就主要是通过读取 user.ini 文件来进行的。逐一比对每条信息就行。

```
int login(char username[30], char password[30]) {
    char user[30];
    char pass[30];
    FILE *cfp[30]; // 文件指针
    if ((cfp[30] = fopen("user.ini", "r")) == nullptr) {
        printf("File client.txt could not be opened\n");
        fclose(cfp[30]);
        return 0;
    } else {
        while (!feof(cfp[30])) {
            fscanf(cfp[30], "%s%s", user, pass);
            if ((strcmp(username, user) == 0) && (strcmp(password, pass) == 0)) {
                fclose(cfp[30]);
                return 1;
            }
        }
    }
    fclose(cfp[30]);
    return 0;
} // 登录
```

用户名: alexfan

密码: 123456alex

登陆成功....

```
          请选择(请输入数字编号): 2

2
请输入用户名: alexfan
alexfan
请输入密码: 123456alex
123456alex
登录成功
```


4.HELP 菜单

这个实现起来就很简单啦，只需要把功能清单逐一 print 出来就行。

```
void help() {
    cout << "-----" << endl;
    cout << "-----Alex Fan File System 操作手册-----" << endl;
    cout << " 1.touch + <文件名>          新建文件 " << endl;
    cout << " 2.rm + <文件名>           删除文件 " << endl;
    cout << " 3.cp + <文件名>          复制文件 " << endl;
    cout << " 4.ps                     粘贴文件 " << endl;
    cout << " 5.open + <文件名>        打开文件 " << endl;
    cout << " 6.vi + <文件名>         编辑文件 " << endl;
    cout << " 7.mkdir + <文件名>       新建目录 " << endl;
    cout << " 8.rmdir + <文件名>      删除目录 " << endl;
    cout << " 9.ls                    展开目录 " << endl;
    cout << " 10.cd + <文件名>        进入下级目录" << endl;
    cout << " 11.cd ..               返回上级目录" << endl;
    cout << " 12.clear              清屏 " << endl;
    cout << " 13.exit              注销 " << endl;
    cout << "-----" << endl;
    << endl;
} /*用户注册写入文件函数*/
```

实现效果图如下:

```
D:\Desktop\OS\task5\af_FileSystem\cmake-build-debug\af_FileSystem>
                                     Alex Fan File System
                                     1. sign up
                                     2. login
                                     3. help
                                     4. sign out
                                     -----
                                     请选择(请输入数字编号):3
-----
-----Alex Fan File System 操作手册-----
1.touch + <文件名>          新建文件
2.rm + <文件名>           删除文件
3.cp + <文件名>          复制文件
4.ps                     粘贴文件
5.open + <文件名>        打开文件
6.vi + <文件名>         编辑文件
7.mkdir + <文件名>       新建目录
8.rmdir + <文件名>      删除目录
9.ls                    展开目录
10.cd + <文件名>        进入下级目录
11.cd ..               返回上级目录
12.clear              清屏
13.exit              注销
-----
--
请按任意键继续. . .
```

5.命令行工具如下:

首先需要通过 operate 函数来进行选定所需要执行的命令。

```
int operate(char name[30], char pass[30]) {
    func_filesystem af_filesystem;
    af_filesystem.setUser(name, pass);
    while (1) {
        system("CLS");
        while (1) {
            cout << endl;
            string choice;
            af_filesystem.showPath();
            cin >> choice;
            if (choice == "mkdir")
                af_filesystem.newDir();
            else if (choice == "touch")
                af_filesystem.newFile();
            else if (choice == "rmdir")
                af_filesystem.deleteDir();
            else if (choice == "rm")
                af_filesystem.deleteFile();
            else if (choice == "cd")
                af_filesystem.open_dir();
            else if (choice == "open")
                af_filesystem.open_file();
            else if (choice == "ls")
                af_filesystem.ls();
            else if (choice == "cp")
                af_filesystem.copyFile();
        }
    }
}
```

然后逐一进行如下的命令测试:

```
#alexfan/touch readme.md
touch readme.md
CREATE -OK

#alexfan/ls
ls
readme.md

#alexfan/

mkdir alex
mkdir alex
CREATE -OK

#alexfan/ls
ls
alex

readme.md

#alexfan/cd alex
cd alex

#alexfan/alex/cd ..
```

直接回退到未登陆状态

```
#alexfan/alex/exit
exit
```

Alex Fan File System

1. sign up
2. login
3. help
4. sign out

请选择(请输入数字编号):

附录 实验代码

1.ui_filesystem.cpp

```
1. //  
2. // Created by AlexFan on 2020/3/2.  
3. //  
4. #include<iostream>  
5. #include<string>  
6. #include"func_filesystem.h"  
7. #include"ui_filesystem.h"  
8. #include<Windows.h>  
9. using namespace std;  
10.  
11. void start(){  
12.     int choice = 0;  
13.     char name[30], pass[30], pass1[30];  
14.     system("CLS");  
15.     Sleep(2);  
16.     while (true) {  
17.         system("CLS");  
18.         cout << "\t\t_____ \t" << endl;  
19.         cout << "\t\t        Alex Fan File System          \t" << endl;  
20.         cout << "\t\t        1. sign up                \t" << endl;  
21.         cout << "\t\t        2. login                  \t" << endl;  
22.         cout << "\t\t        3. help                    \t" << endl;  
23.         cout << "\t\t        4. sign out                 \t" << endl;  
  
24.         cout << "\t\t_____ \t" << endl;  
25.         cout << "\t\t 请选择(请输入数字编号): " ;  
26.         cin >> choice;  
27.         switch (choice) {
```

```

28.      /*选择注册*/
29.      case 1: {
30.          cout << "请输入用户名:";
31.          cin >> name;
32.          cout << "请输入密码: ";
33.          cin >> pass;
34.          cout << "请确认密码: ";
35.          cin >> pass1;
36.          while (strcmp(pass, pass1) != 0) {
37.              cout << "密码不一致, 请重试" << endl;
38.              cout << "请输入密码: ";
39.              cin >> pass;
40.              cout << "请确认密码: ";
41.              cin >> pass1;
42.          }
43.          if (regist(name, pass) == 1){
44.              cout << "注册成功.." << endl;
45.              Sleep(2);
46.          }
47.          else{
48.              cout << "注册失败" << endl;
49.              Sleep(2);
50.          }
51.      }
52.      break;
53.
54.      /*选择登录*/
55.      case 2: {
56.          cout << "请输入用户名:";
57.          cin >> name;
58.          cout << "请输入密码:";
59.          cin >> pass;
60.          if (login(name, pass) == 1) {
61.              cout << "登录成功" << endl;
62.              operate(name, pass);
63.
64.          } else{
65.              cout << "登录失败, 请检查用户名和密码" << endl;
66.              Sleep(2);
67.          }}
68.      break;
69.      case 3:{
70.          system("CLS");
71.          help();

```

```

72.         system("Pause");
73.     }break;
74.
75.     case 4: {
76.         system("CLS");
77.         cout << "退出文件系统" << endl;
78.         Sleep(3);
79.         exit(0);
80.     }
81.     break;
82.     /*其他选项*/
83.     default:
84.         cout << "命令不存在, 请重新选择"<< endl;
85.
86.         break;
87.     }
88. }
89.
90. }//开始文件系统
91. void help() {
92.     cout << "-----" << endl;
93.     cout << "-----Alex Fan File System 操作手册
        -----" << endl;
94.     cout << " 1.touch + <文件名>                新建文
        件 " << endl;
95.     cout << " 2.rm + <文件名>                删除文
        件 " << endl;
96.     cout << " 3.cp + <文件名>                复制文
        件 " << endl;
97.     cout << " 4.ps                粘贴文
        件 " << endl;
98.     cout << " 5.open + <文件名>            打开文
        件 " << endl;
99.     cout << " 6.vi + <文件名>            编辑文
        件 " << endl;
100.    cout << " 7.mkdir + <文件名>          新建目
        录 " << endl;
101.    cout << " 8.rmdir + <文件名>          删除目
        录 " << endl;
102.    cout << " 9.ls                展开目
        录 " << endl;
103.    cout << " 10.cd + <文件名>            进入下级目录
        " << endl;

```

```

104.     cout << " 11.cd.."                                返回上级目录
        " << endl;
105.     cout << " 12.clear                                清
        屏        " << endl;
106.     cout << " 13.exit                                注
        销        " << endl;
107.     cout << "_____ " <<
        endl
108.         << endl;
109.
110. }/*用户注册写入文件函数*/
111. int regist(char username[30], char password[30]) {
112.
113.     FILE *cfptr;//文件指针
114.     if ((cfptr = fopen("user.ini", "a+")) == nullptr) {
115.         printf("File client.txt could not be opened\n");
116.         fclose(cfptr);
117.         return 0;
118.     } else {
119.         fprintf(cfptr, "%s %s\n", username, password);
120.         fclose(cfptr);
121.         return 1;
122.
123.     }
124. }/*用户名字密码检验，是否已经注册（与文件内数据比较）*/
125. int login(char username[30], char password[30]) {
126.     char user[30];
127.     char pass[30];
128.     FILE *cfptr;//文件指针
129.     if ((cfptr = fopen("user.ini", "r")) == nullptr) {
130.         printf("File client.txt could not be opened\n");
131.         fclose(cfptr);
132.         return 0;
133.     } else {
134.         while (!feof(cfptr)) {
135.             fscanf(cfptr, "%s%s", user, pass);
136.             if ((strcmp(username, user) == 0) && (strcmp(password, pass) == 0))
            {
137.                 fclose(cfptr);
138.                 return 1;
139.             }
140.         }
141.     }
142.     fclose(cfptr);

```

```

143.     return 0;
144. } //登录
145. int operate(char name[30], char pass[30]) {
146.     func_filesystem af_filesystem;
147.     af_filesystem.setUser(name, pass);
148.     while (1) {
149.         system("CLS");
150.         while (1) {
151.             cout << endl;
152.             string choice;
153.             af_filesystem.showPath();
154.             cin >> choice;
155.             if (choice == "mkdir")
156.                 af_filesystem.newDir();
157.             else if (choice == "touch")
158.                 af_filesystem.newFile();
159.             else if (choice == "rmdir")
160.                 af_filesystem.deleteDir();
161.             else if (choice == "rm")
162.                 af_filesystem.deleteFile();
163.             else if (choice == "cd")
164.                 af_filesystem.open_dir();
165.             else if (choice == "open")
166.                 af_filesystem.open_file();
167.             else if (choice == "ls")
168.                 af_filesystem.ls();
169.             else if (choice == "cp")
170.                 af_filesystem.copyFile();
171.             else if (choice == "ps") {
172.                 af_filesystem.pasteFile();
173.             } else if (choice == "vi")
174.                 af_filesystem.write();
175.             else if (choice == "cd..")
176.                 af_filesystem.goback();
177.             else if (choice == "pwd")
178.                 af_filesystem.pwd();
179.             else if (choice == "clear") {
180.                 system("CLS");
181.                 help();
182.             } else if (choice == "exit") {
183.                 system("CLS");
184.                 cout << "用户: " << name << "正在注销"
185.                     << endl;
186.                 Sleep(3);

```

```

187.         return 0;
188.     } else if (choice == "help") {
189.         help();
190.     }
191.     else
192.         cout << "命令无效! " << endl;
193.     }
194. }
195.
196. } //选择操作

```

2.ui_filesystem.h

```

1. //
2. // Created by AlexFan on 2020/3/2.
3. //
4.
5. #ifndef AF_FILESYSTEM_UI_FILESYSTEM_H
6. #define AF_FILESYSTEM_UI_FILESYSTEM_H
7.
8. void start();
9. void help();
10. int regist(char username[30], char password[30]);
11. int login(char username[30], char password[30]);
12. int operate(char name[30], char pass[30]);
13.
14. #endif //AF_FILESYSTEM_UI_FILESYSTEM_H

```

3.func_filesystem.cpp

```

1. //
2. // Created by AlexFan on 2020/3/2.
3. //
4.
5. #include "func_filesystem.h"
6. #include<iostream>
7. #include<cstring>
8.
9. using namespace std;
10. int disk_storage=10000; //全局变量虚拟磁盘空闲空间大小
11. string error[] = {"/", "\\", ":", "<", ">", "|", "*", "&"} ; //命名中的非法字符
12.
13. func_filesystem::func_filesystem() {
14.     size = 0;

```



```

15.     currentDir = nullptr;
16.     copytempfile = nullptr;
17. }//初始化文件系统
18.
19. func_filesystem::~~func_filesystem() { //释放用户所占空间
20.     disk_storage += size;
21.     size = 0;                // 置 0
22.
23.     directory *d = root;
24.     file *f = currentDir->filePtr;
25.     while (f != nullptr) {
26.         if (f->nextFile == nullptr) {
27.             this->dele_file(f);
28.             f = nullptr;
29.             break;
30.         }
31.         while (f->nextFile->nextFile != nullptr)
32.             f = f->nextFile;
33.         this->dele_file(f->nextFile);
34.         f->nextFile = nullptr;
35.         f = currentDir->filePtr;
36.     }
37.     while (d != nullptr) {
38.         if (d->nextDir == nullptr) {
39.             this->dele_dir(d);
40.             d = nullptr;
41.             break;
42.         }
43.         while (d->nextDir->nextDir != nullptr)
44.             d = d->nextDir;
45.         this->dele_dir(d->nextDir);
46.         d->nextDir = nullptr;
47.         d = root;
48.     }
49.     cout << "已释放所有资源" << endl;
50. }//释放存储空间
51.
52. int func_filesystem::newFile() {
53.
54.     file *p = nullptr;
55.     p = new file;
56.
57.     if (p == nullptr) {
58.         cout << "CREATE          -FALSE";

```

```

59.         return 0;
60.     }
61.     cin>>p->name ;
62.     /*命名检测*/
63.     string tempname(p->name) ;
64.     for(int i = 0 ;i< 8 ;++i)
65.     {
66.         if(tempname.find(error[i],0)!=string::npos)//从字符串的下标为0处开始查找 error[i],如果没找到, 返回一个特别的标志 c++中用 npos 表示
67.         {
68.             cout << "RENAME          -FALSE"<<endl;
69.             return 0 ;
70.         }
71.     }
72.
73.     /*创建时候情况如下
74.     * 1. 目录下没有文件
75.     * 2. 目录下有文件, 新文件命名冲突
76.     * 3. 目录下有文件, 新文件无命名冲突
77.     * */
78.     /*检测有无同名函数*/
79.     if (currentDir->filePtr == nullptr) {
80.         p->nextFile = currentDir->filePtr;
81.         currentDir->filePtr = p;
82.     } else {
83.         file *q = new file;
84.         q = currentDir->filePtr;
85.         while (q != nullptr) {
86.             if (strcmp(p->name, q->name)==0) {
87.                 cout << "FILE EXISTS          -FALSE" << endl;
88.                 return 0;
89.             }
90.             q = q->nextFile;
91.         }
92.
93.         /*重置链表结构*/
94.         p->nextFile = currentDir->filePtr;
95.         //p->size=0;
96.         currentDir->filePtr = p;
97.         directory *h = currentDir;
98.
99.         /*更改上级目录的大小*/
100.        while (h != nullptr) {
101.            h->size += p->size;

```

```

102.         h = h->preDir;
103.     }
104.
105. }
106. currentDir->filePtr->size = 0;
107. cout <<"CREATE          -OK" << endl;
108. disk_storage = disk_storage - p->size;
109. size += p->size;
110. return 1;
111. }//touch
112.
113. int func_filesystem::newDir() {
114.     directory *p, *h;
115.     p = new directory;
116.     cin >> p->name;
117.
118.     /*命名检测*/
119.     string tempname(p->name) ;
120.     for(int i = 0 ;i< 8 ;++i)
121.     {
122.         if(tempname.find(error[i],0)!=string::npos)
123.         {
124.             cout << "RENAME          -FALSE"<<endl;
125.             return 0 ;
126.         }
127.     }
128.
129.
130.     p->dirPtr = nullptr;
131.     p->size = 0;
132.     p->filePtr = nullptr;
133.     p->nextDir = nullptr;
134.     if (currentDir->dirPtr == nullptr)
135.         h = nullptr;
136.     else
137.         h = currentDir->dirPtr;
138.
139.     /*创建时候情况如下
140.      * 1. 目录下没有子目录
141.      * 2. 目录下有子目录, 命名冲突
142.      * 3. 目录下有子目录, 无命名冲突
143.      * */
144.     /*检测有无同名目录*/
145.

```

```

146.     while (h != nullptr) {
147.         if (strcmp(h->name, p->name)==0) {
148.             cout << "DIR EXISTS" << "-FALSE" << endl;
149.             return 0;
150.         }
151.         h = h->nextDir;
152.     }
153.
154.     /*重置链表结构*/
155.     p->preDir = currentDir;
156.     p->nextDir = currentDir->dirPtr;
157.     currentDir->dirPtr = p;
158.
159.     cout << "CREATE" << "-OK" << endl;
160.     return 1;
161. }//mkdir
162.
163. int func_filesystem::dele_file(file *f) {
164.     delete f;
165.     f = nullptr;
166.     return 1;
167. }//rm
168.
169. int func_filesystem::deleteFile() {
170.     char temp[NAME_LENGTH];
171.
172.     cin >> temp;
173.     file *f;
174.     file *above = nullptr;
175.     f = currentDir->filePtr;
176.
177.     /*
178.      * 判断该目录下有无需要删除的文件
179.      * */
180.
181.     while (f != nullptr) {
182.         if (!strcmp(f->name, temp))
183.             break;
184.         above = f;
185.         f = f->nextFile;
186.     }
187.     if (f == nullptr) {
188.         cout << "NO FILE" << "-FALSE" << endl;
189.         return 0;

```

```

190.     }
191.     disk_storage += f->size;
192.     directory *d = currentDir;
193.     while (d != 0) //修改删除文件后各级目录的大小
194.     {
195.         d->size -= f->size;
196.         d = d->preDir;
197.     }
198.
199.     /*
200.      * 删除时考虑
201.      * 1. 需要删除的文件恰好是目录文件链表中的头节点
202.      * 2. 需要删除的文件在链表中间
203.      * */
204.
205.     if (f == currentDir->filePtr)//删除文件结点
206.         currentDir->filePtr = currentDir->filePtr->nextFile;
207.     else
208.         above->nextFile = f->nextFile;
209.     size -= f->size;
210.     delete f;
211.     f = nullptr;
212.     cout << "DELETE          -OK" << endl;
213.     return 1;
214. }//rm
215.
216. int func_filesystem::dele_dir(directory *d) {
217.     delete d;
218.     d = nullptr;
219.     return 1;
220. }//rmdir
221.
222. int func_filesystem::deleteDir() {
223.     char n[NAME_LENGTH];
224.
225.     directory *p, *pre = nullptr;
226.     p = root;
227.     p = currentDir->dirPtr;
228.     cin >> n; //删除的文件名
229.
230.     /*查找所需要删除的目录*/
231.     while (p != nullptr) {
232.         if (strcmp(p->name, n)==0)
233.             {pre = p;break;}

```

```

234.         p = p->nextDir;
235.     }
236.
237.     if (p == nullptr) {
238.         cout << "DELETE          -FALSE" << endl;
239.         return 0;
240.     }
241.
242.     /*删除目录时需要考虑
243.      * 1. 该目录处于父目录的目录链表的位置
244.      * 2. 该目录下是否有子目录或者子文件
245.      * */
246.     disk_storage += p->size;
247.     if (p == currentDir->dirPtr)
248.         currentDir->dirPtr = currentDir->dirPtr->nextDir;
249.     else
250.         p->preDir->nextDir = p->nextDir;
251.
252.     pre = currentDir;
253.     while (pre != nullptr) //修改删除目录后各级目录大小
254.     {
255.         pre->size -= p->size;
256.         pre = pre->preDir;
257.     }
258.     size -= p->size;
259.     directory *d = p->dirPtr;
260.     file *f = p->filePtr;
261.     if (f != nullptr) {
262.         while (p->filePtr->nextFile != nullptr) //删除此目录下的文件
263.         {
264.             f = p->filePtr;
265.             while (f->nextFile->nextFile != nullptr) //寻找最后一个文件结点
266.                 f = f->nextFile;
267.             this->dele_file(f->nextFile);
268.             f->nextFile = nullptr;
269.         }
270.         if (p->filePtr->nextFile == nullptr) {
271.             this->dele_file(p->filePtr);
272.             p->filePtr = nullptr;
273.         }
274.     }
275.     if (d != nullptr) {
276.         while (p->dirPtr->nextDir != nullptr) //删除此目录下的目录
277.         {

```

```

278.         d = p->dirPtr;
279.         while (d->nextDir->nextDir != nullptr)//寻找最后一个文件结点
280.             d = d->nextDir;
281.         this->dele_dir(d->nextDir);//递归调用此函数
282.         d->nextDir = nullptr;
283.     }
284.     if (p->dirPtr->nextDir == nullptr) {
285.         this->dele_dir(p->dirPtr);
286.         p->dirPtr = nullptr;
287.     }
288. }
289. delete p,d,f;
290.
291. cout << "DELETE          -OK" << endl;
292. return 1;
293.
294. }//rmdir
295.
296. int func_filesystem::open_dir() {
297.     char name[NAME_LENGTH];
298.     directory *p;
299.     p = currentDir->dirPtr;
300.
301.     cin >> name;
302.     while (p != nullptr) {
303.         if (strcmp(p->name, name) == 0) {
304.             currentDir = p;
305.             return 1;
306.         }
307.         p = p->nextDir;
308.     }
309.     cout << "NO DIR          -FALSE" << endl;
310.     return 0;
311. }//cd
312.
313. int func_filesystem::open_file() {
314.     char n[NAME_LENGTH];
315.     cin >> n;
316.     file *f = currentDir->filePtr;
317.     while (f != nullptr) {
318.         if (strcmp(f->name, n)==0) {
319.             cout << f->content << endl;
320.             return 1;
321.         }

```

```

322.         f = f->nextFile;
323.     }
324.     cout << "NO FILE" << " -FALSE" << endl;
325.     return 0;
326. } //open
327.
328. file *func_filesystem::copy_file(file *h) {
329.     file *f;
330.     f = new file;
331.     f->size = h->size;
332.     strcpy(f->name, h->name);
333.     f->content = h->content;
334.     return f;
335. } //cp
336.
337. file *func_filesystem::copyFile() {
338.     file *h;
339.     char n[NAME_LENGTH];
340.     cin >> n;
341.     h = currentDir->filePtr;
342.
343.     while (h != nullptr) {
344.         if (!strcmp(h->name, n))
345.             break;
346.         h = h->nextFile;
347.     }
348.     if (h == nullptr) {
349.         cout << "NO FILE" << " -FALSE" << endl;
350.         return nullptr;
351.     }
352.     copytempfile = copy_file(h);
353.     cout << "COPY" << " -OK" << endl;
354.     return copytempfile;
355. } //cp
356.
357. int func_filesystem::pasteFile() {
358.     file *h = currentDir->filePtr;
359.     file *pTemp = h;
360.     if (copytempfile == nullptr) {
361.         cout << "NO SOURCE" << " -FALSE" << endl;
362.         return 0;
363.     }
364.     /*如果当前目录没有文件*/
365.     if (h == nullptr) {

```



```

366.         if (disk_storage < copytempfile->size) {
367.             cout << "NO ENOUGH SPACE" << endl;
368.             return 0;
369.         }
370.         currentDir->filePtr = copy_file(copytempfile);
371.         currentDir->size += copytempfile->size;
372.     }
373.     else {
374.         while (h != nullptr) {
375.             if (!strcmp(h->name, copytempfile->name)) {
376.                 cout << "FILE EXISTS" << endl;
377.                 return 0;
378.             }
379.             //break;
380.             h = h->nextFile;
381.         }
382.
383.         if (disk_storage < copytempfile->size) {
384.             cout << "NO ENOUGH SPACE" << endl;
385.             return 0;
386.         }
387.         currentDir->filePtr = copy_file(copytempfile);
388.         currentDir->filePtr->nextFile = pTemp;
389.         //currentDir->filePtr->nextFile = h;
390.         currentDir->size += copytempfile->size;
391.         cout << "PASTE" << endl;
392.         return 1;
393.     }
394.     return 0;
395.
396. } //paste
397.
398. int func_filesystem::write() {
399.     char n[NAME_LENGTH];
400.     string s;
401.     cin >> n;
402.     file *f = currentDir->filePtr;
403.     while (f != 0) {
404.         if (!strcmp(f->name, n)) {
405.             cin >> s;
406.             f->content = s;
407.             f->size = s.length();
408.             disk_storage -= f->size;
409.             directory *d = currentDir;

```

```

410.         while (d != 0)//修改编辑文件后各级目录的大小
411.         {
412.             d->size += f->size;
413.             d = d->preDir;
414.         }
415.         cout << "EDIT          -OK" << endl;
416.         size += f->size;
417.         return 1;
418.     }
419.     f = f->nextFile;
420. }
421.
422. cout << "NO FILE          -FALSE" << endl;
423. return 0;
424. }//vi
425.
426.
427. int func_filesystem::ls() {
428.     directory *d = currentDir->dirPtr;
429.     file *f = currentDir->filePtr;
430.     if (d == nullptr && f == nullptr) {
431.         return 0;
432.     }
433.     if (d != nullptr) {
434.         while (d != nullptr) {
435.             cout << d->name<<endl;
436.             d = d->nextDir;
437.         }
438.         cout << endl;
439.     }
440.
441.     if (f != nullptr) {
442.         while (f != nullptr) {
443.             cout << f->name<< endl;
444.             f = f->nextFile;
445.         }
446.     }
447.     return 1;
448. }//打印目录下所有文件/文件夹
449.
450. int func_filesystem::show_path(directory *d)//实现显示当前路径的函数
451. {
452.     if (d->preDir == nullptr)
453.         cout << root->name;

```

```

454.     if (d->preDir != nullptr) {
455.         this->show_path(d->preDir); //递归调用此函数
456.         cout << d->name;
457.     }
458.     cout << "/";
459.     return 1;
460. }
461.
462. int func_filesystem::showPath()//显示当前路径
463. {
464.     cout<<"#";
465.     show_path(currentDir);
466.     return 1;
467. }
468. int func_filesystem::pwd(){
469.     cout << "The current directory: ";
470.     show_path(currentDir);
471.     return 0;
472. }//打印当前目录
473.
474. int func_filesystem::goback() {
475.     if (currentDir->preDir == nullptr) {
476.         cout << "当前目录为根目录" << endl;
477.         return 0;
478.     }
479.     currentDir = currentDir->preDir;
480.     return 1;
481. }//cd ..
482.
483. int func_filesystem::setUser(char *n, char *c) {
484.     directory *root = new directory;
485.     strcpy(root->name, n);
486.     strcpy(name, n);
487.     strcpy(password, c);
488.
489.     this->root = root;
490.     currentDir = root;
491.     currentDir->preDir = nullptr;
492.     currentDir->dirPtr = nullptr;
493.     currentDir->filePtr = nullptr;
494.     currentDir->nextDir = nullptr;
495.     return 1;
496. }

```

4.func_filesystem.h

```
1. //
2. // Created by AlexFan on 2020/3/2.
3. //
4.
5. #ifndef AF_func_filesystem_FUNC_func_filesystem_H
6. #define AF_func_filesystem_FUNC_func_filesystem_H
7. #include <iostream>
8. #include <cstring>
9. #include <string>
10. #include <iomanip>
11. #include <cstdlib>
12. #include <cstdio>
13. using namespace std;
14.
15. #define NAME_LENGTH 20 // ÎÄ¿ö»ðÕë¿ìÁû×Ôî´ó×ÖÝ
16. extern int disk_storage;
17.
18.
19. typedef struct file // ÎÄ¿ö¹¹
20. {
21.     char name[NAME_LENGTH]; // ÎÄ¿ö
22.     int size; // ÎÄ¿ö´óÐ
23.     struct file *nextFile; // Ò, ÎÄ¿öÁÐ±íÖÐ»ð, ÒÎÄ¿ö
24.     string content; // ÎÄ¿öÀÜÝ
25. } file;
26.
27. typedef struct directory // ¿ì¿ö¹¹
28. {
29.     char name[NAME_LENGTH]; // ¿ì¿öÁû×Ö
30.     int size; // ¿ì¿ö´óÐ
31.     directory *nextDir; // °ó¿ì¿ö
32.     directory *preDir; // Ç°¿ì¿ö
33.     file *filePtr; // , ¿ì¿öÎÄ¿öÁû±í±íÖ, Ò
34.     directory *dirPtr; // , ¿ì¿öÎÄ¿öÁû±í±íÖ, Ò
35. } directory;
36.
37.
38. class func_filesystem // ÎÄ¿öµ³À
39. {
40. private:
41.     directory *currentDir; // µÇ°¿ì¿ö
42.     file *copytempfile; // ÓÖ¿ö±´ÎÄ¿ö±µÀÜ±Í¿ö
43.     directory *root; // , ¿ì¿ö
```

```

44.     char password[NAME_LENGTH];    //ÓÃ»$ÃÜÂë
45.     char name[NAME_LENGTH];        //ÓÃ»$ÃÜ³Æ
46.     int size; //ÓÃ»$ËüÊ¹ÓÃ¿Õ%ä´óÐ;
47.
48. public:
49.     func_filesystem();//¹¹Ôì¹º-Êý
50.     ~func_filesystem();//Ïö¹¹º-Êý
51.
52.     /*
53.     *ÏÃ%p²Üx÷
54.     */
55. public:
56.     int newFile(); //´´%ÏÃ%p
57.     int dele_file(file *file); //É%³ýÏÃ%p
58.     int deleteFile(); //É%³ýÏÃ%pÇ°µÃÃß%ÃÐ¶Ï
59.     int open_file(); //´ð¿³ÏÃ%p
60.     file *copy_file(file *h); //´ÖÆÏÃ%p
61.     file *copyFile(); //´ÖÆÏÃ%pÇ°µÃÃß%ÃÐ¶Ï
62.     int pasteFile(); //Õ³ÏüÏÃ%p
63.     int write(); //±à%ÏÃ%p
64.
65.     /*
66.     * Ä¿Ã²²Üx÷
67.     */
68. public:
69.     int newDir(); //´´%Ä¿Ã%
70.     int dele_dir(directory *d); //É%³ýÄ¿Ã%
71.     int deleteDir(); //É%³ýÄ¿Ã%Ç°µÃÃß%ÃÐ¶Ï
72.     int open_dir(); //´ð¿³Ä¿Ã%
73.     int ls(); //ÏÔÊ%µ±Ç°Ä¿Ã%ÁÜÝ
74.
75.     /*
76.     * Â·%¶²²Üx÷
77.     */
78. public:
79.     int show_path(directory *dir); //ÏÔÊ%Â·%¶µÃ²¿·ÖÊµÏÖ
80.     int showPath(); //ÏÔÊ%Â·%¶
81.     int pwd();
82.     int goback(); //·µ»ØÏÖ»%¶Ä¿Ã%
83.
84.     /*
85.     * ÓÃ»$²Üx÷
86.     */
87. public:

```

```
88.     int setUser(char *, char *);  
89. };  
90. #endif //AF_func_filesystem_FUNC_func_filesystem_H
```

5.main.cpp

```
1. #include "ui_filesystem.h"  
2.  
3. int main() {  
4.     start();  
5.     return 0;  
6. }
```