



华中科技大学

操作系统原理课程设计报告

姓 名： 朱槐志
学 院： 计算机科学与技术学院
专 业： 计算机科学与技术
班 级： CS1807
学 号： U201814745
指导教师： 胡贯荣

| | |
|------|--|
| 分数 | |
| 教师签名 | |

2021 年 4 月 2 日

目 录

| | | |
|-----------|--------------------------------|-----------|
| 1 | 实验一 Linux 用户界面的使用 | 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..... | 1 |
| 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..... | 3 |
| | 附录 实验代码..... | 4 |
| | Task1..... | 4 |
| | Task2..... | 5 |
| 2. | 实验二 增加系统调用 | 17 |
| 2.1 | 实验目的..... | 17 |
| 2.2 | 实验内容..... | 17 |
| 2.3 | 实验设计..... | 17 |
| 2.3.1 | 开发环境..... | 17 |
| 2.3.2 | 实验设计..... | 18 |
| 2.4 | 实验调试..... | 18 |
| 2.4.1 | 实验步骤..... | 18 |
| 2.4.2 | 实验调试及心得..... | 20 |
| | 附录 实验代码..... | 20 |
| 3. | 实验三 增加设备驱动 | 23 |
| 3.1 | 实验目的..... | 23 |
| 3.2 | 实验内容..... | 23 |
| 3.3 | 实验设计..... | 23 |
| 3.3.1 | 开发环境..... | 23 |
| 3.3.2 | 实验设计..... | 23 |
| 3.4 | 实验调试..... | 24 |
| 3.4.1 | 实验步骤..... | 24 |
| 3.4.2 | 实验调试及心得..... | 25 |
| | 附录 实验代码..... | 27 |
| 4 | 实验四 /proc 文件分析..... | 34 |
| 4.1 | 实验目的..... | 34 |
| 4.2 | 实验内容..... | 34 |
| 4.3 | 实验设计..... | 34 |
| 4.3.1 | 开发环境..... | 34 |
| 4.3.2 | 实验设计..... | 34 |

| | | |
|-------|--------------|----|
| 4.4 | 实验调试..... | 35 |
| 4.4.1 | 实验步骤..... | 35 |
| 4.4.2 | 实验调试及心得..... | 37 |
| 附录 | 实验代码..... | 39 |

1 实验一 Linux 用户界面的使用

1.1 实验目的

正文统一采用小四号宋体/Times New Roman 和 1.25 倍行距。

1.2 实验内容

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

选择：容错、cp。

- 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）

基本要求：三个独立子进程，各自窗口显示；

选择：三个进程誊抄演示。

1.3 实验设计

1.3.1 开发环境

WSL(Windows Subsystem for Linux)

OS:Ubuntu 20.04.1 LTS x86_64

Kernel: 5.9.9silence2.0Yoga2021-standard

Shell: bash 5.0.17

宿主机：

系统：Windows10 家庭中文版，64 位操作系统，基于 x64 的版本号： 2004

处理器：11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz

内存：16.0GB(15.8 GB 可用)

1.3.2 实验设计

- 1) 了解 read, write, open 系统调用函数的使用

Write 系统调用函数原型:

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

它接受三个参数,第一个参数是文件的文件描述符,第二个参数是缓冲区的地址,第三个参数是要写入的字节个数。它能把缓冲区 buf 中的前 nbytes 字节写入到与文件描述符 flides 有关的文件中,并返回实际写入到文件中的字节数。

Read 系统调用函数原型:

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

它接受三个参数,与 write 完全相同,只不过它是从与文件描述符 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。

Open 系统调用函数原型:

```
int open(const *path, int oflags);
```

将准备打开的文件或是设备的名字作为参数 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。

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

在前一种调用方式上加上了第三个参数 mode,主要搭配 O_CREAT 使用,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。

2) task1 设计

函数接受三个参数,第一个参数默认为编译后程序名,第二个参数是已经存在的源文件文件名,第三个参数是目标文件名。因为没有指定文件路径,所以需把编译后的程序和源文件,目标文件放在同一目录下。

首先判断命令行参数个数,如果不等于三,则输出:参数错误,直接返回。参数个数正确后,以只读打开源文件,如果打开失败,则输出:要复制的源文件不存在!直接返回。

然后打开用 creat 系统调用函数以只写方式打开目标文件,这样无论目标文件存不存在都能成功打开并返回文件描述符。

接着循环从源文件中读取字节写到缓冲区中,再从缓冲区中读取字节写入到目标文件中。一次循环读取 1024 个字节。当 read 返回值小于等于 0 的时候循环退出,即复制文件结束。

3) task2 设计

我一共做了三个并发进程,通过 fork() 来实现。分别显示斐波那契数列的当前值,1-1000 累积动态求和,显示当前时间。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。

1.4 实验调试

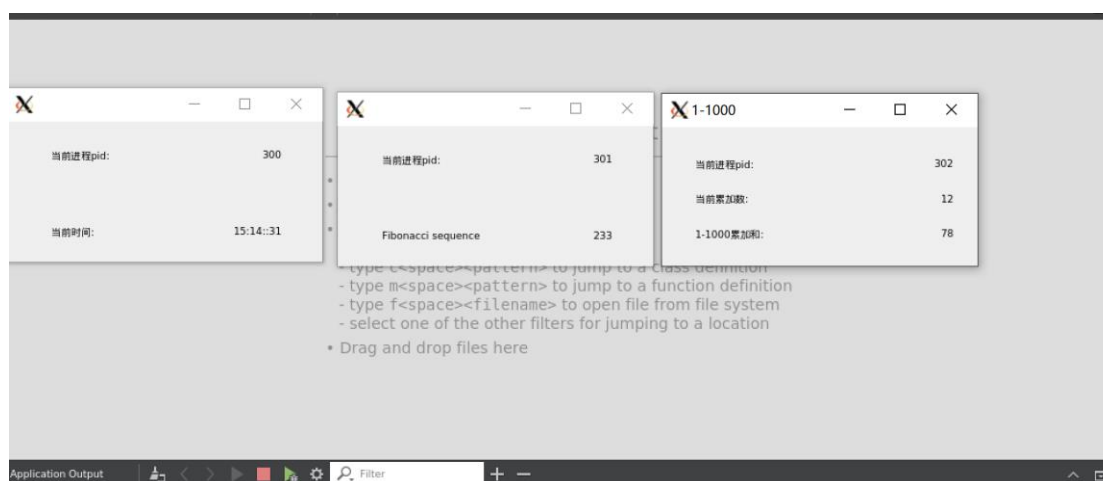
1.4.1 实验步骤

task1:

一开始打开目标文件我用的是 open 系统调用函数，但是如果目标文件不存在的话程序就会因为文件打开失败就自动返回，不是很符合我们的常识。后来想的是先判断是否打开失败，如果打开失败在用 creat 创建。再后来了解到即使 creat 的文件是已存在的，它还是会返回正确的文件描述符，故省略了判断那段代码，去掉 open，直接使用 creat，这样如果目标文件不存在的话就会自动创建了，如果创建失败就会输出错误然后返回。

task2:

1. 从系统资源文件中获取 TIME 信息
2. 每秒刷新一次，并通过 slot 传输到 QT
3. 编写 main 界面
4. 编写信息显示界面
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 read, write, open, creat 等系统调用函数的使用

- (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实验中的进程知识
- (5) 学习到了 Linux 系统文件结构

附录 实验代码

Task1

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

int main(int argc, char* argv[]){
    int fd, fd1, fd2;
    char bufs[1024];
    int len;
    if(argc != 3)
    {
        printf("参数错误\n");
    }
    else
    {
        //argv[1]是源文件
        //argv[2]为目标文件
        //打开源文件，返回新的文件描述符
        fd = open(argv[1], O_RDONLY|O_CREAT);
        if(fd != -1)
        {
            //先创建，创建失败再打开
            fd1 = creat(argv[2], 0775);

            if(fd1 != -1)
            {
                fd2 = open(argv[2], O_WRONLY);
                while( (len = read(fd, bufs, 1024)) > 0)
                    write(fd2, bufs, len);
            }
        }
    }
}
```

```

        else
            printf("创建文件失败\n");
    }
    else
        printf("要复制的文件不存在\n");
}
return 0;
}

```

Task2

(1) main.cpp

```

#include "widget.h"
#include "widget1.h"
#include "widget2.h"
#include <QApplication>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int pid;
    if ((pid = fork()) == 0)
    {
        QApplication a(argc, argv);
        widget1 w;
        w.setWindowTitle("斐波那契数列");
        w.show();
        a.exec();
        exit(0);
    }
    if ((pid = fork()) == 0)
    {
        QApplication a(argc, argv);
        widget2 w;
        w.setWindowTitle("1-1000 累加求和");
        w.show();
        a.exec();
        exit(0);
    }
    QApplication a(argc, argv);
    Widget w;
    w.setWindowTitle("时间显示窗口");
}

```



```

        w.show();
        return a.exec();
    }
}

(2) widget.cpp
#include "widget.h"
#include "ui_widget.h"
#include <unistd.h>
#include <QDateTime>
#include <QTimer>

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    this->setFixedSize(this->width(), this->height());
    this->move(600, 400);

    int pid = getpid();
    ui->label_2->setText(QString::number(pid, 10));

    QDateTime curtime = QDateTime::currentDateTime();
    ui->label_4->setText(curtime.toString("hh:mm:ss"));

    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(Update()));
    timer->start(1000);
}

void Widget::Update(){
    QDateTime curtime = QDateTime::currentDateTime();
    ui->label_4->setText(curtime.toString("hh:mm:ss"));
}

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

(3) widget1.cpp
#include "widget1.h"
#include "ui_widget1.h"
#include <unistd.h>
#include <QTimer>

```

```

widget1::widget1(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::widget1)
{
    ui->setupUi(this);
    this->setFixedSize(this->width(), this->height());
    this->move(900,400);

    int pid = getpid();
    ui->label_2->setText(QString::number(pid, 10));

    num = 1;
    num2 = 1;
    ui->label_4->setText(QString::number(num, 10));

    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(Update()));
    timer->start(1000);
}

```

```

void widget1::Update(){

    ui->label_4->setText(QString::number(num, 10));
    int temp = num;
    num = num + num2;
    num2 = temp;
}

```

```

widget1::~~widget1()
{
    delete ui;
}

```

(4) widget2.cpp

```

#include "widget2.h"
#include "ui_widget2.h"
#include <unistd.h>
#include <QTimer>

```

```

widget2::widget2(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::widget2)
{
    ui->setupUi(this);
    this->setFixedSize(this->width(), this->height());
}

```

```

this->move(1200, 400);

int pid = getpid();
ui->label_2->setText(QString::number(pid, 10));

sum = 0;
num = 1;
ui->label_4->setText(QString::number(sum, 10));

QTimer *timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(Update()));
timer->start(1000);
}

```

```

void widget2::Update(){
    if(num <= 1000)
        sum += num++;
    ui->label_4->setText(QString::number(sum, 10));
    ui->label_6->setText(QString::number(num - 1, 10));
}

```

```

widget2::~widget2()

```

```

{
    delete ui;
}

```

(5) widget.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
    <class>Widget</class>
    <widget class="QWidget" name="Widget">
        <property name="geometry">
            <rect>
                <x>0</x>
                <y>0</y>
                <width>285</width>
                <height>126</height>
            </rect>
        </property>
        <property name="windowTitle">
            <string>Widget</string>
        </property>
    <widget class="QWidget" name="layoutWidget">
        <property name="geometry">
            <rect>
                <x>40</x>

```

```

    <y>20</y>
    <width>209</width>
    <height>89</height>
</rect>
</property>
<layout class="QVBoxLayout" name="verticalLayout">
  <item>
    <layout class="QHBoxLayout" name="horizontalLayout">
      <item>
        <widget class="QLabel" name="label">
          <property name="text">
            <string>当前进程 pid:</string>
          </property>
        </widget>
      </item>
      <item>
        <spacer name="horizontalSpacer">
          <property name="orientation">
            <enum>Qt::Horizontal</enum>
          </property>
          <property name="sizeHint" stdset="0">
            <size>
              <width>40</width>
              <height>20</height>
            </size>
          </property>
        </spacer>
      </item>
      <item>
        <widget class="QLabel" name="label_2">
          <property name="text">
            <string>TextLabel</string>
          </property>
        </widget>
      </item>
    </layout>
  </item>
  <item>
    <spacer name="verticalSpacer">
      <property name="orientation">
        <enum>Qt::Vertical</enum>
      </property>
      <property name="sizeHint" stdset="0">
        <size>

```

```

        <width>20</width>
        <height>28</height>
    </size>
</property>
</spacer>
</item>
<item>
    <layout class="QHBoxLayout" name="horizontalLayout_2">
        <item>
            <widget class="QLabel" name="label_3">
                <property name="text">
                    <string>当前时间:</string>
                </property>
            </widget>
        </item>
        <item>
            <spacer name="horizontalSpacer_2">
                <property name="orientation">
                    <enum>Qt::Horizontal</enum>
                </property>
                <property name="sizeHint" stdset="0">
                    <size>
                        <width>40</width>
                        <height>20</height>
                    </size>
                </property>
            </spacer>
        </item>
        <item>
            <widget class="QLabel" name="label_4">
                <property name="text">
                    <string>TextLabel</string>
                </property>
            </widget>
        </item>
    </layout>
</item>
</layout>
</widget>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>

```

(6) widget1.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>widget1</class>
  <widget class="QWidget" name="widget1">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>285</width>
        <height>126</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>Widget</string>
    </property>
    <widget class="QWidget" name="layoutWidget">
      <property name="geometry">
        <rect>
          <x>40</x>
          <y>20</y>
          <width>209</width>
          <height>89</height>
        </rect>
      </property>
      <layout class="QVBoxLayout" name="verticalLayout">
        <item>
          <layout class="QHBoxLayout" name="horizontalLayout">
            <item>
              <widget class="QLabel" name="label">
                <property name="text">
                  <string>当前进程 pid:</string>
                </property>
              </widget>
            </item>
            <item>
              <spacer name="horizontalSpacer">
                <property name="orientation">
                  <enum>Qt::Horizontal</enum>
                </property>
                <property name="sizeHint" stdset="0">
                  <size>
                    <width>40</width>
                    <height>20</height>
                  </size>
                </property>
              </spacer>
            </item>
          </layout>
        </item>
      </layout>
    </widget>
  </widget>
</ui>
```

```

        </size>
    </property>
</spacer>
</item>
<item>
    <widget class="QLabel" name="label_2">
        <property name="text">
            <string>TextLabel</string>
        </property>
    </widget>
</item>
</layout>
</item>
<item>
    <spacer name="verticalSpacer">
        <property name="orientation">
            <enum>Qt::Vertical</enum>
        </property>
        <property name="sizeHint" stdset="0">
            <size>
                <width>20</width>
                <height>28</height>
            </size>
        </property>
    </spacer>
</item>
<item>
    <layout class="QHBoxLayout" name="horizontalLayout_2">
        <item>
            <widget class="QLabel" name="label_3">
                <property name="text">
                    <string>Fibonacci sequence</string>
                </property>
            </widget>
        </item>
        <item>
            <spacer name="horizontalSpacer_2">
                <property name="orientation">
                    <enum>Qt::Horizontal</enum>
                </property>
                <property name="sizeHint" stdset="0">
                    <size>
                        <width>40</width>
                        <height>20</height>
                    </size>
                </property>
            </spacer>
        </item>
    </layout>
</item>

```

```

        </size>
    </property>
</spacer>
</item>
<item>
    <widget class="QLabel" name="label_4">
        <property name="text">
            <string>TextLabel</string>
        </property>
    </widget>
</item>
</layout>
</item>
</layout>
</widget>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>

```

(7) widget2.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
    <class>widget2</class>
    <widget class="QWidget" name="widget2">
        <property name="geometry">
            <rect>
                <x>0</x>
                <y>0</y>
                <width>285</width>
                <height>125</height>
            </rect>
        </property>
        <property name="windowTitle">
            <string>Widget</string>
        </property>
    <widget class="QWidget" name="">
        <property name="geometry">
            <rect>
                <x>30</x>
                <y>20</y>
                <width>234</width>
                <height>89</height>
            </rect>
        </property>
    </widget>
</ui>

```



```

</property>
<layout class="QVBoxLayout" name="verticalLayout">
  <item>
    <layout class="QHBoxLayout" name="horizontalLayout">
      <item>
        <widget class="QLabel" name="label">
          <property name="text">
            <string>当前进程 pid:</string>
          </property>
        </widget>
      </item>
      <item>
        <spacer name="horizontalSpacer">
          <property name="orientation">
            <enum>Qt::Horizontal</enum>
          </property>
          <property name="sizeHint" stdset="0">
            <size>
              <width>40</width>
              <height>20</height>
            </size>
          </property>
        </spacer>
      </item>
      <item>
        <widget class="QLabel" name="label_2">
          <property name="text">
            <string>TextLabel</string>
          </property>
        </widget>
      </item>
    </layout>
  </item>
  <item>
    <layout class="QHBoxLayout" name="horizontalLayout_3">
      <item>
        <widget class="QLabel" name="label_5">
          <property name="text">
            <string>当前累加数:</string>
          </property>
        </widget>
      </item>
      <item>
        <spacer name="horizontalSpacer_3">

```

```

    <property name="orientation">
      <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
      <size>
        <width>40</width>
        <height>20</height>
      </size>
    </property>
  </spacer>
</item>
<item>
  <widget class="QLabel" name="label_6">
    <property name="text">
      <string>TextLabel</string>
    </property>
  </widget>
</item>
</layout>
</item>
<item>
  <layout class="QHBoxLayout" name="horizontalLayout_2">
    <item>
      <widget class="QLabel" name="label_3">
        <property name="text">
          <string>1-1000 累加和:</string>
        </property>
      </widget>
    </item>
    <item>
      <spacer name="horizontalSpacer_2">
        <property name="orientation">
          <enum>Qt::Horizontal</enum>
        </property>
        <property name="sizeHint" stdset="0">
          <size>
            <width>40</width>
            <height>20</height>
          </size>
        </property>
      </spacer>
    </item>
    <item>
      <widget class="QLabel" name="label_4">

```

```
<property name="text">
  <string>TextLabel</string>
</property>
</widget>
</item>
</layout>
</item>
</layout>
</widget>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>
```

2.实验二 增加系统调用

2.1 实验目的

1. 掌握 linux 内核编译的过程
2. 掌握给 linux 内核增加系统调用的方法
3. 掌握系统调用函数的编写
4. 掌握内核中的 api

2.2 实验内容

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

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

2.3 实验设计

2.3.1 开发环境

WSL(Windows Subsystem for Linux)

OS:Ubuntu 20.04.1 LTS x86_64

Kernel: 5.9.9silence2.0Yoga2021-standard

Shell: bash 5.0.17

宿主机：

系统：Windows10 家庭中文版，64 位操作系统，基于 x64 的版本号： 2004

处理器： 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz

内存： 16.0GB(15.8 GB 可用)

2.3.2 实验设计

- (1) 下载内核文件并解压
- (2) 编译内核，确保能编译成功再修改内核文件，增加系统调用
- (3) 编译内核
- (4) 编写测试程序，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

WSL 编译内核整体步骤与通常编译 Linux 内核很相似，只是 Makefile 使用的.config 文件来自微软；同时 WSL2 也不是用 grub 启动的，因此需要在 windows 的某目录下替换内核文件。

1. 下载内核文件到 home 目录
2. 因为我们使用的是微软的配置，所以要设置编译配置文件。到 WSL2 的 github 源码处复制配置文件内容到本地.config 文件中。
3. 配置文件添加个人信息，打造属于自己的内核版本。修改 Makefile 里的 EXTRAVERSION ， 改 为 silence 。 再 修 改 .config 里 的 CONFIG_LOCALVERSION，把”-microsoft-standard”改为”silence”。
4. 安装编译依赖项：

```
apt update #更新源
```

```
apt install build-essential flex bison libssl-dev libelf-dev
```

5. 正式编译，因为自己的电脑是 8 核的，所有编译的时候线程数设成了 16
- ```
make -j16;make modules -j16;make modules_install -j16;make
install -j16
```





//增加系统调用函数体

```
SYSCALL_DEFINE2(mcf,const char __user*,source,const char __user*,target){
 long ret = 0;
 char buffer[1000];
 // 转化为内核存储空间
 char * source_kd = strndup_user(source, PAGE_SIZE);
 char * target_kd = strndup_user(target, PAGE_SIZE);
 // 异常处理
 //判断返回指针是否错误
 if (IS_ERR(source_kd)) {
 printk("source path error1");
 ret = PTR_ERR(source);//如果出错了，返回指针里保存的错误代码
 goto error;
 }
 if (IS_ERR(target_kd)) {
 printk("target path error1");
 ret = PTR_ERR(target);
 goto error;
 }
 printk("source path : %s, target path : %s", source_kd, target_kd);

 //防止使用的缓冲区超过了用户空间的地址范围
 mm_segment_t old_fs = get_fs();
 set_fs(KERNEL_DS);
 //文件打开
 struct file* source_fd = filp_open(source_kd, O_RDONLY, S_IRUSR);
 struct file* target_fd = filp_open(target_kd, O_WRONLY | O_CREAT,
S_IRUSR|S_IWUSR);
 // 异常处理
 if (IS_ERR(source_fd)) {
 ret = PTR_ERR(source);
 printk("source file path error2\n");
 goto error;
 }
 if (IS_ERR(target_fd)) {
 ret = PTR_ERR(target);
 printk("target file path error2\n");
 goto error;
 }
 // 读取文件
 int read_num = 0;
```



```

// vfs_read(source_fd, buffer, sizeof(buffer), &(source_fd->f_pos));
// vfs_write(target_fd, buffer, sizeof(buffer), &(target_fd->f_pos));
int count = 0;
while ((read_num = vfs_read(source_fd,buffer, sizeof(buffer), &source_fd->f_pos)) > 0) {
 vfs_write(target_fd, buffer, read_num, &target_fd->f_pos);
 count += 1;
}
//读写完成后再恢复原先 fs
set_fs(old_fs);

error:
 return ret;
}

```

## 3.实验三 增加设备驱动

### 3.1 实验目的

1. 掌握增加设备驱动程序的方法
2. 掌握设备驱动程序的编写
3. 了解字符设备的使用

### 3.2 实验内容

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

基本要求：演示实现字符设备读、写；

选择：键盘缓冲区，不同进程、追加、读取。

### 3.3 实验设计

#### 3.3.1开发环境

WSL(Windows Subsystem for Linux)

OS:Ubuntu 20.04.1 LTS x86\_64

Kernel: 5.9.9silence2.0Yoga2021-standard

Shell: bash 5.0.17

宿主机：

系统：Windows10 家庭中文版，64 位操作系统，基于 x64 的

版本号： 2004

处理器：11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz

内存：16.0GB( 15.8 GB 可用)

#### 3.3.2实验设计

首先是编写设备驱动程序，涉及的内容主要有

- 1) 完成函数关系绑定

2) 模块的初始化、需要的功能、模块的卸载以及释放内存。 其次是修改通用的 Makefile 文件完成编译操作。然后使用设备驱动模块的 加载指令完成加载。再者，使用设备文件指令使其生成对应的设备文件名。最后再编写应用程序对新添加的设备进行测试。

在 file\_operations 结构体 fops 中存在着大量的函数入口点，在这里指定我们要实现的各个函数如下：

```
static const struct file_operations fops =
{
 .owner = THIS_MODULE,
 .open = driver_open,
 .release = driver_release,
 .read = driver_read,
 .write = driver_write,
 .llseek = driver_llseek,
 .unlocked_ioctl = driver_ioctl,
};
```

| 入口点             | 函数原型                                                                                                       | 功能            |
|-----------------|------------------------------------------------------------------------------------------------------------|---------------|
| .open           | static int zxcypdriver_open(struct inode *inodep, struct file *filep);                                     | 打开设备          |
| .release        | static int zxcypdriver_release(struct inode *inodep, struct file *filep)                                   | 关闭设备          |
| .read           | static ssize_t zxcypdriver_read(struct file *filep, char __user *buf, size_t count, loff_t *offset)        | 从设备中读数据       |
| .write          | static ssize_t zxcypdriver_write(struct file *filep, const char __user *buf, size_t count, loff_t *offset) | 向设备中写数据       |
| .llseek         | static loff_t zxcypdriver_llseek(struct file *filep, loff_t offset, int whence)                            | 更改设备当前的偏移量    |
| .unlocked_ioctl | static long zxcypdriver_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)                     | 通用接口，这里实现清空设备 |

### 3.4 实验调试

#### 3.4.1 实验步骤

- 1、编写驱动程序和 Makefile，参考 ppt
- 2、将该程序复制到 wsl 下的任意一个目录

我是先在内核文件夹的同级目录创建了一个目录 OS\_drive，在里面 touch 一个.c 文件，再用记事本打开，复制即可。

- 3、返回该内核根目录，输入指令：

```
make modules_install
```

结果如图：

```
root@silence:~/linux-5.9.9# make modules_install
DEPMOD 5.9.9silence2.0Yoga2021-standard
```

图 3-1 make modules\_install

4、然后还是在内核根目录，再创建一个软连接/usr/src/{你的内核名称} 连接到内核源码目录：

```
sudo ln -s {源码文件夹路径} /usr/src/{你的内核名称}
```

//下面是我执行的命令

```
sudo ln -s {../OS_drive} /usr/src/{5.9.9silence2.0Yoga2021-standard}
```

5、重新 make 驱动程序：进入源码所在文件夹输入 make 指令，结果如下图：

```
root@silence:~/OS_drive# make
make -C /lib/modules/5.9.9silence2.0Yoga2021-standard/build M=/root/OS_drive
make[1]: Entering directory '/root/linux-5.9.9'
 CC [M] /root/OS_drive/Silence_Drive.o
 MODPOST /root/OS_drive/Module.symvers
WARNING: modpost: missing MODULE_LICENSE() in /root/OS_drive/Silence_Drive.o
 CC [M] /root/OS_drive/Silence_Drive.mod.o
 LD [M] /root/OS_drive/Silence_Drive.ko
make[1]: Leaving directory '/root/linux-5.9.9'
root@silence:~/OS_drive# ls
Makefile Silence_Drive.c Silence_Drive.mod Silence_Drive.mod.o built-in.a
Module.symvers Silence_Drive.ko Silence_Drive.mod.c Silence_Drive.o modules.order
root@silence:~/OS_drive#
```

图 3-2 make

6、insmod 文件名.ko

//我是在内核根目录执行 insmod 指令的，不确定在文件所在目录 insmod 有没有用

```
insmod ../OS_drive/Silence_Drive.ko
```

7、至此设备驱动程序就加入成功了，如果要卸载模块的话，就在.ko 文件所在目录执行 rmmod \*.ko 命令就行：

```
rmmod *.ko
```

### 3.4.2实验调试及心得

安装好模块之后编写测试程序，用命令行传入参数，如果 args[1]=="read"，就执行读操作，args[2]为要读入的字符数；如果 args[1]=="write"就执行写操作，args[2]为要写入的字符串。

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

```
#include <stdlib.h>
```

```

#include <sys/types.h>
#include <sys/ioctl.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

#define MEM_CLEAR 1

//使用方式: ./Silence_test write hello
// ./Silence_test read 5

int main(int argc, char **argv) {
 int fd, start, num;
 char buf[1024];
 fd = open("./Silence_Drive/Silence_Drive.ko", O_RDWR);
 if (fd < 0) {
 printf("Open error!\n");
 return 0;
 }
 if (argc == 3 && strcmp(argv[1], "read", 4) == 0) {
 start = 0;
 num = atoi(argv[2]);
 lseek(fd, start, SEEK_SET);
 read(fd, buf, num);
 printf("Read: %s\n", buf);
 }
 else if (argc == 3 && strcmp(argv[1], "write", 5) == 0) {
 start = 0;
 lseek(fd, start, SEEK_CUR);
 write(fd, argv[2], strlen(argv[2]));
 printf("Write succeed!\n");
 }
 else if (argc == 3 && strcmp(argv[1], "ioctl", 5) == 0) {
 if (strcmp(argv[2], "clear", 5) == 0) {
 ioctl(fd, MEM_CLEAR, NULL);

```

```

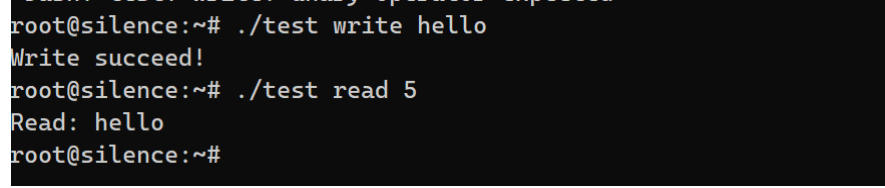
 printf("Clear success!\n");
 }

}

close(fd);
return 0;
}

```

执行情况如下：



```

root@silence:~# ./test write hello
Write succeed!
root@silence:~# ./test read 5
Read: hello
root@silence:~#

```

图 3-3 测试

心得：本次实验主要是添加设备驱动程序的过程，其中包括驱动程序的编译、加载、测试等过程。在这期间遇到了很多问题，如新编写的驱动程序不知道放在哪个目录下进行编译。一开始把编译得到的驱动程序放在了内核文件夹里面，想了想还是放在内核文件夹外更合理毕竟是模块法。另外给 WSL 增加驱动程序的过程和纯 linux 也不大相同，在一开始是有点茫然地，不过最后还是找到了正确的操作步骤。

## 附录 实验代码

### Makefile

```

ifneq ($(KERNELRELEASE),)
#kbuild syntax.

obj-m +=Silence_Drive.o
else
PWD :=$(shell pwd)
KVER :=$(shell uname -r)
KDIR ::=/lib/modules/$(KVER)/build
all:
$(MAKE) -C $(KDIR) M=$(PWD)
clean:
rm -f *.cmd *.o *.mod *.ko
endif

```

## Silence\_Drive.c

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

#define MEM_SIZE 0x4000 //缓冲区大小
#define GLOBAL_MAJOR 500 //设备号
#define MEM_CLEAR 1

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("A character driver module");

struct silence_dev {
 struct cdev cdev;
 unsigned char mem[MEM_SIZE];
};

static struct silence_dev *silence_devp;
static struct device *device;
static struct class *class;

static int driver_open(struct inode *inodep, struct file *filep) {
 printk(KERN_INFO "silence Driver: open\n");
 filep->private_data = silence_devp;
 return 0;
}

static int driver_release(struct inode *inodep, struct file *filep) {
 printk(KERN_INFO "silence Driver: release\n");
 return 0;
}

static ssize_t driver_read(struct file *filep, char __user *buf, size_t count, loff_t *offset)
{
 int ret = 0;
```

```

 printk(KERN_INFO "silence Driver: start read\n");

 size_t avail = MEM_SIZE - *offset; // 剩余可读字符数量
 struct silence_dev *dev = filep->private_data;

 // count 小于缓冲区剩余可读大小
 if (count <= avail) {
 if (copy_to_user(buf, dev->mem + *offset, count) != 0)
 return -EFAULT;
 *offset += count;
 ret = count;
 }
 // count 大于缓冲区剩余可读大小
 else {
 if (copy_to_user(buf, dev->mem + *offset, avail) != 0)
 return -EFAULT;
 *offset += avail;
 ret = avail;
 }

 printk(KERN_INFO "silence Driver: read %u bytes\n", ret);
 return ret;
}

static ssize_t driver_write(struct file *filep, const char __user *buf, size_t count, loff_t *offset)
{
 int ret = 0;
 printk(KERN_INFO "silence Driver: start write\n");

 size_t avail = MEM_SIZE - *offset;
 struct silence_dev *dev = filep->private_data;
 memset(dev->mem + *offset, 0, avail);
 printk(KERN_INFO "silence Driver: After write\n");

 if (count > avail) {
 if (copy_from_user(dev->mem + *offset, buf, avail) != 0)
 return -EFAULT;
 *offset += avail;
 ret = avail;
 }
 else {
 if (copy_from_user(dev->mem + *offset, buf, count) != 0)
 return -EFAULT;
 }
}

```



```

 *offset += count;
 ret = count;
 }
 printk(KERN_INFO "silence Driver: written %u bytes\n", ret);
 return ret;
}

```

//设定 write 或者 read 操作正确的起始下标

```
static loff_t driver_llseek(struct file *filep, loff_t offset, int whence)
```

```

{
 loff_t ret = 0;
 printk(KERN_INFO "silence Driver: start llseek\n");

```

```
 switch (whence) {
```

//当输入的偏移小于 0 或者大于缓冲区大小

case 0:

```

 if (offset < 0) {
 ret = -EINVAL;
 break;
 }

```

```

 if (offset > MEM_SIZE) {
 ret = -EINVAL;
 break;
 }

```

```

 ret = offset;
 break;

```

//输入的偏移大小合法

case 1:

//f\_pos 是文件当前位置

//判断当前位置+偏移是否越界

```

 if ((filep->f_pos + offset) > MEM_SIZE) {
 ret = -EINVAL;
 break;
 }

```

```

 if ((filep->f_pos + offset) < 0) {
 ret = -EINVAL;
 break;
 }

```

```

 ret = filep->f_pos + offset;
 break;

```

default:

```

 ret = -EINVAL;
 }

```

```

 if (ret < 0)
 return ret;

 printk(KERN_INFO "silence Driver: set offset to %u\n", ret);
 filep->f_pos = ret;
 return ret;
 }

//清除设备缓冲区
static long driver_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)
{
 struct silence_dev *dev = filep->private_data;
 printk(KERN_INFO "silence Driver: start memory clear\n");
 switch (cmd) {
 case MEM_CLEAR:
 memset(dev->mem, 0, MEM_SIZE);
 printk("silence Driver: memory is set to zero\n");
 break;
 default:
 return -EINVAL;
 }
 return 0;
}

static const struct file_operations fops = {
 .owner = THIS_MODULE,
 .open = driver_open,
 .release = driver_release,
 .read = driver_read,
 .write = driver_write,
 .llseek = driver_llseek,
 .unlocked_ioctl = driver_ioctl,
};

static int __init silencedriver_init(void) {
 int ret;
 dev_t devno = MKDEV(GLOBAL_MAJOR, 0);
 printk(KERN_INFO "Load module: silence_driver\n");
 //注册字符设备编号
 ret = register_chrdev_region(devno, 1, "silence_driver");
 if (ret < 0) {

```

```

 printk(KERN_ALERT "Registering the character device failed with %d\n", ret);
 return ret;
 }

 //参数一为要分配的块的大小
 silence_devp = kzalloc(sizeof(struct silence_dev), GFP_KERNEL);
 if (silence_devp == NULL) {
 printk(KERN_ALERT "Alloc memory for device failed\n");
 ret = -ENOMEM;
 goto failed;
 }
 //缓冲区初始化
 memset(silence_devp->mem, 0, MEM_SIZE);

 //静态初始化字符设备
 cdev_init(&silence_devp->cdev, &fops);
 silence_devp->cdev.owner = THIS_MODULE;
 //传入 cdev 结构的指针，起始设备编号，以及设备编号范围
 cdev_add(&silence_devp->cdev, devno, 1);

 /* Creat device file */
 class = class_create(THIS_MODULE, "silence_driver");
 if (IS_ERR(class)) {
 ret = PTR_ERR(class);
 printk(KERN_ALERT "Creat class for device file failed with %d\n", ret);
 goto failed;
 }
 device = device_create(class, NULL, devno, NULL, "silence_driver");
 if (IS_ERR(device)) {
 class_destroy(class);
 ret = PTR_ERR(device);
 printk(KERN_ALERT "Creat device file failed with %d\n", ret);
 goto failed;
 }

 return 0;

failed:
 unregister_chrdev_region(devno, 1);
 return ret;
}

static void __exit silencedriver_exit(void) {

```

```
printk(KERN_INFO "Unload module: silence_driver\n");

device_destroy(class, MKDEV(GLOBAL_MAJOR, 0));
class_unregister(class);
class_destroy(class);

cdev_del(&silence_devp->cdev);
kfree(silence_devp);
unregister_chrdev_region(MKDEV(GLOBAL_MAJOR, 0), 1);
}

module_init(silencedriver_init);
module_exit(silencedriver_exit);
```

## 4 实验四 /proc 文件分析

### 4.1 实验目的

了解/proc 文件的特点和使用方法

### 4.2 实验内容

使用 GTK/QT 实现系统监控器：

1. 了解/proc 文件的特点和使用方法；
2. 监控系统中进程运行情况；
3. 用图形界面实现系统资源的监控。

### 4.3 实验设计

#### 4.3.1 开发环境

WSL(Windows Subsystem for Linux)

OS:Ubuntu 20.04.1 LTS x86\_64

Kernel: 5.9.9silence2.0Yoga2021-standard

Shell: bash 5.0.17

宿主机：

系统：Windows10 家庭中文版，64 位操作系统，基于 x64 的

版本号： 2004

处理器： 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz

内存： 16.0GB( 15.8 GB 可用)

#### 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

## 4.4 实验调试

### 4.4.1 实验步骤

具体实现如下:

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);
}
```

图 4-1 构造函数

用于初始化 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));
 m_cpu_total__ = total;
 m_cpu_used__ = use;
 }
 }
}

```

图 4-2 刷新函数

用于实时更新窗口显示的信息

### 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!"), QMessageBox::Yes);
 return;
 }

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

 while (1)
 {
 tempStr = tempFile.readLine();
 pos = tempStr.indexOf("MemTotal");
 if (pos != -1)
 {
 memTotal = tempStr.mid(pos + 10, tempStr.length() - 12);

```

图 4-3 内存信息显示函数

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!"), QMessageBox::Ok);
 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];
```

图 4-4 基本配置信息函数

用于显示系统的基本配置信息

### 4.4.2 实验调试及心得

- (1) 本次实验其实基本都是将对应文件中的内容读入，再将他们放入缓冲区
  - (2) 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。
  - (3) 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
- 运行结果如下图：





图 4-5 系统信息展示



图 4-6 进程信息展示

## 附录 实验代码

```
1. main.cpp
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
 QApplication a(argc, argv);
 MainWindow w;
 w.show();

 return a.exec();
}
```

## 2. mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QFile>
#include <QMessageBox>
#include <QDir>
#include <QListWidget>
#include <QListWidgetItem>
#include <QStringList>
#include <QTimer>
#include <QDateTime>
#include <QProcess>
#include <QPainter>
#include <QStringList>
#include <QTableWidget>

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);
}

void MainWindow::Update()
{
 QProcess process;

 process.start("cat /proc at");
```

```

 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));

 m_cpu_total__ = total;
 m_cpu_used__ = use;
 }
 }
 }
}

void MainWindow::timer_update_currentTabInfo()
{
 QFile tempFile;
 QDateTime time;
 QString tempStr;
 int pos;
 ui->label_CurrentTime->setText(time.currentDateTime().toStrin
g("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!"), QMessageBox::Yes);
 return;
}
QString memTotal;
QString memFree;
QString memUsed;
int nMemTotal, nMemFree, nMemUsed;

while (1)
{
 tempStr = tempFile.readLine();
 pos = tempStr.indexOf("MemTotal");
 if (pos != -1)
 {
 memTotal = tempStr.mid(pos + 10, tempStr
.length() - 13);
 memTotal = memTotal.trimmed();
 nMemTotal = memTotal.toInt() / 1024;
 }
 else if (pos = tempStr.indexOf("MemFree"), pos
!= -1)
 {
 memFree = tempStr.mid(pos + 9, tempStr.l
ength() - 12);
 memFree = memFree.trimmed();
 nMemFree = memFree.toInt() / 1024;
 break;
 }
}

nMemUsed = nMemTotal - nMemFree;

memUsed = QString::number(nMemUsed, 10);
memFree = QString::number(nMemFree, 10);
memTotal = QString::number(nMemTotal, 10);

ui->progressBar_RAM->setValue(nMemUsed * 100 / nMemTotal);

int index = ui->tabWidget_INFO->currentIndex();
if (index == 0)


```

```

 {
 struct sysinfo info;
 sysinfo(&info);
 struct tm *ptm = nullptr;
 ptm = gmtime(&info.uptime);
 char time_buf[30];
 sprintf(time_buf, "%02d:%02d:%02d", ptm->tm_hour,
ptm->tm_min, ptm->tm_sec);
 ui->label_runningtime->setText(QString(time_buf));
 }

 tempFile.close(); //关闭内存信息文件
}

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  uinfo"); //打开 CPU 信
息文件

 if (!tempFile.open(QIODevice::ReadOnly))
 {
 QMessageBox::warning(this, tr("warning"), t
r("The cpuinfo file can not open!"), QMessageBox::Yes);
 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", p
tm->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));

 //循环读取文件内容，查找需要的信息
 while (1)
 {
 tempStr = tempFile.readLine();
 if (pos = tempStr.indexOf("model name"),
pos != -1)
 {
 pos += 12;
 QString *cpu_type = new QString(t
empStr.mid(pos, tempStr.length() - 12));
 ui->label_CPUType->setText(*cpu_type);
 break;
 }
 }
 tempFile.close();

 tempFile.setFileName("/proc 🍷ersion"); //说明正在
运行的内核版本
 if (!tempFile.open(QIODevice::ReadOnly))
 {
 QMessageBox::warning(this, tr("warning"), t
r("The version file can not open!"), QMessageBox::Yes);
 return;
 }
 tempStr = tempFile.readLine();

 int pos1 = tempStr.indexOf("(");
 QString *os_type = new QString(tempStr.mid(14, p
os1 - pos - 2));
 ui->label_SystemVersion->setText(*os_type);

 tempFile.close(); //关闭操作系统信息文件
 }

 else if (index == 1)
 { //ProgInfo

```

```

ui->listWidget_process->clear();
QDir qd("/proc");
QStringList qsList = qd.entryList();
QString qs = qsList.join("\n");
QString id_of_pro;
bool ok;
int find_start = 3;
int a, b;
int nProPid; //进程 PID
QString proName; //进程名
QString proState; //PPID
QString proPri; //进程优先级
QString proMem; //进程占用内存
QListWidgetItem *title = new QListWidgetItem("PID
\t" + QString::fromUtf8("名称") + "\t\t" +

 QString::fromUtf8("PPID
") + "\t" +

 QString::fromUtf8("优先
级") + "\t" +

 QString::fromUtf8("占用
内存"),

 ui->listWidget_process);
//循环读取进程
while (1)
{
 //获取进程 PID
 a = qs.indexOf("\n", find_start);
 b = qs.indexOf("\n", a + 1);
 find_start = b;
 id_of_pro = qs.mid(a + 1, b - a - 1);
 nProPid = id_of_pro.toInt(&ok, 10);
 if (!ok)
 {
 break;
 }

 //打开 PID 所对应的进程状态文件
 tempFile.setFileName("/proc/" + id_of_pro
+ "
at");

```



```

 if (!tempFile.open(QIODevice::ReadOnly))
 {
 QMessageBox::warning(this, tr("warni
ng"), tr("The pid stat file can not open!"), QMessageBox::Yes
);
 return;
 }
 tempStr = tempFile.readLine();
 if (tempStr.length() == 0)
 {
 break;
 }
 a = tempStr.indexOf("(");
 b = tempStr.indexOf(")");
 proName = tempStr.mid(a + 1, b - a -
1);

 proName.trimmed(); //删除两端的空格
 proState = tempStr.section(" ", 3, 3);
 proPri = tempStr.section(" ", 17, 17);
 proMem = tempStr.section(" ", 22, 22);

 if (proName.length() >= 13)
 {
 QListWidgetItem *item = new QList
WidgetItem(id_of_pro + "\t" +

 proName

+ "\t" +

 proState

+ "\t" +

 proPri +

 proMem,

 ui->listWidget_pr
ocess);
 }
 else
 {
 QListWidgetItem *item = new QList
WidgetItem(id_of_pro + "\t" +

```

```

+ "\t\t" +
proName

+ "\t" +
proState

"\t" +
proPri +

proMem,

ui->listWidget_pr
ocess);
 }
 tempFile.close();
}

tempFile.close(); //关闭该 PID 进程的状态文件
}

}

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

void MainWindow::on_tabWidget_INFO_currentChanged(int index)
{
 showinfo(index); //显示 tab 中的内容
 return;
}

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

```

### 3. mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

```

```

#include <QMainWindow>

```

```

#include <unistd.h>
#include <QTimer>
#include <sys/sysinfo.h>
#include <QObject>
#include <QProcess>
#include <QDebug>
#include <QWidget>
#include <QList>

namespace Ui {
 class MainWindow;
}

class MainWindow : public QMainWindow
{
 Q_OBJECT

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

private:
 Ui::MainWindow *ui;
 QList<float> yList;
 QList<float> yList1;

private slots:
 void showinfo(int index);
 void on_pushButton_shutdown_clicked();
 void on_tabWidget_INFO_currentChanged(int index);
 void timer_update_currentTabInfo();

public slots:
 void Update();

private:
 double usage;
 double m_cpu_total__ = 0;
 double m_cpu_used__ = 0;
};

#endif // MAINWINDOW_H

```

#### 4. mainwindow.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">

```

```

<class>MainWindow</class>
<widget class="QMainWindow" name="MainWindow">
 <property name="geometry">
 <rect>
 <x>0</x>
 <y>0</y>
 <width>403</width>
 <height>540</height>
 </rect>
 </property>
 <property name="windowTitle">

 <string> Alex PC Monitor< ring>
 </property>
 <widget class="QWidget" name="centralWidget">
 <widget class="QTabWidget" name="tabWidget_INFO">
 <property name="geometry">
 <rect>
 <x>10</x>
 <y>0</y>
 <width>381</width>
 <height>291</height>
 </rect>
 </property>
 <property name="currentIndex">
 <number>0</number>
 </property>
 <widget class="QWidget" name="tab">
 <attribute name="title">

 <string>系统信息< ring>
 </attribute>
 <widget class="QFrame" name="frame_2">
 <property name="geometry">
 <rect>
 <x>10</x>
 <y>130</y>
 <width>361</width>
 <height>121</height>
 </rect>
 </property>
 <property name="frameShape">
 <enum>QFrame::StyledPanel</enum>
 </property>

```

```

 <property name="frameShadow">
 <enum>QFrame::Raised</enum>
 </property>
 <widget class="QLabel" name="label_7">
 <property name="geometry">
 <rect>
 <x>20</x>
 <y>30</y>
 <width>111</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

 <string>系统版本号: < ring>
 </property>
 </widget>
 <widget class="QLabel" name="label_8">
 <property name="geometry">
 <rect>
 <x>20</x>
 <y>81</y>
 <width>91</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

 <string>CPU 型号: < ring>
 </property>
 </widget>
 <widget class="QLabel" name="label_SystemVersion">
 <property name="geometry">
 <rect>
 <x>130</x>
 <y>20</y>
 <width>341</width>
 <height>41</height>
 </rect>
 </property>
 <property name="text">

 <string>TextLabel< ring>
 </property>

```

```

</widget>
<widget class="QLabel" name="label_CPUType">
 <property name="geometry">
 <rect>
 <x>90</x>
 <y>79</y>
 <width>341</width>
 <height>41</height>
 </rect>
 </property>
 <property name="font">

 <pointsize>9</pointsize>

 </property>
 <property name="text">

 <string>TextLabel< ring>
 </property>
</widget>
</widget>
<widget class="QFrame" name="frame_5">
 <property name="geometry">
 <rect>
 <x>10</x>
 <y>20</y>
 <width>361</width>
 <height>111</height>
 </rect>
 </property>
 <property name="frameShape">
 <enum>QFrame::StyledPanel</enum>
 </property>
 <property name="frameShadow">
 <enum>QFrame::Raised</enum>
 </property>
 <widget class="QLabel" name="label_4">
 <property name="geometry">
 <rect>
 <x>20</x>
 <y>40</y>
 <width>81</width>
 <height>21</height>
 </rect>
 </property>
 </widget>
</property>
</widget>

```

```

</property>
<property name="text">

 <string>启动时间: < ring>
</property>
</widget>
<widget class="QLabel" name="label_5">
 <property name="geometry">
 <rect>
 <x>20</x>
 <y>70</y>
 <width>101</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

 <string>运行时长: < ring>
 </property>
</widget>
<widget class="QLabel" name="label_6">
 <property name="geometry">
 <rect>
 <x>20</x>
 <y>10</y>
 <width>71</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

 <string>主机名: < ring>
 </property>
</widget>
<widget class="QLabel" name="label_hostname">
 <property name="geometry">
 <rect>
 <x>130</x>
 <y>10</y>
 <width>341</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

```

```

 <string>AlexFanLinux< ring>
 </property>
</widget>
<widget class="QLabel" name="label_boottime">
 <property name="geometry">
 <rect>
 <x>130</x>
 <y>40</y>
 <width>341</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

 <string>2020.3.1 18:30:10< ring>
 </property>
</widget>
<widget class="QLabel" name="label_runningtime">
 <property name="geometry">
 <rect>
 <x>130</x>
 <y>70</y>
 <width>341</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

 <string>00: 00< ring>
 </property>
</widget>
</widget>
<widget class="QWidget" name="tab_1">
 <attribute name="title">

 <string>进程信息< ring>
 </attribute>
 <widget class="QListWidget" name="listWidget_process">
 <property name="geometry">
 <rect>
 <x>10</x>
 <y>20</y>
 </rect>
 </property>
 </widget>
</widget>

```



```

 <width>391</width>
 <height>281</height>
 </rect>
</property>
</widget>
</widget>
</widget>
<widget class="QLabel" name="label_9">
 <property name="geometry">
 <rect>
 <x>20</x>
 <y>450</y>
 <width>91</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

 <string>当前时间: < ring>
 </property>
</widget>
<widget class="QLabel" name="label_10">
 <property name="geometry">
 <rect>
 <x>20</x>
 <y>315</y>
 <width>91</width>
 <height>16</height>
 </rect>
 </property>
 <property name="text">

 <string>CPU 使用率: < ring>
 </property>
</widget>
<widget class="QLabel" name="label_11">
 <property name="geometry">
 <rect>
 <x>20</x>
 <y>360</y>
 <width>91</width>
 <height>16</height>
 </rect>
 </property>

```

```

 <property name="text">

 <string>内存使用率: < ring>
 </property>
</widget>
<widget class="QPushButton" name="pushButton_shutdown">
 <property name="geometry">
 <rect>
 <x>269</x>
 <y>420</y>
 <width>81</width>
 <height>81</height>
 </rect>
 </property>
 <property name="font">

 <pointsize>19</pointsize>

 </property>
 <property name="text">

 <string>关 机< ring>
 </property>
</widget>
<widget class="QLabel" name="label_CurrentTime">
 <property name="geometry">
 <rect>
 <x>90</x>
 <y>450</y>
 <width>341</width>
 <height>20</height>
 </rect>
 </property>
 <property name="text">

 <string>Time< ring>
 </property>
</widget>
<widget class="QProgressBar" name="progressBar_RAM">
 <property name="geometry">
 <rect>
 <x>140</x>
 <y>360</y>
 <width>211</width>

```

```

 <height>21</height>
 </rect>
</property>
<property name="value">
 <number>24</number>
</property>
</widget>
<widget class="QLabel" name="cpu_rate">
 <property name="geometry">
 <rect>
 <x>140</x>
 <y>315</y>
 <width>191</width>
 <height>21</height>
 </rect>
 </property>
 <property name="text">

 <string>10< ring>
 </property>
</widget>
<widget class="QLabel" name="label">
 <property name="geometry">
 <rect>
 <x>240</x>
 <y>310</y>
 <width>31</width>
 <height>31</height>
 </rect>
 </property>
 <property name="text">

 <string>%< ring>
 </property>
 <property name="textFormat">
 <enum>Qt::PlainText</enum>
 </property>
</widget>
<zorder>label_9</zorder>
<zorder>label_10</zorder>
<zorder>label_11</zorder>
<zorder>pushButton_shutdown</zorder>
<zorder>label_CurrentTime</zorder>
<zorder>progressBar_RAM</zorder>

```

```
 <zorder>tabWidget_INF0</zorder>
 <zorder>cpu_rate</zorder>
 <zorder>label</zorder>
 </widget>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>
```