

**课 程 实 验 报 告**

**课程名称： 操作系统原理**

**专业班级： CS1503班**

**学 号： U201514559**

**姓 名： 周铭昊**

**指导教师： 郑然**

**报告日期： 2018年3月31日**

**计算机科学与技术学院**

**目录**

[实验一 熟悉Linux下的编程环境 1](#_Toc510292748)

[**1.1实验目的** 1](#_Toc510292749)

[**1.2 实验内容** 1](#_Toc510292750)

[**1.3 实验过程** 1](#_Toc510292751)

[1.3.1 编程思路 2](#_Toc510292752)

[1.3.2 遇到的问题及解决方式 3](#_Toc510292753)

[1.3.3 实验测试与结果分析 3](#_Toc510292754)

[**1.4 实验总结** 4](#_Toc510292755)

[实验二 掌握添加系统调用的方法 5](#_Toc510292756)

[**2.1 实验目的** 5](#_Toc510292757)

[**2.2 实验内容** 5](#_Toc510292758)

[**2.3 实验过程** 5](#_Toc510292759)

[2.3.1 编程思路 7](#_Toc510292760)

[2.3.2 遇到的问题及解决方式 8](#_Toc510292761)

[2.3.3 实验测试与结果分析 8](#_Toc510292762)

[**2.4 实验总结** 9](#_Toc510292763)

[实验三 增加设备驱动程序 10](#_Toc510292764)

[**3.1 实验目的** 10](#_Toc510292765)

[**3.2 实验内容** 10](#_Toc510292766)

[**3.3 实验过程** 10](#_Toc510292767)

[3.3.1 编程思路 10](#_Toc510292768)

[3.3.2 遇到的问题及解决方式 11](#_Toc510292769)

[3.3.3 实验测试与结果分析 11](#_Toc510292770)

[**3.4 实验总结** 11](#_Toc510292771)

[实验四 使用QT实现系统监控器 12](#_Toc510292772)

[**4.1 实验目的** 12](#_Toc510292773)

[**4.2 实验内容** 12](#_Toc510292774)

[**4.3 实验过程** 12](#_Toc510292775)

[4.3.1 编程思路 13](#_Toc510292776)

[4.3.2 遇到的问题及解决方式 14](#_Toc510292777)

[4.3.3 实验测试与结果分析 15](#_Toc510292778)

[**4.4 实验总结** 19](#_Toc510292779)

# 实验一 熟悉Linux下的编程环境

## **1.1实验目的**

1、掌握Linux操作系统的使用方法，包括键盘命令、系统调用；

2、熟悉Linux下的编程环境。

## **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(使用系统调用open/read/write...）；

2．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)。 如三个进程誊抄演示。

## **1.3 实验过程**

1. 首先安装编程环境，从ubuntu官网下载ubuntu16.04.3系统镜像，通过虚拟机软件VMware安装虚拟机。在ubuntu虚拟机下编写一个c程序，用系统调用(open/writed等)而不是库函数(fwrite/fopen等)来实现文件誊抄，编写完成后用ctrl+alt+t快捷键打开终端，进入主目录/home/lumos，使用gcc -o proj1\_copy proj1\_copy.c命令来进行编译，成功生成proj1\_copy执行文件，在终端中用./proj1\_copy命令，执行结果如图1.1所示，成功生成FileWrite文件。

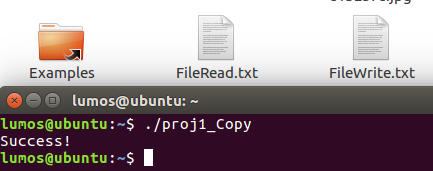


图1.1 拷贝程序执行结果

2. 第二个实验，先安装gtk+3.0的编程环境，通过以下四步完成：

(1)、安装gcc/g++/gdb/make等基本编程工具：sudo apt-get install build-essential

(2)、安装GTK+3.0：sudo apt-get install libgtk-3-dev

(3)、安装pkg-config：sudo apt-get install pkg-config

(4)、安装帮助文件，方便查看帮助：sudo apt-get install devhelp

安装完成后，确认一下GTK+版本 pkg-config –modversion gtk+-3.0，结果显示GTK+版本为3.18.9。

随后编写三个并发进程的程序：一个程序显示当前时间，一个显示从1加到100的累加求和，一个显示从100到1的倒计时，刷新周期都为1秒。这里把累加求和作为主进程，另外两个窗口作为子进程。

编写完成后通过指令gcc `pkg-config --cflags gtk+-3.0` -o proj1\_Three proj1\_Three.c `pkg-config --libs gtk+-3.0` 进行编译，编译后成功生成proj1\_Three执行文件。

### 1.3.1 编程思路

1. 文件拷贝程序：通过系统调用open("/home/lumos/FileRead.txt",O\_RDONLY);用只读方式打开FileRead文件，再次使用open("/home/lumos/FileWrite.txt",O\_WRONLY|O\_CREAT|O\_TRUNC,S\_IRUSR|S\_IWUSR) 用读写方式打开一个文件FileWrite，O\_TRUNC参数的作用是当文件FileWrite已经存在时，把这个文件清空，再进行写入操作，S\_IRUSR|S\_IWUSR则是文件权限的管理。打开两个文件之后，在一个循环中，通过read(r\_fp,&buffer,sizeof(char)); 把文件FileRead读取到缓冲区buffer中，逐个字符读取，然后逐个字符地写入文件FileWrite中，直到读取完成后跳出循环。

2. 三个窗口显示三个并发进程：首先通过fork()创建两个子进程，在每个子进程内并行调用对应的窗口初始化函数。三个进程的窗口初始化程序类似，以累计求和窗口的初始化函数init\_add为例：

先进行环境初始化gtk\_init(&argc,&argv); 在程序使用到GTK+工具库之前，必须对它进行初始化；然后通过gtk\_window\_new(GTK\_WINDOW\_TOPLEVEL) 创建一个窗口；再对这个窗口的属性进行设置，如标题、窗口宽度、窗口大小；这些窗口的基本属性设置完成后把关闭窗口信号绑定到一个关闭事件，当发出关闭信号时，进入回调函数"destroy"，使程序结束；然后把一个标签绑定到这个窗口上，用于显示对应信息，注意这个标签一定要使用label=gtk\_label\_new(NULL)进行实例化，通过g\_timeout\_add(1000,getAdd,(void \*)label)来控制每隔1000ms，即一秒钟刷新一次数据，最后显示所有窗口，并用gtk\_main()作为主循环，至此窗口程序结束。

三个窗口区别主要在于获取的数据不同：getAdd函数中使用一个全局变量count进行累加，并把这个数据写入到字符缓冲buf中，然后通过gtk\_label\_set\_markup(GTK\_LABEL(data), buf)把缓冲区里的数据写入到标签里；倒计时程序与累加求和类似，不赘述；getTime函数用于获取系统当前时间，用到两个结构体time\_t和tm，前者用来存储从1970年到现在经过了多少秒，后者存储的是年月日这样的具体时间结构，通过stuct tm\* localtime(const time\_t \*timep) 函数获取经过时区转换的当前时间，再通过gchar \*time\_dis = g\_strdup\_printf("%2d:%2d:%2d",(time\_tm->tm\_hour), (time\_tm->tm\_min), (time\_tm->tm\_sec))来把需要用到的时间信息输出到字符缓冲区中，最后把字符缓冲区的输入到标签里，即完成了把当前时间信息显示在标签上的功能。

### 1.3.2 遇到的问题及解决方式

1. 文件拷贝程序：最初在打开目标文件时，使用的是open("/home/lumos/FileWrite.txt",O\_WRONLY|O\_CREAT|O\_TRUNC); 并没有声明文件的权限，导致生成的文件显示成加锁的模式，无法访问。解决方法是加上两个参数控制访存权限。S\_IRUSR的功能是允许文件的所有者阅读它，S\_IWUSR的功能是允许文件所有者写它。

2. 三个并发进程窗口程序：最初使用gcc -o proj1\_Three proj1\_Three.c `pkg-config --libs gtk+-3.0`指令进行编译，结果显示error，不能成功生成执行文件，编译错误如图1.2所示，后来查询相关资料，改用gcc `pkg-config --cflags gtk+-3.0` -o proj1\_Three proj1\_Three.c `pkg-config --libs gtk+-3.0` 进行编译，可以成功生成执行文件。

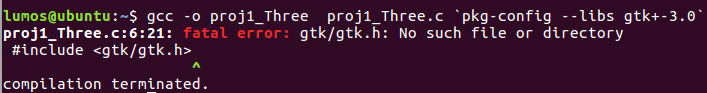


图1.2 新版GTK3编译错误

### 1.3.3 实验测试与结果分析

1. 文件拷贝程序：进行两项测试

(1).目标文件不存在时，执行拷贝程序后，观察能否生成正确的目标文件，要求内容与原文件相同。

先把FileWrite文件删除，执行一次拷贝程序，如图1.3所示，成功生成FileWrite文件，且内容与FileRead文件内容一致，程序正确运行。

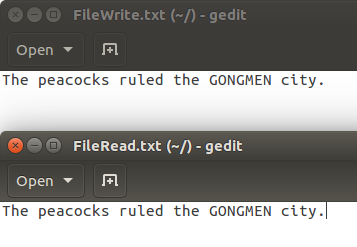


图1.3 文件拷贝程序测试1

(2). 目标文件已经存在时，在其中输入与源文件中不同的数据，观察执行拷贝程序后目标文件的内容。

修改FileWrite文件内容为aaaaaa，执行一次拷贝程序，如图1.4所示， FileWrite文件内容与FileRead文件内容一致，程序正确运行。

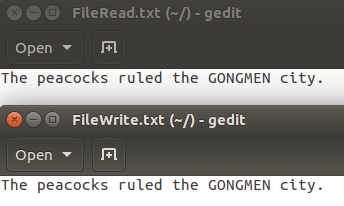


图1.4 文件拷贝程序测试2

2. 三个窗口显示三个并发进程：直接执行程序，程序的执行结果如图1.5所示，程序正确运行。

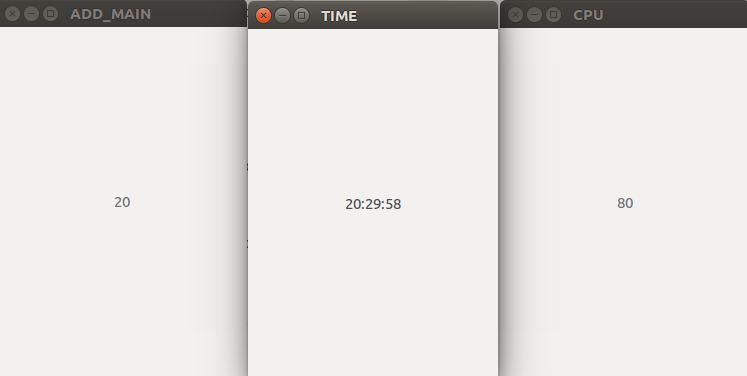


图1.5 三个窗口显示三个并发进程测试

## **1.4 实验总结**

通过本次实验，熟悉了linux下的终端命令操作，学会了GTK的安装方法和基本用法，学会如何通过系统调用控制文件的读写及权限管理，了解了time\_t和tm两种时间结构体的内容，学会了如何获取当前时间，多进程的窗口直观显示也让我对并行进程有了更深的理解。

# 实验二 掌握添加系统调用的方法

## **2.1 实验目的**

1、掌握系统调用的实现过程；

2、通过编译内核方法，增加一个新的系统调用；

3、另编写一个应用程序，使用新增加的系统调用。

## **2.2 实验内容**

1. 内核编译、生成，用新内核启动；

2. 新增系统调用实现：文件拷贝或P、V操作。

## **2.3 实验过程**

1. 在[https://www.kernel.org/](https://www.kernel.org/。网站下载linux4.24.16) 网站下载linux-4.24.16.tar.xz内核代码，进入linux系统，通过su指令进入管理员模式，把系统内核压缩包拷贝到目录/usr/src下，然后使用xz -d linux-4.24.16.tar.xz和tar -xvf linux4.24.16.tar两条命令解压缩，安装一些基本的工具软件之后，进入linux-4.24.16目录下，依次执行以下三条命令：

sudo make mrproper 作用是清楚编译过程中产生的所有中间文件；

sudo make clean 作用是清除上一次产生的编译中间文件；

sudo make menuconfig 出现选择的图形化界面之后，直接选择exit退出，弹出的提示里选择save保存，实现内核的默认配置。

随后执行sudo make -j4进行四线程编译，大概两小时之后，编译完成，在终端输入sudo make modules\_install来安装内核模块，安装完成后再输入sudo make install来安装内核。内核安装结束之后就重新启动系统，在虚拟机出现vmware字样时就要迅速的按住shift键，进入内核选择加载界面，如图2.1所示，选择linux4.24.16进入系统，就完成了使用新内核启动的功能。



图2.1 用新内核启动

2. 新增系统调用，实现文件拷贝功能。首先进入root管理者模式，进入到目录/usr/src下，用gedit linux-4.24.16/kernel/sys.c命令打开，在该文件末尾加上自己编写的系统调用函数sys\_mycopy。编程思路与实验一基本一致，不同的是在内核中，进行系统调用的文件操作时如果要访问用户空间，则需要使用get\_fs, set\_fs对他们进行保护。编写完成后如图2.2所示，保存退出。

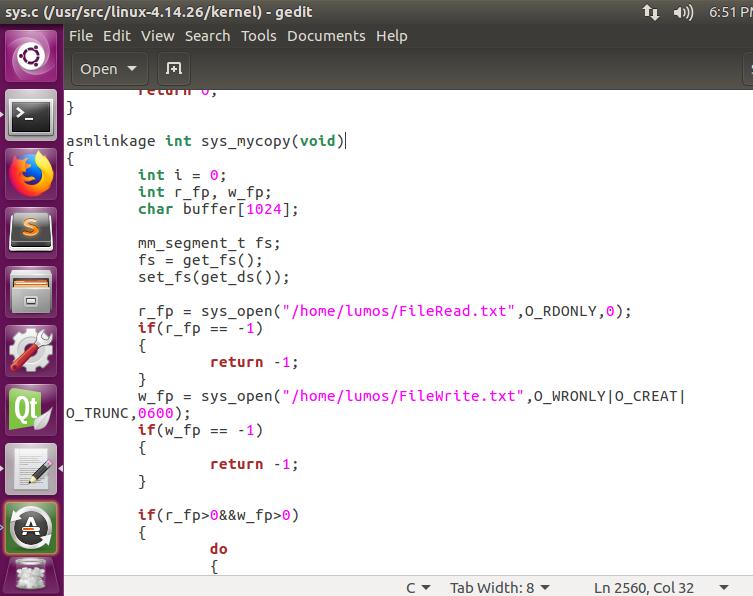


图2.2 文件拷贝系统调用函数

再执行 gedit linux-4.24.16/arch/x86/include/asm/syscalls.h ，添加系统调用函数声明：asmlinkage int sys\_mycopy(void)。如图2.3所示，保存退出。

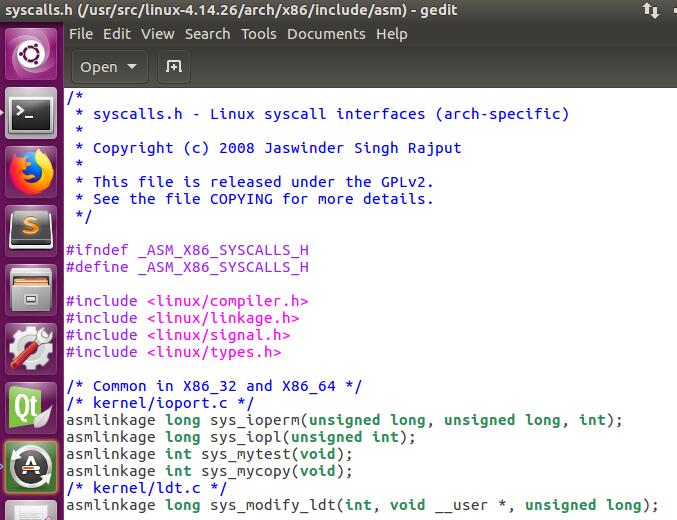


图2.2 添加系统调用函数声明

然后再添加系统调用号，打开linux-4.11.6/arch/x86/entry/syscalls/syscall\_64.tbl，然后根据格式添加，如图2.3所示，系统调用号为334，保存退出。

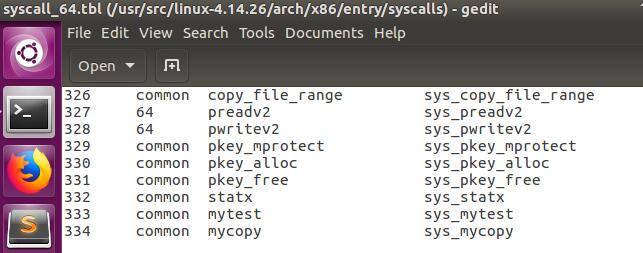


图2.3 添加系统调用号

最后按照实验一的相应步骤，再次编译内核，编译完成后安装内核到系统中，重新启动时选择新内核启动即可。启动后，编写测试程序，在测试程序中使用该系统调用，编译执行测试程序，观察实验结果并记录分析。

### 2.3.1 编程思路

编程思路与实验一基本一致，只是系统调用会要求使用的缓冲区不能在内核区，因此在读写文件前，执行两条语句：

mm\_segment\_t fs;

fs = get\_fs();

得到当前fs，然后使用set\_fs(get\_ds())设置fs为内核，get\_ds的作用是获得kernel的内存访问地址范围。读写结束之后再使用set\_fs(fs)恢复为原先fs。这样就可以在内核中使用系统调用sys\_open和sys\_write了。

测试文件的编写只需要使用#define MYCOPY 334定义一下系统调用号，再使用syscall(MYCOPY)调用自己编写的内核函数即可。

### 2.3.2 遇到的问题及解决方式

1. 内核函数中使用读写文件的系统调用时使用了open，write，编译内核时报错。

解决方式：应该使用sys\_open和sys\_write。

2. 最初调用sys\_call时参数设置按照实验一的方式只写了两个，编译报错。

解决方式：sys\_open的参数应当依照函数定义设置成3个sys\_open(const char \_\_user \*filename, int flags, int mode)

3. 添加系统调用函数声明时打开的是syscalls.h而不是syscall.h，两个文件名十分相似。

### 2.3.3 实验测试与结果分析

(1).目标文件不存在时，执行测试程序后，观察能否生成正确的目标文件，要求内容与原文件相同。

先把FileWrite文件删除，执行一次测试程序，如图2.4所示，成功生成FileWrite文件，且内容与FileRead文件内容一致，程序正确运行。

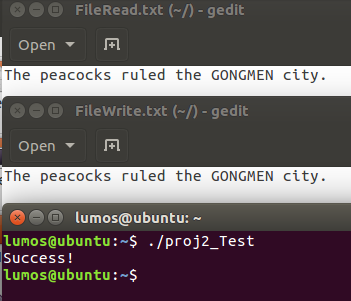


图2.4 文件拷贝系统调用测试1

(2). 目标文件已经存在时，在其中输入与源文件中不同的数据，观察测试程序后目标文件的内容。

修改FileWrite文件内容为aaaaaa，执行一次测试程序，如图2.5所示， FileWrite文件内容与FileRead文件内容一致，程序正确运行。

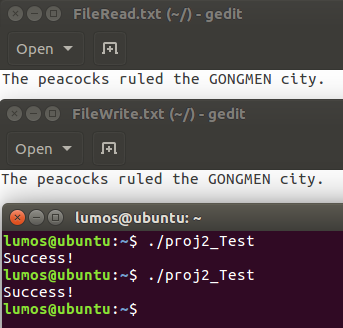


图2.5 文件拷贝系统调用测试2

## **2.4 实验总结**

通过本次实验，我学会了使用新内核启动linux操作系统，学会了自己编写系统调用函数的方法，对linux操作系统内核的理解更加深刻，熟悉了内核中系统调用、系统调用声明、系统调用号存储的文件位置，对用户态和内核态中函数的区别也有了更深的了解。

# 实验三 增加设备驱动程序

## **3.1 实验目的**

1、掌握增加设备驱动程序的方法；

2、通过模块方法，增加一个新的设备驱动程序，其功能可以简单。

## **3.2 实验内容**

实现字符设备的驱动，演示简单字符键盘缓冲区或一个内核单缓冲区。

要求：输入一个字符串后，在输出的字符串前加上自己的名字缩写。

## **3.3 实验过程**

编写一个简单的字符设备驱动程序，功能为输入一个字符串到驱动设备，然后从驱动设备输出一个这个字符串，并在该前面插入自己的名字缩写“zmh”。编写完成后，进入root模式，在对应目录下输入make命令执行Makefile，生成一些中间文件，然后执行insmod s\_device.ko把自己编写的驱动程序加载进内核中，然后使用lsmod命令查看是否加载成功，如图3.1所示，加载成功。

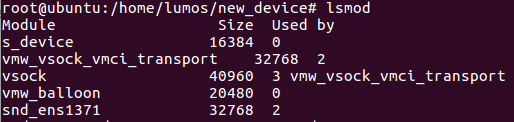


图3.1 lsmod结果

驱动程序成功加载之后，编写测试程序，输入一个字符串到设备驱动中，然后从设备驱动中读取一个字符串输出到终端界面。编译执行该测试程序，观察记录并分析测试结果。

### 3.3.1 编程思路

首先初始化注册设备，选择由系统分配主设备号，这样可以避免设备号冲突， register\_chrdev(0, "Mydev", &Myfop)函数返回0时代表设备初始化成功。然后实现设备操作，static int Myopen(struct inode \*, struct file \*)函数的功能是开启驱动设备，具体通过try\_module\_get系统调用实现。static int Myrelease(struct inode \*, struct file \*)函数的功能是释放设备文件，具体通过module\_put系统调用实现。Myread函数实现从驱动设备中读出数据的功能，通过系统调用copy\_to\_user把内核缓冲区的内容读出。Mywrite函数实现把数据写入驱动程序的功能，首先把内核缓冲区中的前三个字符赋值为“zmh”，然后逐个字符把函数输入的字符串赋值给缓冲区。

测试程序中，先使用open("/dev/s\_device",O\_RDWR, S\_IRUSR|S\_IWUSR)打开设备驱动，打开成功后输入字符串到缓冲区buf1中，然后通过write系统调用把这个字符串写入到到字符设备中，写入成功后使用read系统调用从字符设备中读出到缓冲区buf2，然后把buf2输出到屏幕上。

### 3.3.2 遇到的问题及解决方式

使用老师提供的Makefile文件，在执行make语句时总是报错。

解决方法：把“mymodule-objs :=test.o//模块的文件组成”这一行删除，或者直接采用别的makefile文件。

### 3.3.3 实验测试与结果分析

测试输入“hello”，应当输出“zmhhello”，程序运行结果如图3.2所示，结果正确。

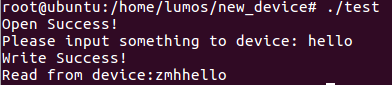


图3.2 输入hello结果

## **3.4 实验总结**

通过本次实验，我了解了linux系统字符驱动设备的具体结构和实现方法，学会了使用printk进行调试，对用户空间和内核空间的数据交换方式有了进一步的理解。

# 实验四 使用QT实现系统监控器

## **4.1 实验目的**

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。

## **4.2 实验内容**

使用GTK/QT实现系统监控器

(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)同过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，优先级等等

(8)cpu使用率的图形化显示(2分钟内的历史纪录曲线)

(9)内存和交换分区(swap)使用率的图形化显示(2分钟内的历史纪录曲线)

(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

----参照WINDOWS的任务管理器，实现其中的几个功能。

## **4.3 实验过程**

选择使用Qt进行本次实验。首先在linux下安装qt5.5，在qt官网下载对应linux系统的安装包，选择mingw带编译器版本，下载完成后进入ubuntu系统解压安装，然后在qt中创建一个TaskManager工程用来编写本次实验内容。编写完成后，要静态编译生成可执行文件。首先配置qmake，通过命令sudo apt-get install build-essential libpcap0.8-dev libx11-dev libfreetype6-dev libavahi-gobject-dev libSM-dev libXrender-dev libfontconfig-dev libXext-dev配置需要的安装包，然后通过sudo ./configure -static -release -qt-zlib -qt-libpng -qt-libmng -qt-libjpeg -nomake demos -nomake examples -qt-sql-sqlite -prefix /usr/local/Trolltech/Qt-5.5.0\_static配置config，配置结束后输入sudo make进行编译，大约一个小时之后编译完成，在终端输入make install 安装，安装完成后配置环境变量。通过sudo gedit /home/lumos/.profile修改这个文件，在文件最后加上以下内容：

QTDIR=/usr/local/Trolltech/Qt-5.5.0\_static/

PATH=$QTDIR/bin:$PATH

MANPATH=$QTDIR/doc/man:$MANPATH

LD\_LIBRARY\_PATH=$QTDIR/lib:$LD\_LIBRARY\_PATH

export QTDIR PATH MANPATH LD\_LIBRARY\_PATH

保存退出后重新启动ubuntu，进入TaskManager文件夹，修改.pro文件，在最后加上CONFIG+=static，然后执行qmake命令生成Makefile文件，再修改Makefile文件内容，在CXXflags选项后面加上 -static。最后执行make命令，编译成功后生成可执行文件TaskManager，然后在命令行./ TaskManager即可打开该程序，进行测试，观察记录并分析测试结果。

### 4.3.1 编程思路

首先根据功能需求，绘制ui界面，在主界面上添加四个label作为状态栏，分别显示当前时间、CPU使用率、已用内存空间、可用内存空间。然后添加一个tabWidget，加入4个选项卡：系统、进程、性能、关于。

在系统选项卡下，获取并显示主机名，按行读取/proc/sys/kernel/hostname文件，把读取到的字符串写入str1，然后使用str1.simplified()把这个字符串两边的空格和换行符去掉，得到主机名，显示在对应label上。然后获取并显示系统启动时间，同样的方式读取文件/proc/uptime下的内容，转化为float型后，除以86400得到已运行的天数，除去天数部分，剩下的时间除以3600得到已经运行的小时数，再除去这部分除以60得到已经运行的分钟数，最后减去之前计算过的时间即可得到已经运行的秒数。系统的空闲时间用相同方式得到，然后绑定到label上显示出来。再打开/proc/sys/kernel/ostype得到系统名，打开/proc/sys/kernel/osrelease得到系统版本号，两个字符串连接到一起输出到label显示出来。最后打开/proc/cpuinfo文件显示cpu的型号和主频。

在进程选项卡下，加入一个listwidget用于显示所有进程信息，首先设置目录项QDir my\_dir("/proc")，然后通过QStringList p\_List = my\_dir.entryList()把所有的pid读取到p\_List中，并且插入分节符。然后设置一行目录，分别为"NAME"、"PID"、"PPID"、"STAT"、"MEM"、"优先级"，然后循环读取pid，再打开"/proc/" + pid + "/stat"下的文件得到该进程信息，进行一些字符串处理得到需要显示的进程信息，加一个判断条件，只显示进程状态为sleep的进程，当p\_List中所有pid被读取完成后退出循环。还实现了搜索和关闭进程的功能，搜索就是把输入的字符串与pid或者进程名进行对比，如果是则搜索完成，关闭进程即获取当前在listwidget中选中的行，获取pid，然后执行system(kill pid)命令。

在性能选项卡下，要显示cpu利用率曲线、内存使用率曲线、交换分区曲线。要计算cpu的利用率，采样两个间隔50ms的时间取样，分别把两次的所有cpu使用情况求和得到s1，s2，把s2-s1得到这个时间间隔内的总cpu时间totalCpuTime，然后计算两次cpu使用情况的idle时间差idleTime，计算cpu使用率的公式为cpu\_rate = 100\* (totalCpuTime-idleTime)/totalCpuTime。计算内存和交换分区的使用率，读取/proc/meminfo下的信息，把已用的空间大小除以总大小即可。画图部分，首先QPainter painter(ui->tabWidget\_2)新建一个painter并且父窗口为显示区域(即性能选项卡)，然后绘制一个xy坐标系，坐标系的绘制参考了csdn上的教程，用一个数组a储存要显示的数字，然后计算平均值和最大最小值并绘制，最后绘制刻度线，其实就是一段一段的小直线。通过选项卡的tab值不同选择不同的值赋给数组a，就完成了显示不同曲线的功能。

由于获取cpu利用率时时间间隔取了50ms，因此要绘制20个点的话需要一定的时间计算，所以绘制之前先用数组info储存要显示的值，存满20个值后，开始显示（在此之前在坐标系上显示一张图“绘制中，请稍后”，时间通过timer2控制），然后就没过100ms刷新一次info的值。

所有数据刷新间隔均为100ms，除了进程信息，因为进程里有搜索功能，如果搜索到对应结果，刷新后会清空搜索记录，所以进程信息需要手动按下Reflesh按钮进行刷新，切换到其他选项卡再切换回来也可以刷新。

最后实现关机功能，在主窗口新增一个按钮，按钮槽函数中，设置一个Qmessagebox, 可以选择是否确认关机，确认关机后，执行system("/sbin/init 0")命令即可关机。

### 4.3.2 遇到的问题及解决方式

获取时间时，如果使用str2.toInt()则无法得到具体时间，解决方法是使用str2.toFloat()即可得到字符串对应的具体数字。

绘图时如果把计算到的cpu利用率直接赋给painter下的数组，会造成一次只能更新一个数字，所以坐标轴中只有一个点时正确的，其他点都是0。

解决方法是用一个数组info来保存20个点的cpu利用率，然后一次性赋值给绘图数据数组a，这样就可以绘制出完整的图像。

### 4.3.3 实验测试与结果分析

1. 系统功能测试，观察能否正确显示系统信息和状态栏信息。结果如图4.1所示，程序结果正确。



图4.1 系统功能测试结果

2. 进程功能测试，观察能否显示所有的Sleep状态下的进程信息，结果如图4.2所示，程序运行正确。

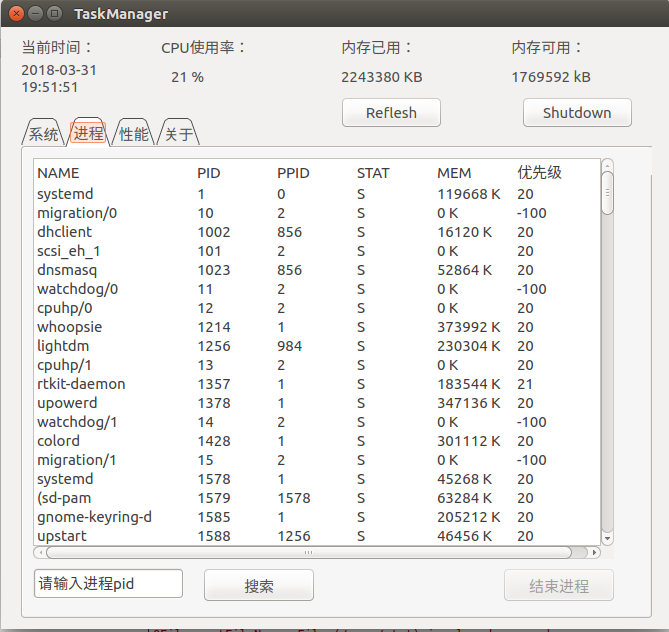


图4.2 进程功能测试结果

3. 进程搜索和结束进程测试，首先打开sublime程序，在进程搜索框内输入sublime\_text，点击搜索按钮，结果如图4.3所示，程序运行正确。

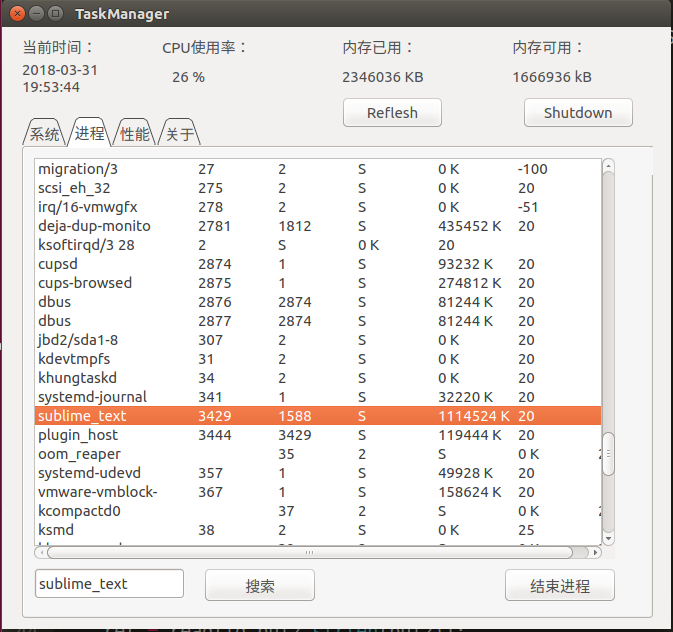


图4.3 搜索进程功能结果

然后点击结束进程按钮，sublime软件成功被关闭，弹出提示killed，如图4.4所示，结束进程功能正确。

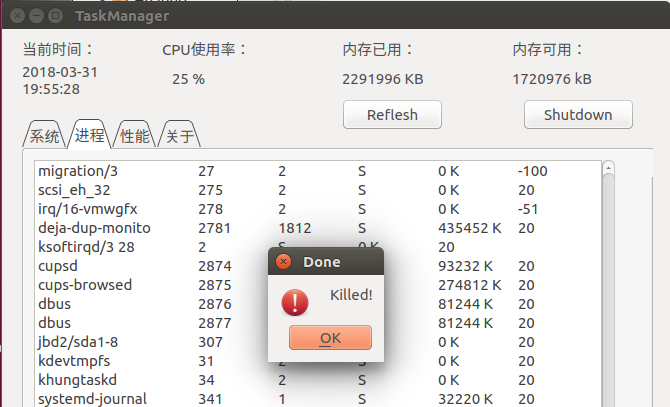


图4.4 结束进程功能结果

4． CPU利用率曲线测试，选择CPU选项卡，得到结果如图4.5所示，是一个实时变化的曲线，8s刷新一次（因为要采集20个点），程序正确运行。

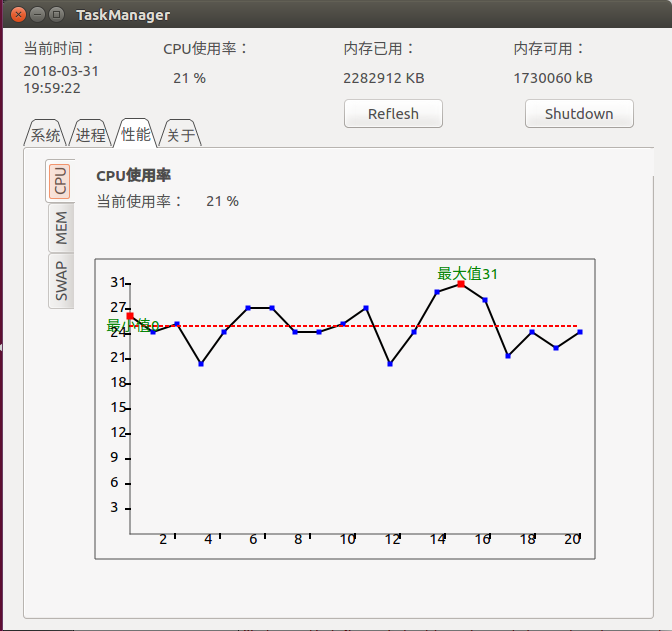


图4.5 CPU利用率曲线

5. 内存使用情况曲线测试，由于内存使用情况较为稳定，没有CPU利用率的波动大，仅凭曲线很难看出，所以采用对照实验，首先截取一张没有启动firefox浏览器时的内存使用情况，如图4.6所示

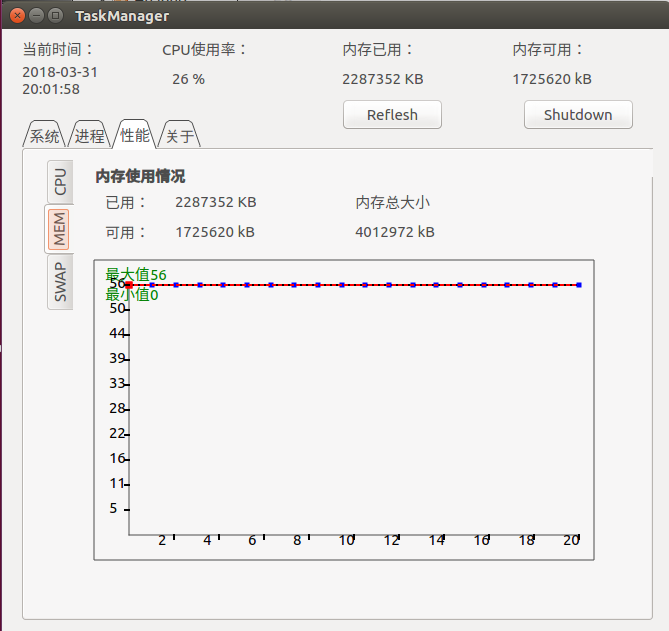


图4.6内存使用情况曲线

然后再打开firefox浏览器，可以看出内存使用率明显上升，结果如图4.7所示

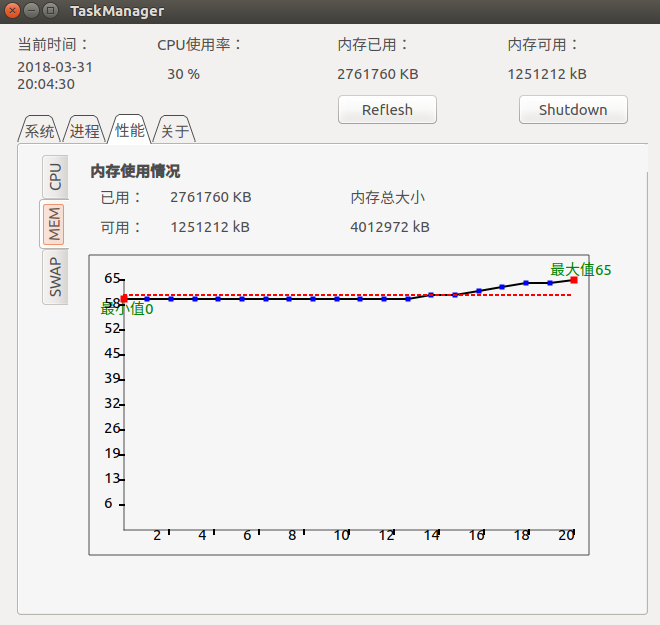


图4.7打开firefox浏览器后内存使用情况曲线

再把firefox浏览器关闭，内存使用率明显下降，结果如图4.8所示，程序运行正确。

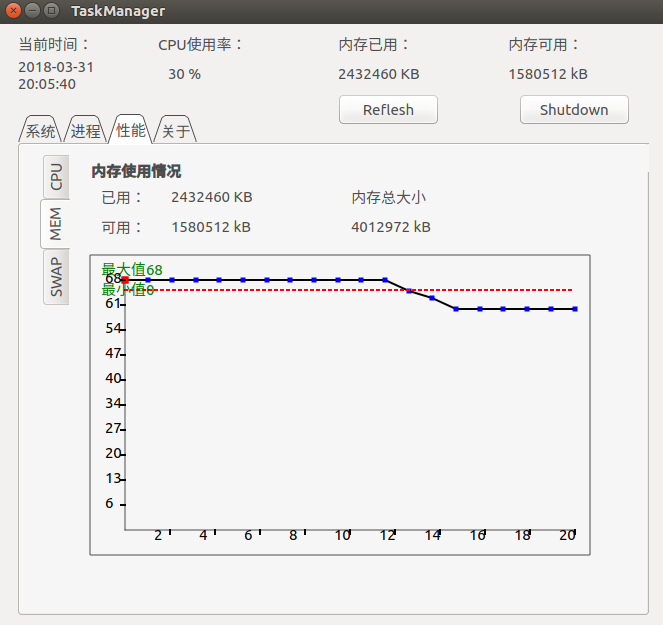


图4.8关闭firefox浏览器后内存使用情况曲线

6. 最后测试关机指令运行结果，点击Shutdown按钮，弹出一个选项框，如图4.9所示，选择ok，ubuntu系统正确关机，程序运行正确。

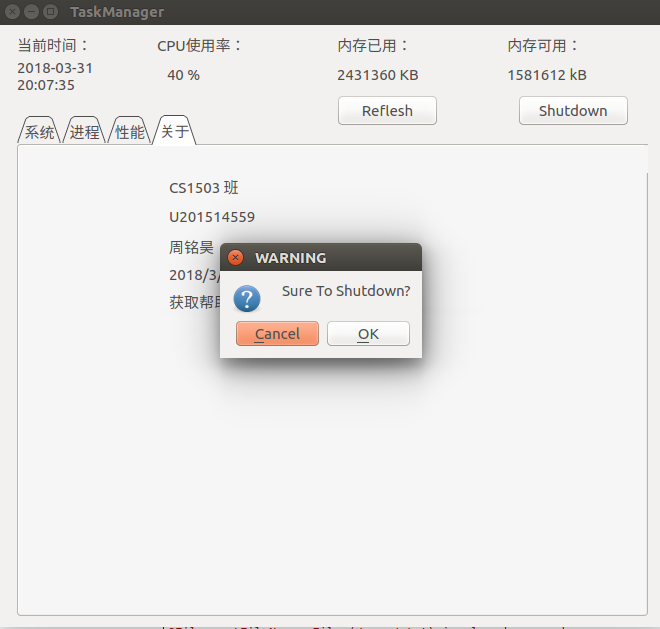


图4.9关机测试

## **4.4 实验总结**

通过本次实验，我了解了linux系统下各类监控信息的存储位置，学会了如何读取这些数据并且显示出来，学会了使用Qt在一个tabWidget内画坐标图，也熟悉了Qt在linux下静态编译的配置qmake过程。

经历这次课设，我对linux系统的熟练度有所提高，掌握了很多指令的使用，理解了很多关键文件的具体含义，学会了在内核态编写系统调用，增加简单的字符驱动设备。对操作系统的具体功能和结构，以及这些功能的具体实现方式有了更深的理解。