

# 目 录

<b>1</b>	<b>实验一 Linux 下编程 .....</b>	<b>1</b>
1.1	实验目的.....	1
1.2	实验内容.....	1
1.3	实验设计.....	1
1.3.1	开发环境.....	1
1.3.2	实验设计.....	1
1.4	实验调试.....	4
1.4.1	实验步骤.....	4
1.4.2	实验调试及心得.....	5
	附录 实验代码.....	7
<b>2</b>	<b>实验二 添加系统调用 .....</b>	<b>23</b>
2.1	实验目的.....	23
2.2	实验内容.....	23
2.3	实验设计.....	23
2.3.1	开发环境.....	23
2.3.2	实验设计.....	23
2.4	实验调试.....	25
2.4.1	实验步骤.....	25
2.4.2	实验调试及心得.....	28
	附录 实验代码.....	29
<b>3</b>	<b>实验三 添加设备驱动程序 .....</b>	<b>31</b>
3.1	实验目的.....	31
3.2	实验内容.....	31
3.3	实验设计.....	31
3.3.1	开发环境.....	31
3.3.2	实验设计.....	31
3.4	实验调试.....	36
3.4.1	实验步骤.....	36
3.4.2	实验调试及心得.....	38
	附录 实验代码.....	40
<b>4</b>	<b>实验四 Linux 系统监控器 .....</b>	<b>47</b>
4.1	实验目的.....	47
4.2	实验内容.....	47
4.3	实验设计.....	48
4.3.1	开发环境.....	48
4.3.2	实验设计.....	48
4.4	实验调试.....	57
4.4.1	实验步骤.....	57

4.4.2	实验调试及心得.....	62
附录	实验代码.....	64

# 1 实验一 Linux 下编程

## 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.掌握在 Linux 下的编程环境。

## 1.2 实验内容

- 1.编写一个 C 程序，用 read、write 等系统调用实现文件拷贝功能。命令形式：  
copy <源文件名> <目标文件名>
- 2.编写一个 C 程序，使用图形编程库 (QT/GTK)分窗口显示三个并发进程的  
运行(一个窗口实时显示当前系统时间，一个窗口循环显示 0 到 9，一个窗口做 1  
到 1000 的累加求和，刷新周期均为 1 秒)。

## 1.3 实验设计

### 1.3.1 开发环境

硬件：Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz

RAM 8GB

软件：Windows 10 下 VirtualBox 虚拟机 Ubuntu 16.04 64 位

Linux kernel: 4.15.0-45-generic

QT Creator 5.3

### 1.3.2 实验设计

- 1.编写一个 C 程序，用 read、write 等系统调用实现文件拷贝功能。

首先定义共享内存区结构体，结构体包含两个成员。分别为存储的数据，与存储的数据长度。同时还要定义存储的数据最大大小、共享内存的数量以及共享内存的起始 key 值（用于子进程获取共享内存区）。

在 writebuf 进程内，首先获取两个信号灯，代表空闲缓冲区以及满缓冲区，然后获取 main 里创建的共享内存区。首先对空闲缓冲区进行一次 P 操作，然后将共享内存区连接到当前进程的地址空间中，然后读取规定大小的数据存放在共

享内存区的数据区，返回读取数据的实际长度，然后对满缓冲区进行一次 V 操作。撤离当前共享内存区，指向下一共享内存区。如果实际读取的长度小于规定大小，则结束循环。

同理，在 `writebuf` 进程内，首先获取两个信号灯，代表空闲缓冲区以及满缓冲区，然后获取 `main` 里创建的共享内存区。首先对满缓冲区进行一次 P 操作，然后将共享内存区连接到当前进程的地址空间中，然后将共享内存区的数据区的内容写入到目标文件中，返回共享内存区的数据区的内容的实际长度，然后对空闲缓冲区进行一次 V 操作。撤离当前共享内存区，指向下一共享内存区。如果共享内存区的数据区的内容的实际长度小于规定大小，则结束循环。

在 `main` 函数中，产生两个信号灯并对它们赋初值。然后产生共享内存区。创建两个子进程分别执行 `writebuf` 和 `readbuf`。等待两个子进程结束后，删除共享内存区和信号灯，结束。

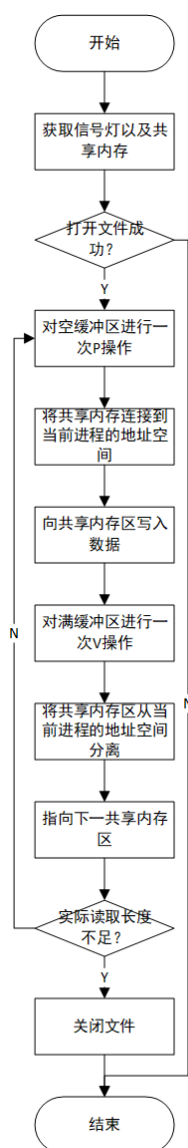


图 1-1 `writebuf` 流程图

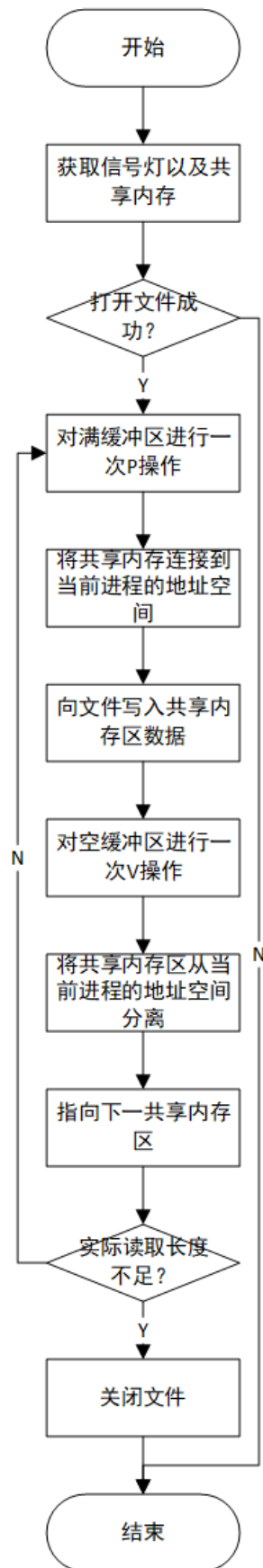


图 1-2 readbuf 流程图

2.编写一个 C 程序，使用图形编程库 (QT/GTK)分窗口显示三个并发进程的运行(一个窗口实时显示当前系统时间，一个窗口循环显示 0 到 9，一个窗口做 1 到 1000 的累加求和，刷新周期均为 1 秒)

#### (1) 显示当前系统时间

利用 `QLabel` 类生成一个 `label` 显示当前的系统时间，设定这个窗口的初始位置以及 `label` 的初始大小。然后利用信号和槽，槽为获取时间的函数。实现实时显示系统时间的功能。

设定一个定时器，当定时器溢出时，就更新一次时间，设定定时 1s。

获取系统时间就利用 `time` 函数获取当前系统时间，然后利用 `ctime` 函数显示在 `label` 上。

#### (2) 循环显示 0 到 9

利用 `QLabel` 类生成一个 `label` 显示当前的系统时间，设定这个窗口的初始位置以及 `label` 的初始大小。然后利用信号和槽，槽为循环显示的函数。

设定一个定时器，当定时器溢出时，就更新一次时间，设定定时 1s。

循环显示就声明一个私有成员 `cnt`，将  $(cnt++) \% 10$  送入需要显示的信息中去，显示在 `label` 上。

#### (3) 1 到 1000 的累加求和

利用 `QLabel` 类生成一个 `label` 显示当前的系统时间，设定这个窗口的初始位置以及 `label` 的初始大小。然后利用信号和槽，槽为累加函数。

设定一个定时器，当定时器溢出时，就更新一次时间，设定定时 1s。

累加就声明一个私有成员 `sum` 和 `adder`，每次调用累加函数时，判断 `adder` 是否小于 1000，如果是，则 `adder++`，`sum+=adder`，然后将 `sum` 显示在 `label` 上。

#### (4) main 函数

类似进程的处理方法，在 `main` 函数里创建三个子进程，每个进程实现显示一个窗口的功能，最后等三个子进程都结束后，主程序结束。

## 1.4 实验调试

### 1.4.1 实验步骤

1.在 OS 实验的基础上修改代码，使得符合课程设计的要求。

2.学习 QT 相关知识，并且编写三个窗口显示程序和主程序，并编译运行

## 1.4.2 实验调试及心得

1.编写一个 C 程序，用 read、write 等系统调用实现文件拷贝功能。

(1) 运行结果

下图显示 copy 程序运行之前，文件夹里只有 src 文件，而在运行了 ./copy src dst 后，文件夹里多了一个 dst 文件，这个就是复制后的文件。然后使用 diff 命令，可以看到两个文件完全没有差异。

```
may@may-VirtualBox:~/05/Lab1_1$ ls
copy main.c my_struct.h readbuf readbuf.c src writebuf writebuf.c
may@may-VirtualBox:~/05/Lab1_1$ ./copy src dst
Copy ...
Done.
may@may-VirtualBox:~/05/Lab1_1$ ls
copy dst main.c my_struct.h readbuf readbuf.c src writebuf writebuf.c
may@may-VirtualBox:~/05/Lab1_1$ diff src dst
may@may-VirtualBox:~/05/Lab1_1$
```

图 1-3 copy 程序测试图

(2) 实验调试

由于本次实验是基于上学期实验课的代码修改而来，没有遇到调试上的问题。

(3) 实验心得

本次实验让我掌握了 linux 下共享内存的概念与使用方法。共享内存也是进程间进行通信的一种方法，在之前几次实验中我也学到了其他的方法，这次实验则是提出了共享内存的概念。与此同时，我还掌握了环形缓冲的结构与使用方法，知道了它的工作原理，将课堂上的知识运用到实践中来了。结合前面几次实验，我对 Linux 进程间的不同的通信方式有了广泛的了解。

2.编写一个 C 程序，使用图形编程库 (QT/GTK)分窗口显示三个并发进程的运行(一个窗口实时显示当前系统时间，一个窗口循环显示 0 到 9，一个窗口做 1 到 1000 的累加求和，刷新周期均为 1 秒)

(1) 运行结果

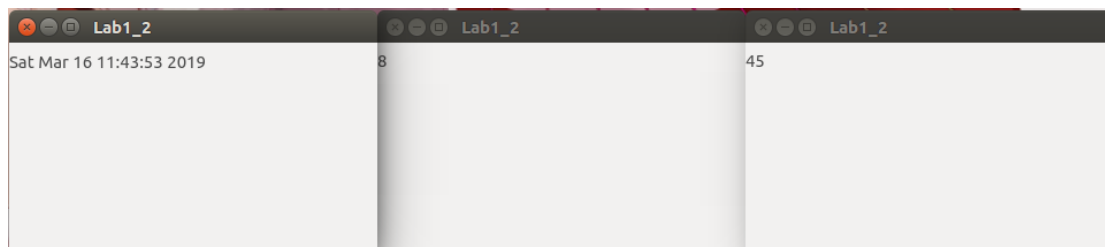


图 1-4 多窗口程序测试图

由上图可看出，最左边的窗口显示当前系统时间，中间窗口循环显示 0-9，最右边的窗口显示累加结果。

(2) 实验调试

本次实验遇到的调试问题主要是 QT 的使用问题。查找相关手册就解决了问题，没有遇到什么有实际价值的问题。

在安装 QT 的时候，也需要配置环境，通过上网查找资料，使用了以下命令

```
export QTDIR=/opt/Qt5.3.0/5.3/gcc_64
```

```
export PATH=QTDIR/bin:PATH
```

```
export LD_LIBRARY_PATH=QTDIR/lib:LD_LIBRARY_PATH
```

然后在 `bashrc` 文件中 `PATH` 行加上 `QTbin` 所在的目录。

### （3）实验心得

本次实验，我掌握了 QT 的一些基本方方，包括定时器的使用，信号和槽的使用。本次实验对之后的实验起到了一个铺垫的作用，为以后的实验打下了基础。不过在这次实验中学到的方法还不够，只能解决最基本的显示窗口的问题，在之后还要多加学习 QT 相关的知识。



## 附录 实验代码

```
#ifndef MYSTRUCT_H
#define MYSTRUCT_H

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#define DATA_SIZE 100000
#define BUF_SIZE 10 //共享内存数量
#define s_addr 100

union semun {
    int          val;
    struct semid_ds *buf;
    unsigned short *array;
    struct seminfo *__buf;
};

typedef struct sh_mem
{
    char data[DATA_SIZE];
    int  length;
}sh_mem;

#endif

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <string.h>
#include <fcntl.h>
#include "my_struct.h"
```

```
int buf_empty;//空闲缓冲区
```

```
int buf_full;//满缓冲区
```

```
void P(int semid,int index)
```

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

```
void V(int semid,int index)
```

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

```
int main(int argc, char **argv) {
```

```
    int fp;
```

```
    int shm_id[BUF_SIZE];
```

```
    sh_mem *p;
```

```
    //获取信号灯
```

```
    buf_empty = semget(1,1,IPC_CREAT|0666);
```

```
    buf_full = semget(2,1,IPC_CREAT|0666);
```

```
    //获取共享内存区
```

```
    for(int i=0;i<BUF_SIZE;++i)
```

```

{
    shm_id[i] = shmget(s_addr+i,sizeof(sh_mem),IPC_CREAT|0666);
}

```

//打开文件

```

// if((fp = open("dst",O_WRONLY|O_CREAT,0666)) == -1)
// {
//     printf("Fail to open file.\n");
// }

```

```

if (argc <= 2) {
    if((fp = open("dst",O_WRONLY|O_CREAT,0666)) == -1)
    {
        printf("Fail to open file.\n");
        return -1;
    }
} else {
    if((fp = open(argv[2],O_WRONLY|O_CREAT,0666)) == -1)
    {
        printf("Fail to open file.\n");
        return -1;
    }
}

```

```

int i=0,temp=0;
while (1) {
    P(buf_full,0);
    p=(sh_mem *)shmat(shm_id[i],0,0);
    write(fp,p->data,p->length);
    V(buf_empty,0);
    temp=p->length;//判断结束条件
    shmdt(p);
    i=(i+1)%BUF_SIZE;//环形缓冲区
    if(temp<DATA_SIZE) break;
}

```

```

    }

    //关闭文件
    close(fp);

    return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <fcntl.h>
#include "my_struct.h"

int buf_empty;//空闲缓冲区
int buf_full;//满缓冲区

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

void V(int semid,int index)
{
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = 1;
    sem.sem_flg = 0;

```

```

    semop(semid,&sem,1);
    return;
}

int main(int argc, char **argv) {
    int fp;
    int shm_id[BUF_SIZE];
    sh_mem *p;

    //获取信号灯
    buf_empty = semget(1,1,IPC_CREAT|0666);
    buf_full  = semget(2,1,IPC_CREAT|0666);

    //获取共享内存区
    for(int i=0;i<BUF_SIZE;++i)
    {
        shm_id[i] = shmget(s_addr+i,sizeof(sh_mem),IPC_CREAT|0666);
    }

    //打开文件
    // if((fp = open("src",O_RDONLY)) == -1)
    // {
    //     printf("Fail to open file.\n");
    // }

    if (argc < 2) {
        if((fp = open("src",O_RDONLY)) == -1)
        {
            printf("Fail to open file. writebuf\n");
            return -1;
        }
    } else {
        if((fp = open(argv[1],O_RDONLY)) == -1)
        {
            printf("Fail to open file. writebuf\n");

```

```

        return -1;
    }
}

int i=0,temp=0;
while (1) {
    P(buf_empty,0);
    p=(sh_mem *)shmat(shm_id[i],0,0);
    p->length=read(fp, p->data, DATA_SIZE);
    V(buf_full,0);
    temp=p->length;//判断结束条件
    shmdt(p);
    i=(i+1)%BUF_SIZE;//环形缓冲区
    if(temp<DATA_SIZE) break;
}

//关闭文件
close(fp);

return 0;
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include "my_struct.h"

int buf_empty;//空闲缓冲区
int buf_full;//满缓冲区

void P(int semid,int index)
{

```

```

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

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

int main(int argc, char **argv) {
    int status;//子进程状态
    pid_t writebuf_pid;
    pid_t readbuf_pid;
    int shm_id[BUF_SIZE];//共享内存
    sh_mem *p;

    //产生信号灯
    buf_empty = semget(1,1,IPC_CREAT|0666);
    buf_full = semget(2,1,IPC_CREAT|0666);

    //赋值
    union semun arg1;
    union semun arg2;
    arg1.val=10;
    arg2.val=0;
    semctl(buf_empty,0,SETVAL,arg1);

```

```

semctl(buf_full,0,SETVAL,arg2);

//产生共享内存区
for(int i=0;i<BUF_SIZE;++i)
{
    shm_id[i] = shmget(s_addr+i,sizeof(sh_mem),IPC_CREAT|0666);
}

printf("Copy ... \n");

while ((writebuf_pid = fork()) == -1) ;
if (writebuf_pid == 0) { //子进程
    execlp("./writebuf", "writebuf", argv[1], argv[2], argv[argc], NULL);
} else { //main
    while ((readbuf_pid = fork()) == -1);
    if (readbuf_pid == 0) { //子进程
        execlp("./readbuf", "readbuf", argv[1], argv[2], argv[argc], NULL);
    } else { // main
        //等待子进程结束
        for (int i = 0; i < 2; i++) {
            waitpid(-1, &status, 0);
        }
        printf("Done.\n");

        //删除共享内存区
        for(int i=0;i<BUF_SIZE;++i)
        {
            shmctl(shm_id[i],IPC_RMID,0);
        }

        //删除信号灯
        semctl(buf_empty,0,IPC_RMID);
        semctl(buf_full,0,IPC_RMID);

        return 0;
    }
}

```



```

    }
}

}

//QT
#ifndef LOOPWINDOW_H
#define LOOPWINDOW_H

#include <QMainWindow>
#include <QLabel>

class LoopWindow : public QMainWindow
{
    Q_OBJECT

public:
    LoopWindow(QWidget *parent = 0);
    ~LoopWindow(void);

private:
    QLabel *label;
    int cnt;
    char *LoopMessage;
    QMainWindow &setText(const char *txt);
    const char* getLoop(void);

private slots:
    void LoopUpdate(void);
};

#endif

#include <QString>

```

```

#include <QTimer>
#include <cstdio>
#include <unistd.h>
#include "loopwindow.h"

using namespace std;

LoopWindow::LoopWindow(QWidget *parent) : QMainWindow(parent)
{
    move(QPoint(800, 200));
    label = new QLabel(this);
    label->setFixedSize(500,30);
    label->setText("");
    LoopMessage = new char[10];

    //信号和槽
    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(LoopUpdate()));
    timer->start(1000);
}

LoopWindow::~~LoopWindow(void)
{
    delete label;
    delete LoopMessage;
}

const char* LoopWindow::getLoop(void)
{
    sprintf(LoopMessage,"%d", (cnt++)%10);
    return LoopMessage;
}

void LoopWindow::LoopUpdate(void)
{

```

```

        label->setText(QString(getLoop()));
    }

#ifndef SUMWINDOW_H
#define SUMWINDOW_H

#include <QMainWindow>
#include <QLabel>

class SumWindow : public QMainWindow
{
    Q_OBJECT

public:
    SumWindow(QWidget *parent = 0);
    ~SumWindow(void);

private:
    QLabel *label;
    int sum;
    int adder;
    char *SumMessage;
    QMainWindow &setText(const char *txt);
    const char *getSum(void);

private slots:
    void SumUpdate(void);
};

#endif

#include <QTimer>
#include <QString>

```

```

#include <cstdio>
#include "sumwindow.h"

using namespace std;

SumWindow::SumWindow(QWidget *parent) : QMainWindow(parent)
{
    move(QPoint(1000, 200));
    label = new QLabel(this);
    label->setFixedSize(500,30);
    label->setText("");
    sum = 0;
    adder = 0;
    SumMessage = new char[10];

    //信号和槽
    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(SumUpdate()));
    timer->start(1000);
}

SumWindow::~SumWindow(void)
{
    delete label;
    delete SumMessage;
}

const char *SumWindow::getSum(void) {
    if (adder < 1000) {
        adder++;
        sum += adder;
        sprintf(SumMessage, "%d", sum);
    }

    return SumMessage;
}

```

```

}

void SumWindow::SumUpdate(void) {
    label->setText(QString(getSum()));
}

#ifndef TIMEWINDOW_H
#define TIMEWINDOW_H

#include <QMainWindow>
#include <QLabel>

class TimeWindow : public QMainWindow
{
    Q_OBJECT

public:
    TimeWindow(QWidget *parent = 0);
    ~TimeWindow(void);

private:
    QLabel *label;
    QMainWindow &setText(const char *txt);
    const char* getTime(void);

private slots:
    void TimeUpdate(void);
};

#endif

#include <QTimer>
#include <QString>

```

```

#include <ctime>
#include "timewindow.h"

using namespace std;

TimeWindow::TimeWindow(QWidget *parent) : QMainWindow(parent)
{
    move(QPoint(600, 200));
    label = new QLabel(this);
    label->setFixedSize(500,50);
    label->setText("");

    //信号和槽
    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(TimeUpdate()));
    timer->start(1000);
}

TimeWindow::~TimeWindow(void)
{
    delete label;
}

const char* TimeWindow::getTime(void) {
    time_t ct;
    ct = time(NULL);
    return ctime(&ct);
}

void TimeWindow::TimeUpdate(void) {
    label->setText(QString(getTime()));
}

#include <QApplication>

```

```

#include <unistd.h>
#include <sys/wait.h>
#include "timewindow.h"
#include "loopwindow.h"
#include "sumwindow.h"

int main(int argc, char *argv[])
{
    pid_t timewindow_pid, loopwindow_pid, sumwindow_pid;

    while ((timewindow_pid = fork()) == -1);
    if (timewindow_pid == 0) {
        QApplication Qapp_time(argc, argv);
        TimeWindow timewindow;
        timewindow.show();
        Qapp_time.exec();
    } else {
        while ((loopwindow_pid = fork()) == -1);
        if (loopwindow_pid == 0) {
            QApplication Qapp_loop(argc, argv);
            LoopWindow loopwindow;
            loopwindow.show();
            Qapp_loop.exec();
        } else {
            while ((sumwindow_pid = fork()) == -1);
            if (sumwindow_pid == 0) {
                QApplication Qapp_sum(argc, argv);
                SumWindow sumwindow;
                sumwindow.show();
                Qapp_sum.exec();
            } else {
                for (int i = 0; i < 3; i++) {
                    waitpid(-1, NULL, 0);
                }
            }
        }
    }
}

```

}

}

}



## 2 实验二 添加系统调用

### 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。

### 2.2 实验内容

- 1.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。
- 2.另编写一个应用程序，调用新增加的系统调用。
- 3.实现的功能是：文件拷贝。

### 2.3 实验设计

#### 2.3.1 开发环境

硬件：Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz

RAM 8GB

软件：Windows 10 下 VirtualBox 虚拟机 Ubuntu 16.04 64 位

Linux kernel: 4.10.17

#### 2.3.2 实验设计

- 1.首先不修改内核代码，直接编译内核。

(1) 从 <https://www.kernel.org/pub/linux/kernel/> 下载 linux 内核代码，版本号为 4.10.17，解压到文件夹/usr/src

(2) 安装编译内核需要的依赖。

```
sudo apt-get install libncurses5-dev libssl-dev //显示内核配置
```

```
sudo apt-get install build-essential openssl //模块
```

```
sudo apt-get install libidn11-dev libidn11
```

(3) 清空旧内核

```
make mrproper
```

```
make clean
```

(4) 生成内核配置文件，编译内核

`make menuconfig` //生成内核配置文件

`make bzImage` //编译内核映像

`make modules` //编译内核模块

`make modules_install` //生成并安装模块

`make install` //安装新的系统

如果计算机有多核，可以在命令后加 `-j 4` 使用两核编译，这样会加快编译速度。

(5) 修改 `grub` 文件

修改 `/etc/default/grub` 文件，注释掉 `GRUB_HIDDEN_TIMEOUT=0`，然后运行 `update-grub` 命令。

(6) 重启，选择新核

执行 `reboot` 命令，选择新编译的内核。

## 2. 添加系统调用

(1) 编写文件拷贝函数。

需要使用内核函数实现功能，因为在内核态不支持其他库。

在这里我是用了 `filp_open` 函数打开文件，然后使用 `vfs_read` 从源文件读数据，存储到 `buffer` 里，返回实际读取的字节数，当读取的字节数不为 0 时，使用 `vfs_write` 函数将 `buffer` 里的数据写入到目标文件中去。最后使用 `filp_close` 函数关闭文件。

在这里要注意的地方：

```
mm_segment_t old_fs = get_fs();
set_fs(KERNEL_DS);
...
set_fs(old_fs);
```

第一步是要将现在的段保存到 `old_fs` 中，然后将段设置为内核段 `KERNEL_DS`，然后才能开始调用内核函数。还要值得注意的地方是 `vfs_read` 和 `vfs_write` 函数中参数 `buffer` 必须在用户空间中，因此 `buffer` 的定义必须在 `set_fs(KERNEL_DS)` 之前。最后还要还原，利用 `set_fs(old_fs)` 实现。

(2) 连接新的系统调用

在 `arch/x86/entry/syscalls/syscall_64.tbl` 中添加中断号，按照类似的规则添加即可。然后在 `include/linux/syscalls.h` 添加中断处理函数声明，这个也按照上面的声明类似即可。然后在内核中 `Makefile` 文件中加上 `core-y += mycopy/`（自己系统调用所在目录），并且在 `mycopy/` 中编写 `makefile` 文件，只需要加上 `obj-y = mycopy.o` 即可，这样可以使得 `mycopy.c` 在内核编译的时候一起编译，并且将生

成的 mycopy.o 链接到内核中。

### (3) 编译新内核

按照 1.的步骤重新编译新内核，重启后选择新编译的内核。

### (4) 编写测试文件测试新的系统调用

只需要调用 syscall 函数即可，所需参数为系统调用号，源文件名，目的文件名。

## 2.4 实验调试

### 2.4.1 实验步骤

#### 1.下载内核源码并解压

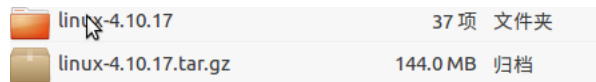


图 2-1 内核源码文件图

#### 2.安装编译内核需要的依赖。

```
sudo apt-get install libncurses5-dev libssl-dev //显示内核配置
```

```
sudo apt-get install build-essential openssl //模块
```

```
sudo apt-get install libidn11-dev libidn11
```

#### 3.清空旧内核

```
make mrproper
```

```
make clean
```

#### 4.生成内核配置文件，编译内核

```
make menuconfig //生成内核配置文件
```

```
make bzImage //编译内核映像
```

```
make modules //编译内核模块
```

```
make modules_install //生成并安装模块
```

```
make install //安装新的系统
```

#### 5.修改 grub 文件

修改/etc/default/grub 文件，注释掉 GRUB\_HIDDEN\_TIMEOUT=0，然后运行 update-grub 命令。

#### 6.重启，选择新核

执行 reboot 命令，选择新编译的内核。

由于第一次编译新内核的时候没有截图，只有在后面编译新内核的有截图，不过因为两个过程比较相似，所以在后面会有截图。

## 7.编写文件拷贝函数

具体的思路已经在实验设计中给出，在此不再赘述。

## 8.连接新的系统调用

在 `arch/x86/entry/syscalls/syscall_64.tbl` 中添加中断号，按照类似的规则添加即可。如下图：

```
-----  
547 x32 pwritev2          compat_sys_pwritev64v2  
548 common mycopy        sys_mycopy|
```

图 2-2 添加系统调用号

然后在 `include/linux/syscalls.h` 添加中断处理函数声明，这个也按照上面的声明类似即可。如下图：

```
asmlinkage long sys_mycopy(const char *src, const char *dst);
```

图 2-3 添加系统调用声明

然后在内核中 `Makefile` 文件中加上 `core-y += mycopy/` (自己系统调用所在目录)，如下图：

```
core-y      := usr/  
core-y      += mycopy/|
```

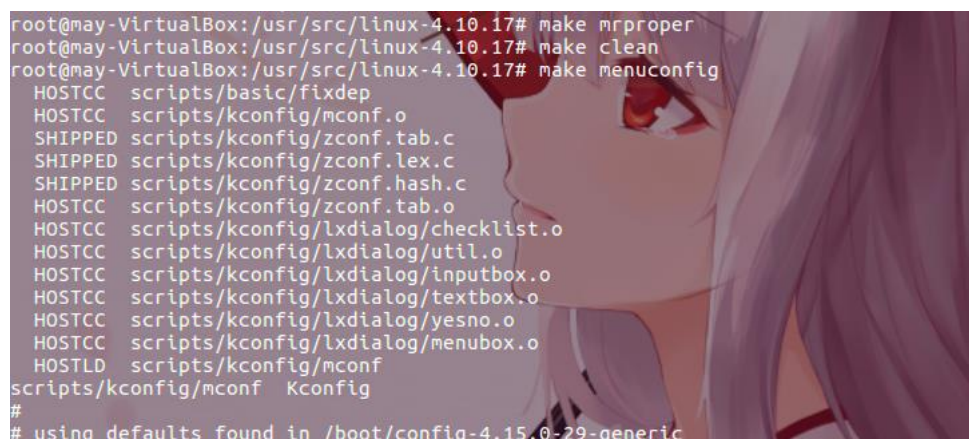
图 2-4 添加 makefile

并且在 `mycopy/` 中编写 `makefile` 文件，只需要加上 `obj-y = mycopy.o` 即可，这样可以使得 `mycopy.c` 在内核编译的时候一起编译，并且将生成的 `mycopy.o` 链接到内核中。

```
obj-y := mycopy.o
```

图 2-5 添加 makefile

## 9.编译新内核



```
root@may-VirtualBox:/usr/src/linux-4.10.17# make mrproper  
root@may-VirtualBox:/usr/src/linux-4.10.17# make clean  
root@may-VirtualBox:/usr/src/linux-4.10.17# make menuconfig  
HOSTCC  scripts/basic/fixdep  
HOSTCC  scripts/kconfig/mconf.o  
SHIPPED  scripts/kconfig/zconf.tab.c  
SHIPPED  scripts/kconfig/zconf.lex.c  
SHIPPED  scripts/kconfig/zconf.hash.c  
HOSTCC  scripts/kconfig/zconf.tab.o  
HOSTCC  scripts/kconfig/lxdialog/checklist.o  
HOSTCC  scripts/kconfig/lxdialog/util.o  
HOSTCC  scripts/kconfig/lxdialog/inputbox.o  
HOSTCC  scripts/kconfig/lxdialog/textbox.o  
HOSTCC  scripts/kconfig/lxdialog/yesno.o  
HOSTCC  scripts/kconfig/lxdialog/menubox.o  
HOSTLD  scripts/kconfig/mconf  
scripts/kconfig/mconf Kconfig  
#  
# using defaults found in /boot/config-4.15.0-29-generic
```

图 2-6 编译新内核

```
CC security/apparmor/file.o
CC crypto/memneq.o
CC security/apparmor/crypto.o
LD kernel/sched/built-in.o
CC kernel/time/time.o
CC crypto/crypto_wq.o
CC security/apparmor/capability.o
CC crypto/algapi.o
CC kernel/time/timer.o
CC fs/seq_file.o
CC security/apparmor/resource.o
LD security/apparmor/apparmor.o
LD security/apparmor/built-in.o
CC fs/xattr.o
CC security/integrity/iint.o
CC crypto/scatterwalk.o
CC security/integrity/integrity_audit.o
CC kernel/time/hrtimer.o
CC crypto/proc.o
CC security/integrity/digsig.o
CC fs/libfs.o
CC security/integrity/digsig_asymmetric.o
CC crypto/aead.o
```

图 2-7 编译新内核

```
t is no longer supported.
Found linux image: /boot/vmlinuz-4.15.0-29-generic
Found initrd image: /boot/initrd.img-4.15.0-29-generic
Found linux image: /boot/vmlinuz-4.10.17
Found initrd image: /boot/initrd.img-4.10.17
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
root@may-VirtualBox: /usr/src/linux-4.10.17#
root@may-VirtualBox: /usr/src/linux-4.10.17#
```

图 2-8 编译新内核

```
Setup is 17436 bytes (padded to 17920 bytes).
System is 7252 kB
CRC 7b39abfa
Kernel: arch/x86/boot/bzImage is ready (#1)
root@may-VirtualBox: /usr/src/linux-4.10.17#
```

图 2-9 编译新内核

```
may@may-VirtualBox:~$ uname -a
Linux may-VirtualBox 4.10.17 #1 SMP Mon Mar 4 23:11:19 CST 2019 x86_64 x86_64 x8
6_64 GNU/Linux
may@may-VirtualBox:~$ uname -r
4.10.17
may@may-VirtualBox:~$
```

图 2-10 编译新内核

## 10.测试新的系统调用

```
may@may-VirtualBox:~/OS/Lab2$ ls
linux-4.10.17.tar.gz  mycopy.c  mycopy_tb.c  src  test  新建文本文档.txt
may@may-VirtualBox:~/OS/Lab2$ ./test
dst  linux-4.10.17.tar.gz  mycopy.c  mycopy_tb.c  src  test  新建文本文档.txt
may@may-VirtualBox:~/OS/Lab2$ diff src dst
may@may-VirtualBox:~/OS/Lab2$
```

图 2-11 测试新的系统调用

如上图所示，测试新的系统调用。可以看到测试之前并没有 dst 文件，测试之后新生成了 dst 文件。然后使用 diff 命令，判断两个文件没有差异。

## 2.4.2 实验调试及心得

### 1.实验调试

因为在做内核编译前做了很多准备，包括把一些依赖都安装了等等。

但是在编写内核代码的时候，尽管知道应该使用内核提供的函数，还是漏掉了一个注意事项。即应该首先将访问限制值设置为内核的内存访问范围。因为一开始编写代码时，以为只需要将调用的函数置换一下即可，就忽略掉了这个点。后来经过查阅资料，发现在调用内核函数 `filp_open`, `vfs_read`, `vfs_write` 等时，为了避免内存保护检查错误，要暂时将访问限制值设置为内核的内存访问范围，然后再修改为原来的值。

### 2.实验心得

在编译未修改的内核源码时，我注意观察了它的过程。在编译内核模块的时候，发现 `driver` 中的文件编译所用时间较长。知道了怎么样添加系统功能调用，这跟理论课上学到的知识十分相符合，首先就要添加新的系统调用号，有一点新颖的地方就是还要添加调用函数声明。后来知道 `syscall` 所有的函数命名都在一个文件，这和我们平时编写程序也没有什么明显的区别。然后就是自己编写系统调用函数了，第一个知道的就是系统调用不能使用标准库提供的函数，只能使用内核函数，所以之前写的程序必须全部替换。然后知道了必须要先切换到内核段，才能使用内核提供函数，最后还要注意还原。还知道了 `makefile` 文件中 `core-y` 参数的含义。

其实在刚开始内核编译的时候，手头上并没有什么资料，这使得实验很难进行下去。后来通过查阅大量资料，了解到了添加系统调用的一般方法，首先自己理解了这些步骤，为什么要这样进行，这样进行是为了什么工作。在理解了这些之后，添加系统调用也就不是一件很难的事情了。

经过这次实验，了解到其实添加系统调用只是内核中的一部分，但是通过这次经验，让自己对内核这个东西并没有很陌生了，之后遇到相关的问题也能通过查阅资料一一解决了。这次实验只是将我引入了大门，了解到一些浅薄的知识，后面的积累都需要时间。

## 附录 实验代码

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <asm/segment.h>

asmlinkage long sys_mycopy (const char *src, const char *dst) {
    struct file *psrc;
    struct file *pdst;
    loff_t read_pos = 0, write_pos = 0;
    int read_bytes_num = 0;
    char buf[100];

    mm_segment_t old_fs = get_fs();
    set_fs(KERNEL_DS);

    psrc = filp_open(src, O_CREAT | O_RDWR, S_IRWXU | S_IRWXG |
S_IRWXO);
    if (IS_ERR(psrc)) {
        printk("mycopy fail to open file.\n");
        return -1;
    }
    pdst = filp_open(dst, O_CREAT | O_RDWR, S_IRWXU | S_IRWXG |
S_IRWXO);
    read_bytes_num = vfs_read(psrc, buf, 90, &read_pos);
    while (read_bytes_num != 0) {
        vfs_write(pdst, buf, read_bytes_num, &write_pos);
        read_bytes_num = vfs_read(psrc, buf, 90, &read_pos);
    }

    filp_close(psrc, 0);
    filp_close(pdst, 0);
}
```

```
    set_fs(old_fs);

    return 0;
}
```

```
#include <unistd.h>
#include <sys/syscall.h>
```

```
int main(void) {
    syscall(548, "src", "dst");
    return 0;
}
```



## 3 实验三 添加设备驱动程序

### 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.实现字符设备的驱动

### 3.2 实验内容

- 1.采用模块方法，添加一个新的字符设备驱动程序，实现打开/关闭、读/写等基本操作
- 2.编写一个应用程序，测试添加的驱动程序

### 3.3 实验设计

#### 3.3.1 开发环境

硬件：Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz

RAM 8GB

软件：Windows 10 下 VirtualBox 虚拟机 Ubuntu 16.04 64 位

Linux kernel: 4.15.0-45-generic

#### 3.3.2 实验设计

- 1.编写设备驱动程序，按照要求实现字符设备的功能，具体设计如下。
- 2.将要编写的函数与 `file_operation` 中对应的函数相连接，因为设备在 linux 中被当做文件处理，如：

```
static const struct file_operations fops = {  
    .owner = THIS_MODULE,  
    .open = my_open,  
    .read = my_read,  
    .write = my_write,
```

```
.release = my_release  
};
```

### 3.编写 my\_open 函数。

注意在这里我考虑了字符设备的**互斥问题**。当这个字符设备被占用时，另一个程序不能强制占用此设备。

因此在这里我设置了一个信号量 `mydeviceMutex` 用来互斥访问。只有当此字符设备空闲时，才可占用。每打开一次，使得打开次数加一，并使用 `printk` 显示出来。

### 4.编写 my\_release 函数

同样，在这里也应该使用信号灯相关操作，应该将字符设备资源释放，也就是资源解锁。然后打印关闭成功信息。

### 5.编写 my\_read 函数

流程图见图 3-1。

### 6.编写 my\_write 函数

流程图见图 3-2。

### 7.编写 init 函数

`Init` 函数只在初始化时被调用一次，其余时间不调用。流程图见图 3-3。

### 8.编写 exit 函数

调用 `device_destroy`, `class_unregister`, `class_destroy`, `unregister_chrdev` 即可。

9.最后将 `module_init` 参数设为 `init` 函数入口，将 `module_exit` 参数设为 `exit` 函数入口即可。

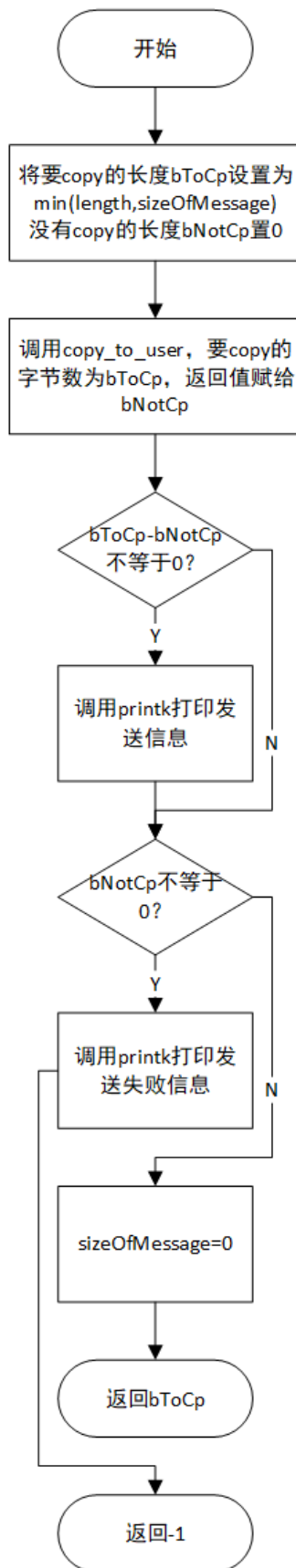


图 3-1 my\_read 函数流程图

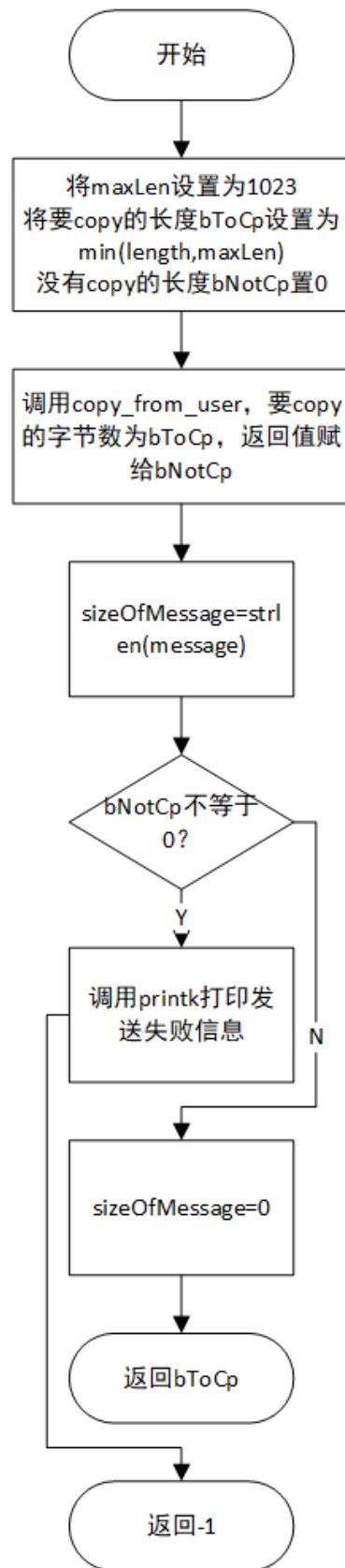


图 3-2 my\_write 函数流程图

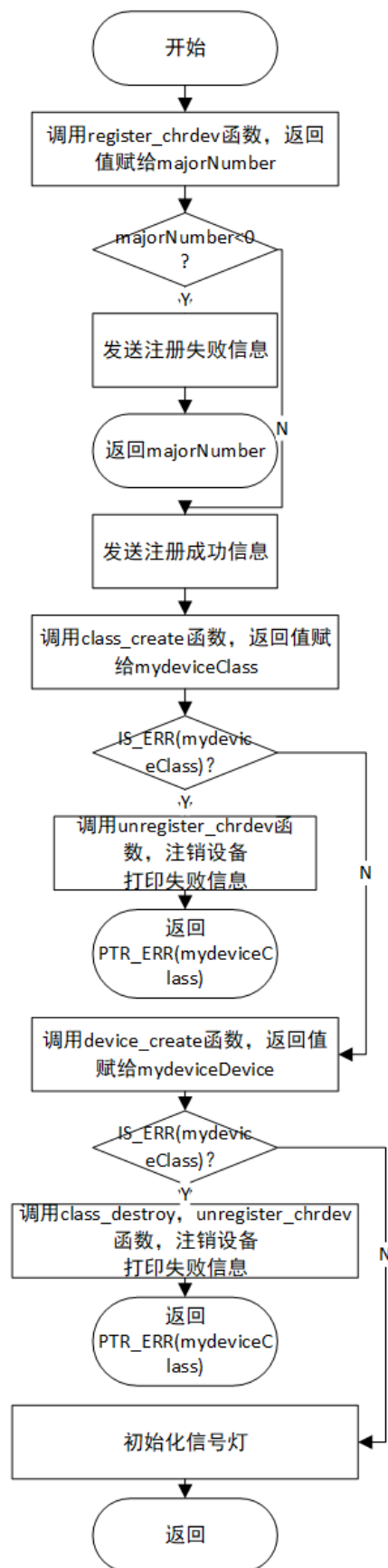


图 3-3 MyCharDev\_init 函数流程图

## 10.编写 makefile

Makefile 比较简单，如下：

```
obj-m+=mydevice.o
```

```
all:
```

```
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
    $(CC) mydevice_tb.c -o test
```

```
clean:
```

```
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
    rm test
```

第一行将 mydevice.o 连接至内核。-C 表示在执行任何 make 之前，将目录切换到内核目录，\$(shell uname -r)返回当前内核版本，具有通用性。M=\$(PWD)表示 make 在实际的项目文件中存在什么命令。

## 11.编写测试文件

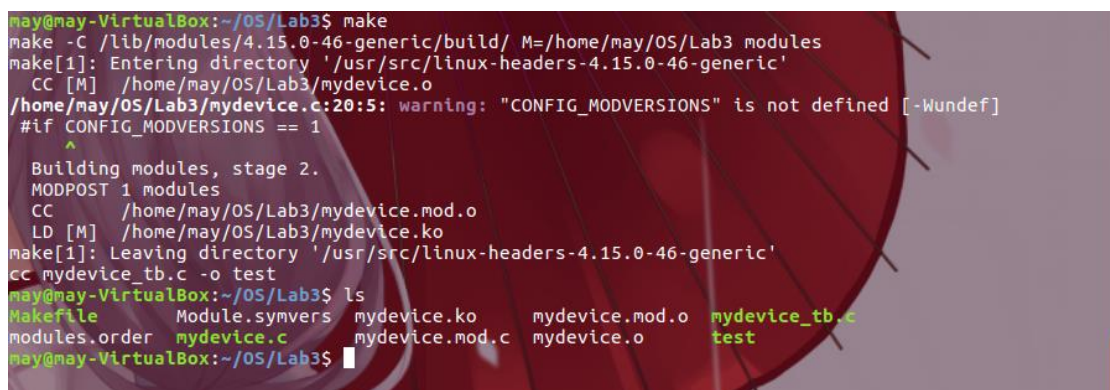
测试文件首先利用 open 函数打开设备，然后输入字符串，使用 write 和 read 函数测试设备是否正常工作。逻辑比较简单，在此不赘述。详细代码见后附清单。

# 3.4 实验调试

## 3.4.1 实验步骤

1.编写设备驱动程序，makefile 文件，驱动测试文件。

2.输入 make 指令，如下图：



```
may@may-VirtualBox:~/OS/Lab3$ make
make -C /lib/modules/4.15.0-46-generic/build/ M=/home/may/OS/Lab3 modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-46-generic'
  CC [M]  /home/may/OS/Lab3/mydevice.o
/home/may/OS/Lab3/mydevice.c:20:5: warning: "CONFIG_MODVERSIONS" is not defined [-Wundef]
  #if CONFIG_MODVERSIONS == 1
    ^
Building modules, stage 2.
MODPOST 1 modules
CC       /home/may/OS/Lab3/mydevice.mod.o
LD [M]   /home/may/OS/Lab3/mydevice.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-46-generic'
cc mydevice_tb.c -o test
may@may-VirtualBox:~/OS/Lab3$ ls
Makefile      Module.symvers  mydevice.ko     mydevice.mod.o  mydevice_tb.c
modules.order mydevice.c      mydevice.mod.c  mydevice.o      test
may@may-VirtualBox:~/OS/Lab3$
```

图 3-4 make

可以看到生成了 mydevice.ko 文件。

### 3.加入模块

输入 `sudo insmod mydevice.ko`，如下：

```
may@may-VirtualBox:~/OS/Lab3$ sudo insmod mydevice.ko
[sudo] may 的密码:
may@may-VirtualBox:~/OS/Lab3$
```

图 3-5 加入模块

### 4.查看模块

输入 `lsmod` 指令，可以看到模块已经加入成功。

```
may@may-VirtualBox:~/OS/Lab3$ lsmod
Module                Size  Used by
mydevice               16384  0
nls_utf8               16384  1
iso9660                45056  1
vboxsf                 45056  0
```

图 3-6 加入模块成功

### 5.查看设备注册号

```
may@may-VirtualBox:/dev$ ls -l My*
crw-rw-rw- 1 root root 243, 0 3月 19 15:38 MyCharDev
may@may-VirtualBox:/dev$
```

```
496.034004 mydevice: loading out-of-tree module taints kern
496.034239 Initializing the MyCharDev LKM
496.034242 Registered successfully with major number 243
496.034243 Device class registered successfully
496.034404 Device class created successfully
may@may-VirtualBox: $
```

图 3-7 查看设备注册号

可以看到主设备号为 243，两者一致。

### 6.测试

```
may@may-VirtualBox:~/OS/Lab3$ sudo ./test
Testing mydevice...
Type a string:
Hello may.sincerity.
Sending message: [Hello may.sincerity.].
Press ENTER to continue.
Receiving message: [Hello may.sincerity.]
Test Complete.
may@may-VirtualBox:~/OS/Lab3$
```

```
Mar 19 15:38:48 may-VirtualBox kernel: [ 496.034404] Device class created succe
ssfully
Mar 19 15:50:12 may-VirtualBox kernel: [ 1179.576028] mydevice has been opened 1
time(s)
Mar 19 15:50:26 may-VirtualBox kernel: [ 1193.390934] Received 20 characters fro
m the user
Mar 19 15:50:29 may-VirtualBox kernel: [ 1197.143622] Sent 20 characters to the
user
Mar 19 15:50:29 may-VirtualBox kernel: [ 1197.143745] mydevice closed successfu
lly.
```

图 3-8 测试

右边的内核输出信息可以看到，功能执行正确。

### 7.删除模块

```
may@may-VirtualBox:~/OS/Lab3$ sudo rmmod mydevice
may@may-VirtualBox:~/OS/Lab3$
```

```
Mar 19 15:53:28 may-VirtualBox kernel: [ 1376.018762] mydevice removed successfu
lly!
```

图 3-9 删除模块

可以看到内核输出删除成功信息，并且此时使用 `lsmod` 查看模块也可以看到 `mydevice` 模块已经不存在了。

## 3.4.2 实验调试及心得

### 1.实验调试

(1)

**故障：**测试到邻接字符数时，程序出现问题。

**原因：**在写入时，没有考虑到最长长度。

**解决方案：**写入的字节数应该是输入长度与最大长度中的最小值。最大长度的设定应该为缓冲区最大长度减 1，这个 1 是为了预留给 NULL 的。

(2)

**故障：**在读写时，程序出现问题。

**原因：**使用了 `sprintf` 函数，在内核区应尽量避免这类函数的使用。

**解决方案：**去掉 `sprintf` 函数，改为 `copy_to_user` 和 `copy_from_user` 函数。

(3)

**故障：**运行测试程序时出现问题。

**原因：**所插入的设备只能由超级用户使用，因此需要 `root` 权限执行测试程序。

**解决方案：**使用 `sudo` 指令。其实出现这个问题时，也上网搜索了一下解决方案，发现确实存在解决方案，但是由于知识有限，并不是很理解这种做法的原理，因此就直接采用 `sudo` 方式测试设备。

### 2.实验心得

通过这次实验，首先就是了解到了内核模块这个概念。由于 `linux` 内核是一个整体结构，因此修改内核变得十分困难，就像实验二中的，添加一个系统调用都需要重新编译一遍内核，十分麻烦，不能做到即插即用。所以为了解决这个问题，引入了模块机制。这样的好处是做到了即插即用，动态增加、删除，十分方便。

重要的就是掌握了简单设备的编写方法。实现了一些基本操作，也了解到内核模块应该调用内核函数，尽量避免使用其他库提供的函数。

同时，在实验基础上，这里我也实现了自动注册主设备号的功能，虽然实现起来有一点麻烦，但是是值得的。因为这样就有了一定了容错率，如果你把主设备号定死了，那么不同版本的 `linux` 上运行你的设备就有可能有问题，同时在不同的 `linux` 机器上运行你的设备也很有可能有问题。因此这个实现是十分必要的。

同样地，在实验的基本要求中，并没有考虑到字符设备的争用问题。字符设备应该是一个互斥使用的设备，不然的话，就会出现信息丢失、覆盖的问题。因此在此次实验中，我加入了信号灯的使用，这个方法有效地解决了资源争用的问题，具体实现方法见上。



同时，也了解到一般的 `makefile` 是如何编写的，这次编写模块的经历，让我想到在内核编译的过程中，模块也是一步一步地链接进去的，与这次实验紧密结合。

在编写过程中遇到的问题也都通过查找资料解决了，这样加深了我的印象，也丰富了我在此方面的知识。

## 附录 实验代码

```
/**
 * @file    mydevice.c
 * @author  may.sincerity
 * @date    6 March 2019
 * @version 0.1
 * @brief    A simple character device module
 */

#include <linux/init.h>          //__init __exit
#include <linux/module.h>
#include <linux/device.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/uaccess.h>       //copy to user
#include <linux/mutex.h>
#define  DEVICE_NAME "MyCharDev"
#define  CLASS_NAME "Char"

//version control  ...it seems that it is not necessary
#if CONFIG_MODVERSIONS == 1
#define MODVERSIONS
#include <linux/modversions.h>
#endif

MODULE_LICENSE("GPL");
MODULE_AUTHOR("may sincerity");
MODULE_DESCRIPTION("A simple linux char device");
MODULE_VERSION("0.1");

static int    majorNumber;
static char    message[1024] = {0};
static short  sizeOfMessage;
static int    devOpenTimes = 0;
```

```

static DEFINE_MUTEX(mydeviceMutex); //declare a mutex for mydevice

static struct class* mydeviceClass = NULL;
static struct device* mydeviceDevice = NULL;

static int my_open(struct inode *inodep, struct file *filep);
static int my_release (struct inode *inodep, struct file *filep);
static ssize_t my_read (struct file *filep, char *buffer , size_t length, loff_t *offset);
static ssize_t my_write (struct file *filep, const char *buffer , size_t length, loff_t
*offset);

static const struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = my_open,
    .read = my_read,
    .write = my_write,
    .release = my_release
};

/**
 * defined in linux/fs.h
 */
static int my_open(struct inode *inodep, struct file *filep) {
    if(!mutex_trylock(&mydeviceMutex)){
        printk("mydevice is being used by another process.\n");
        return -1;
    }
    devOpenTimes++;
    printk("mydevice has been opened %d time(s)\n", devOpenTimes);
    return 0;
}

static int my_release (struct inode *inodep, struct file *filep) {
    mutex_unlock(&mydeviceMutex);
    printk("mydevice closed successfully.\n");
}

```

```

        return 0;
    }

/**
 * @param buffer write data
 * copy_to_user ( * to, *from, size)
 */
static ssize_t my_read (struct file *filep, char *buffer , size_t length, loff_t *offset) {
    size_t bToCp = length > sizeofMessage ? sizeofMessage : length;
    size_t bNotCp = 0;
    if(!bToCp) return 0;
    bNotCp = copy_to_user(buffer, message, bToCp);

    if(bToCp - bNotCp)
    {
        printk("Sent %zu characters to the user\n",bToCp - bNotCp);
    }

    if(bNotCp){
        printk("Failed to send %zu characters to the user\n", bNotCp);
        return -1;
    }
    sizeofMessage = 0;
    return bToCp;
}

static ssize_t my_write (struct file *filep,const char *buffer , size_t length, loff_t
*offset) {
    const size_t maxLen = 1024 - 1;//NULL char
    size_t bToCp = length > maxLen ? maxLen: length;
    size_t bNotCp = 0;
    bNotCp = copy_from_user(message, buffer, bToCp);
    sizeofMessage = strlen(message);
    printk("Received %d characters from the user\n", sizeofMessage);
}

```

```

    if(bNotCp){
        printk("Failed to receive %zu characters!\n", bNotCp);
        return -1;
    }
    return bToCp;
}

```

```

static int __init MyCharDev_init(void){
    printk("Initializing the MyCharDev LKM\n");

    //get a majorNumber automatically
    majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
    if (majorNumber<0){
        printk("Failed to register a major number\n");
        return majorNumber;
    }
    printk("Registered successfully with major number %d\n", majorNumber);

    //device class
    mydeviceClass = class_create(THIS_MODULE, CLASS_NAME);
    if (IS_ERR(mydeviceClass)){
        unregister_chrdev(majorNumber, DEVICE_NAME);
        printk("Failed to register device class\n");
        return PTR_ERR(mydeviceClass);
    }
    printk("Device class registered successfully\n");

    //device driver
    mydeviceDevice = device_create(mydeviceClass, NULL, MKDEV(majorNumber,
0), NULL, DEVICE_NAME);
    if (IS_ERR(mydeviceDevice)){
        class_destroy(mydeviceClass);
        unregister_chrdev(majorNumber, DEVICE_NAME);
    }
}

```

```

        printk("Failed to create the device\n");
        return PTR_ERR(mydeviceDevice);
    }
    printk("Device class created successfully\n");

    mutex_init(&mydeviceMutex);

    return 0;
}

/** __exit
 *   only used in LKM
 */
static void __exit MyCharDev_exit(void){
    device_destroy(mydeviceClass, MKDEV(majorNumber, 0));
    class_unregister(mydeviceClass);
    class_destroy(mydeviceClass);
    unregister_chrdev(majorNumber, DEVICE_NAME);
    printk("mydevice removed successfully!\n");
}

module_init(MyCharDev_init);
module_exit(MyCharDev_exit);

//测试文件
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<string.h>
#include<unistd.h>

#define BUFFER_SIZE 1024
static char receive[BUFFER_SIZE];

```

```

int main(){
    int ret, fd;
    char stringToSend[BUFFER_SIZE];
    printf("Testing mydevice...\n");
    fd = open("/dev/MyCharDev", O_RDWR);
    if (fd < 0){
        perror("Failed to open mydevice.\n");
        return -1;
    }
    printf("Type a string:\n");
    scanf("%[^\n]*c", stringToSend); // %*c ignore the trailing character
    printf("Sending message: [%s].\n", stringToSend);
    ret = write(fd, stringToSend, strlen(stringToSend));
    if (ret < 0){
        perror("Failed to write to mydevice.\n");
        return -1;
    }

    printf("Press ENTER to continue.\n");
    getchar();

    ret = read(fd, receive, BUFFER_SIZE);
    if (ret < 0){
        perror("Failed to read from mydevice.\n");
        return -1;
    }
    printf("Receiving message: [%s]\n", receive);
    printf("Test Complete.\n");
    return 0;
}

```

//makefile

obj-m+=mydevice.o

all:

```
make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules  
$(CC) mydevice_tb.c -o test
```

clean:

```
make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean  
rm test
```



## 4 实验四 Linux 系统监控器

### 4.1 实验目的

- 1.了解和掌握/proc 文件系统的特点和使用方法

### 4.2 实验内容

- 1.了解/proc 文件的特点和使用方法
- 2.监控系统状态，显示系统部件的使用情况
- 3.用图形界面监控系统状态，包括系统基本信息、CPU 和内存利用率、所有进程信息等(可自己补充、添加其他功能)

实现的功能**具体包括**：

- (1)获取并显示主机名
- (2)获取并显示系统启动的时间
- (3)显示系统到目前为止持续运行的时间
- (4)显示系统的版本号
- (5)显示 cpu 的型号和主频大小
- (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)用新线程运行一个其他程序
- (14)关机功能

## 4.3 实验设计

### 4.3.1 开发环境

硬件：Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz

RAM 8GB

软件：Windows 10 下 VirtualBox 虚拟机 Ubuntu 16.04 64 位

Linux kernel: 4.15.0-45-generic

QT Creator 5.3

### 4.3.2 实验设计

下图为总体结构图。

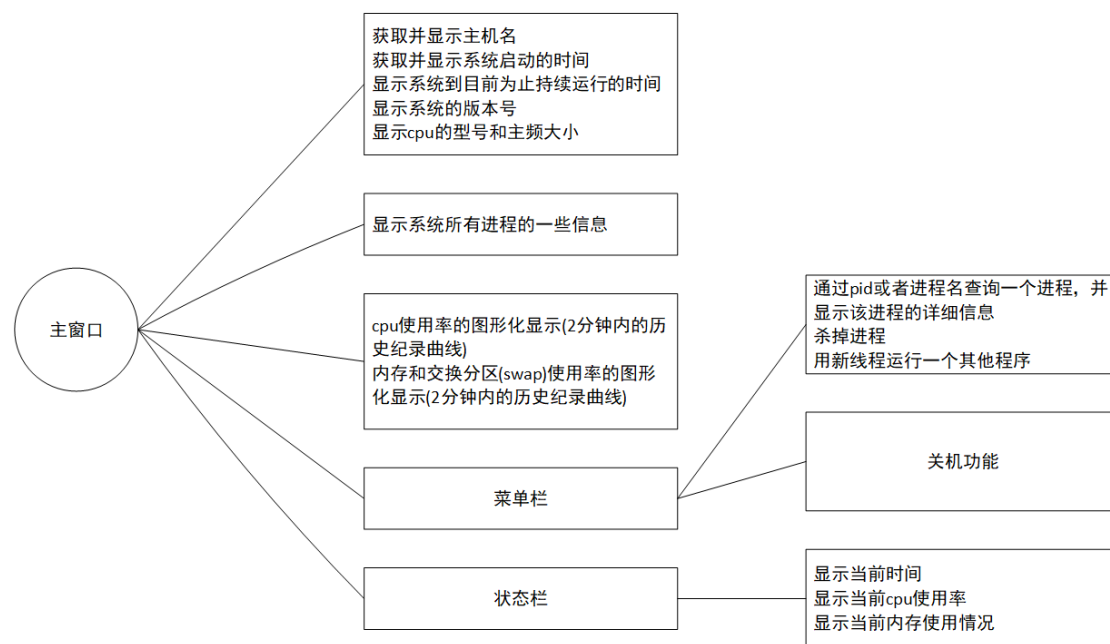


图 4-1 总体结构图

1.定义两个类 `SystemInfo` 和 `ProcInfo`，并且定义所需的成员，用来存储相关信息。

2.获取系统基本信息

首先创建一些 `labels`，用来显示文本信息。获取信息的函数定义为 `MainWindow` 类的成员函数 `getSystemInfo`。

(1) 获取主机名

根据主机名所在文件目录`/proc/sys/kernel/hostname`，利用 `ifstream` 打开文件，将文件的信息通过标准流输入进 `systemInfo` 相应的成员。

#### (2) 获取启动时间

所需要的相关信息都在`/proc/uptime` 中，`uptime` 中保存的是运行时间，因此先使用 `time` 函数获取当前的系统时间，然后用当前时间减去 `uptime` 就可以获得启动时间。然后利用 `gmtime` 函数转换成直观的时间。注意用 `gmtime` 转换时间的时候，转换成的是标准时间，因此在小时上要加上 8 才能变成北京时间。

#### (3) 获取持续运行时间

`proc/uptime` 中保存的是运行时间，单位为 s，因此运用一些简单的运算即可将这个数据转换成天、小时、分、秒这样直观的显示。

#### (4) 获取内核版本

`/proc/sys/kernel/ostype` 中存放的操作系统的类型（本实验中为 Linux）

`/proc/sys/kernel/osrelease` 中存放的是内核版本

因此只需要直接读文件即可。

#### (5) 获取 CPU 信息

Cpu 的信息在`/proc/cpuinfo` 中。这个文件中的信息都很直观，每一行前面表示是什么类型的信息，后面紧跟着就是内容。因此循环读入数据，当读入的信息为 `name` 的时候，接下来的信息就是 CPU 型号的信息。因为 CPU 型号一般都是由几个间隔的单词组成的。所以判断条件就是当读入的信息不是下一行首个单词时，继续执行循环。同理，获取 CPU 的主频也可以利用相同的方法，`MHz` 后面的信息就是 CPU 的主频达效。

除此之外，对于多核的情况，应该将 `core` 数量显示出来，因为本机 CPU 都是统一类型的，因此这里只需要判断 `processor` 的数量即可获取 `core` 数量。每当读到一个 `processor`，`core` 数就加一，直到读完。

### 3.更新系统基本信息

调用 `getSystemInfo`，然后将对应 `label` 的文本设置成需要的信息即可。上文说过，这些信息都存储在 `systemInfo` 这个类里面。然后利用信号和槽技术，设置定时器，每秒自动更新。其实在设计的时候只需要实时更新系统持续运行时间即可，这样可以起到一定的优化作用。

### 4.显示系统进程信息

#### (1) 定义获取信息的函数 `getProcInfo`

通过上学期实验学到的 Linux 文件目录内容。在这里可以用类似的解决办法来获取所有进程的信息。流程图如下：

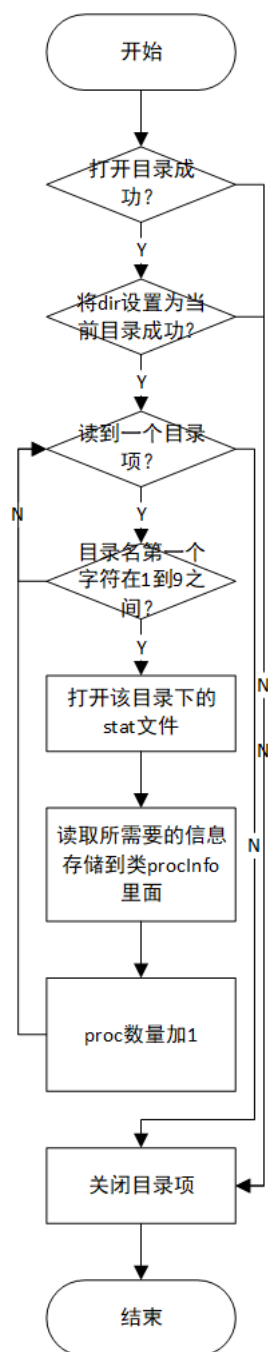


图 4-2 获取进程信息函数流程图

在流程图里，有些细节在下面阐述。

在读取信息这一步，就是确定所需要的信息在文件的哪个位置，然后读取。在调试的过程中，发现了一些特例需要进行处理。首先就是进程名的特例，一般进程名都是一对括号，然而还有((sd-pam))这种，对于这类，我进行了处理。具体的处理方法就是利用 string 的 find 方法，`procName.substr(procName.find('('),1)==temp.npos?1:2,procName.find(''))-(procName.find('('),1)==temp.npos?1:2))`，这段代码中 `procName.find('('),1)==temp.npos?1:2`，表明从第一个位置搜索 '('，如果没有搜索到，子串从第一个位置开始，如果搜索

到了，子串从第二个位置开始。范括号的处理方法也是类似的，在此不再赘述。然后就是一般的进程名是不带括号的，而 Web Content 这类就是例外，为了解决这类问题，就直接采用 if 判断语句，如果刚才读到的这个数据不包含')'，那么就继续读入一个数据，这两个数据才组成进程名的信息。

解决了进程名的问题，接下来就是将内存占用转换成以 MB 为单位的数据。直接将数据除以 1024，然后利用 sprintf 函数将想要的字符串输入进 procInfo 里的成员即可。

## 5.更新进程信息

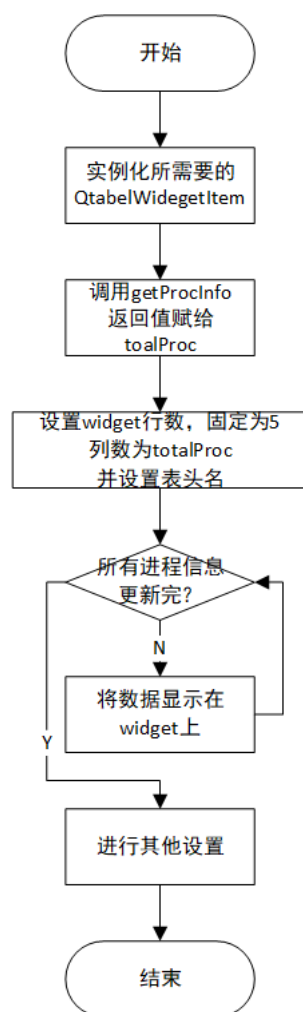


图 4-3 更新进程信息函数流程图

在上述流程图中，设置行列以及表头需要用到 TabelWidget 的方法 setColumnCount, setRowCount, setHorizontalHeaderLabels。将数据显示需要用到 setItem 方法，在这里将 string 转换成 QString 用到了 QString::fromStdString 方法，然后还使用了 setTextAlignment 将文本设置为居中。

流程图中进行其他设置指的是表自动适应窗口大小等。

setEditTriggers(QAbstractItemView::NoEditTriggers)表明不允许修改表项内容，setSelectionBehavior ( QAbstractItemView::SelectRows) 表明一次选中一行，horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch)这条语句实现了表格自动适应窗口大小的功能。

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，用新线程运行一个其他程序

我是通过菜单栏的选项进入一个新的窗口来实现这些功能的。

利用 QMenu 生成 File menu, 然后在这个 menu 下面使用 addAction 方法创建按钮 Search Proccess。然后利用 QMenuBar 生成菜单栏，加入 File menu。然后利用信号和槽技术，当有一个按钮被按下时，执行相应的操作。

如果点击 Search Proccess 按钮，就会弹出一个新的窗口 procDlg。

首先在输入文本框里利用 setPlaceholderText 方法显示提示信息，然后使用水平分离器的 setStretchFactor 方法设置合适的比例。同样也是利用信号和槽技术，将 timer 定时器和搜索进程这个槽相连接。设置定时器时间为 1s，这样可以提供一定的便利性，不用去点击 search 按钮，即可自动显示。这样也保证了一定的实时更新性。

#### (1) searchProcess

搜索其实就是一个遍历的过程，利用 procInfo 可以很方便的获取进程信息。流程图如 4-4:

搜索进程信息只需要在显示所有进程信息的基础上修改一下即可。首先要解决的就是同名进程的问题，在这里我采用的办法是全搜索，找到要找的进程时，将它的序号存进一个数组，然后继续搜索。还用到了 bool 型变量来标志是否找到。如果找到了，就像之前那样显示信息即可。这里流程图上的进行其他设置也和更新进程信息相同。

#### (2) killProcess

这个功能的实现十分简单，但还是要考虑一些存在的问题。首先就是要根据用户输入的类型来决定杀死进程的指令，在这里分为两种（进程名和 pid）。首先需要调用 getProcInfo 函数获取总进程数量，返回值赋给 totalProc。搜索进程信息，看它的进程名或者 pid 是否与输入的相符，如果进程名匹配，那么就要用到 pkill 命令来杀死进程。如果是 pid 匹配，那么就要用到 kill 命令来杀死进程。

使用 QString("%1").arg()来将所需要输入的命令存入 command 里面。然后使用 system 函数，参数为 command.toLatin1().data()将其转换成命令行形式，这样就可以实现调用系统命令。这样就可以方便地根据用户输入的类型来采取不同的命令杀死进程。

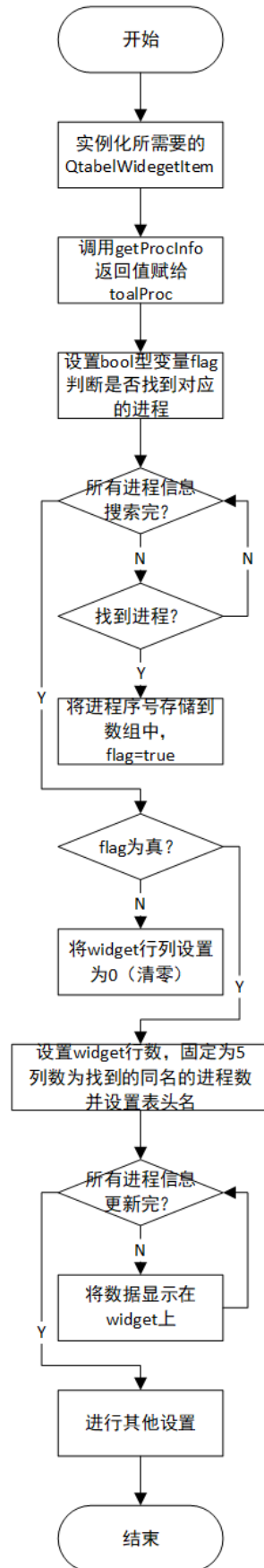


图 4-4 搜索进程信息函数流程图

### (3) createProcess

同样的,这个功能也需要考虑用户输入的多样性。参考 windows task manager,它的实现是通过输入文件的路径名来创建新任务的。在这里我采取了类似的方法。首先利用 Qprocess 实例化一个 pro。

如果输入的是文件的路径名,那么直接使用输入的信息作为参数,调用 pro 的 start 方法,即可创建新的任务。

如果输入的是现在运行的进程名,这里我查阅了一些资料,发现/proc/pid 文件目录下还有一个文件夹 cwd,这是一个链接,链接到当前进程的一个工作目录,因此只需要在这个目录里启动一个新任务即可,可以使用 sprintf 将要输入的参数设置成"/proc/%s/cwd/%s"类型。

(4) 设置三个按钮,分别实现三个对应的功能。也设置信号和槽,信号为点击按钮,槽就是上面的三个函数。

这样,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,用新线程运行一个其他程序的功能也就全部完成了。

## 7. cpu 使用率的图形化显示(2 分钟内的历史纪录曲线)以及内存和交换分区(swap)使用率的图形化显示(2 分钟内的历史纪录曲线)

### (1) 获取图形化显示所需要的信息

首先建立一个 QVariantList 的对象 itemList,用于存储信息。

#### (a) 获取 cpu 利用率相关信息

四个变量 oldCpuTotal, newCpuTotal, oldCpuIdle, newCpuIdle 用来存储新旧 cpu 总时间、空闲时间。由/proc/stat 文件信息可以得到 cpu 的总时间以及空闲时间。有 user、nicer 等等时间,将这些时间相加即可得到 cpu 的总时间,将 idle 时间赋给 newCpuIdle,即可完成函数功能。

#### (b) 获取 memory 利用相关信息

/proc/meminfo 文件清晰地显示了内存的各种信息。在这里我使用 QFile 建立一个对象 meminfo,在这里可以使用匹配的功能。建立一个 QRegularExpression 的对象 matchMessage,然后使用 setPattern 方法设置要匹配的目标文本。比如目标文本为"MemTotal:(.\*) kB\n"这里 (.\* )表示任意文本,然后使用 match 方法匹配到这一行语句,之后使用 captured 方法获取中间那一段数据,之后使用 toUInt 转换成无符号整型数。

类似地,可以获取到其他需要的数据,存储在 itemList 即可。

#### (c) 刷新信息



调用 (a) (b) 两个函数，然后令  $total = newCpuTotal - oldCpuTotal$ ， $idle = newCpuIdle - oldCpuIdle$ 。前者表示 cpu 总时间，后者表示 cpu 空闲时间。这样  $(total - idle) / total$  就可以得到 cpu 的利用率了。

然后设置信号 `updateCurve`，当 `itemList` 改变时，发送信号。

## (2) 绘制曲线图

这里我采用 `QCustomPlot` 类来绘制曲线图。

### (a) 自动适应窗口大小

先直接调用 `QCustomPlot::resizeEvent(event)`，然后要将自己图形上的 `label` 的位置重新设置，调用 `label` 的 `move` 方法，比如左上角的设置为 `(0,0)`，右下角的设置为 `(event->size().width() - infoLabel[3]->width(), event->size().height() - infoLabel[3]->height())`，这个语句的含义就是曲线的宽度减去 `label` 自身的宽度，以及曲线的高度减去 `label` 自身的高度。

### (b) 设置曲线左上角的名字

直接利用 `label` 的 `setText` 方法设置名字即可。

### (c) 设置曲线右上角的刻度

直接利用 `label` 的 `setText` 方法设置刻度，参数为 `QString("%1 %2").arg(yAxis2->range().upper).arg(calibration)`，其中 `yAxis2->range().upper` 表示刻度的上限，`calibration` 表示单位。

### (d) 设置最大时间

因为本次实验要求 120s，为了程序的通用性，加入了这个函数。

首先利用 `clear` 清除点，然后加入 121 个点，之后将 `usage` 的点数也设置为 121，然后将 x 轴的范围设置为 `(0, max)`，将左下角的 `label` 设置为 120s。

### (e) 设置最大使用率

直接设置 y 轴的范围即可。

### (f) 设置曲线图颜色

参照 `windows task manager`，我将曲线中的栅格透明度设置较高，采用 `setAlpha(100)` 即可完成。然后将栅格的颜色设置成主体颜色，例如：  
`xAxis->grid()->setPen(QPen(color))`。

同样地设置 x、y、x2、y2 轴的颜色。

然后设置曲线的颜色也是如此，调用 `graph(index)->setPen(QPen(color))`，发现 `windows` 上曲线与轴的面积也有颜色，在这里只需要调用 `graph(index)->setBrush(QBrush(color))` 即可，同时可以自己设置透明度。

### (g) 增加数据点

首先 `usage` 向量删除最后一个点，然后向头添加需要的数据。然后调用 `graph(index)->setData(time, usage)` 加入数据，调用 `replot` 重新绘制函数。

#### (h) 重新绘制函数

调用 `QCustomPlot` 的 `replot` 方法就行了。

做完上述工作，现在就是设置一些参数了。创建 4 个 `label`，设置宽度和高度以及颜色。然后设置 `label` 的对齐方式，以及坐标轴是否可见。在这里我隐藏了坐标轴的刻度，采用 `setTickLabels(false)` 方法，然后还要讲坐标轴的范围反向，调用 `setRangeReversed(true)`。然后设置小刻度的透明化，采用 `setTickPen(QPen(QColor(255, 255, 255, 0)))`，`setSubTickPen(QPen(QColor(255, 255, 255, 0)))` 方法即可完成。

最后需要调用 `addGraph()` 添加曲线图，以及 `graph(index)->addData(time, usage)` 添加数据点。

#### (3) 在主窗口里实例化

在 `MainWindow` 里实例化曲线图，其实就是设置之前实现的方法的参数，在这里我就不赘述了，详情见代码清单。

做完这些需要设置好信号和槽，当信息发生改变时，就要调用一次 `paintCurve` 函数。当定时器超时，就要刷新曲线。

8. 在状态栏显示当前时间、当前 `cpu` 使用率、当前内存使用情况、当前 `swap` 使用情况

在所需要功能的基础上，我还增加了 `swap` 的显示。其实状态栏的显示比较简单，因为在前面绘制曲线的时候就已经获取了相关信息，这里直接设置文本就好。

#### (1) 显示当前时间

利用 `QTime::currentTime` 获取当前时间，然后利用 `QTime` 的方法 `toString("hh:mm:ss")` 转换成 `xx-xx-xx` 的显示形式。

#### (2) 显示 `cpu` 利用率、内存使用情况、`swap` 使用情况

直接显示绘制曲线图时所用要的数据。

### 9. 关机功能

同样将关机功能的按钮设置在菜单栏，添加方法与 `Search Process` 相同，在

此不赘述。

当点击 shutdown 按钮后，为人性化考虑，会弹出一个警告窗口，这个直接用 QT 封装好的类就行了。比如：`QMessageBox::Yes == QMessageBox::warning(this,"Shutdown","Are you sure to continue?",QMessageBox::Yes|QMessageBox::Cancel)`。如果点击 Yes，那么调用 `QProcess::execute` 方法执行 `shutdown -h now` 指令即可关机。如果点击 Cancel，则会取消关机这个操作。

## 4.4 实验调试

### 4.4.1 实验步骤

1.获取并显示主机名、获取并显示系统启动的时间、显示系统到目前为止持续运行的时间、显示系统的版本号、显示 cpu 的型号和主频大小。

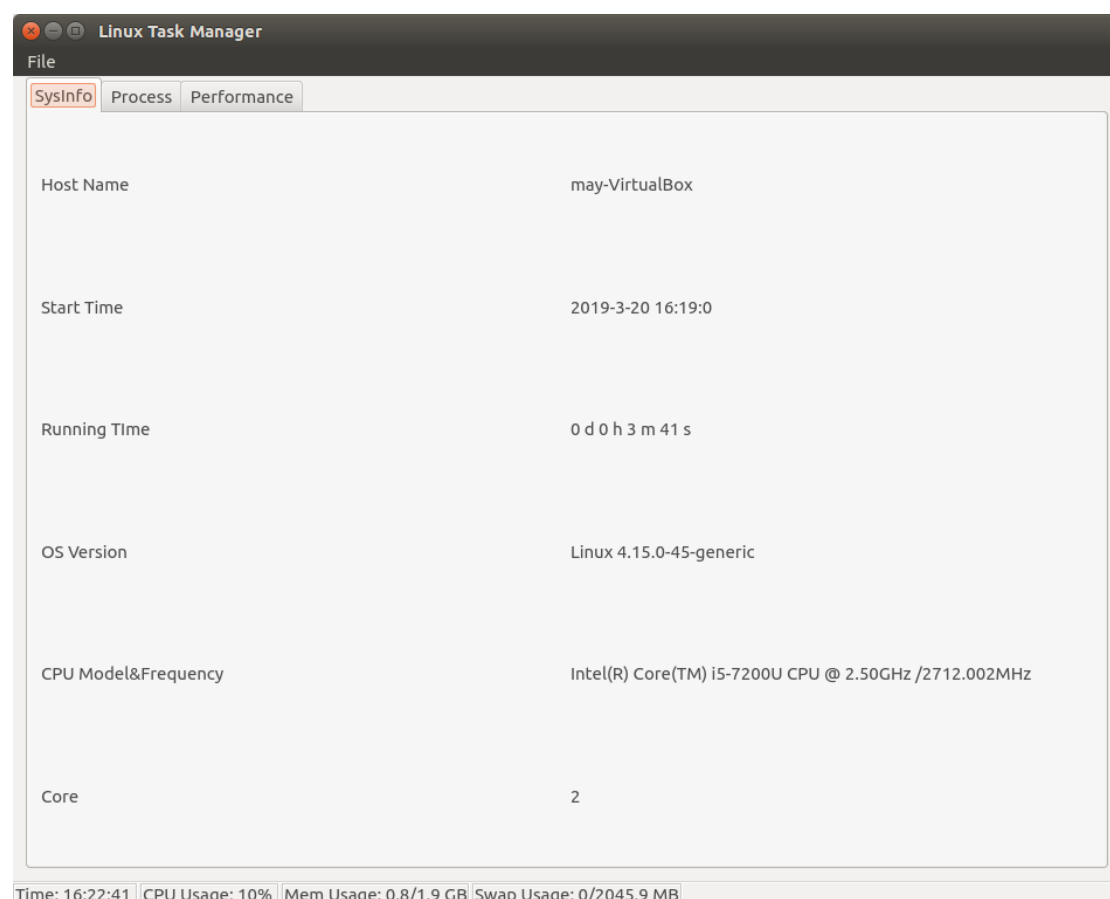
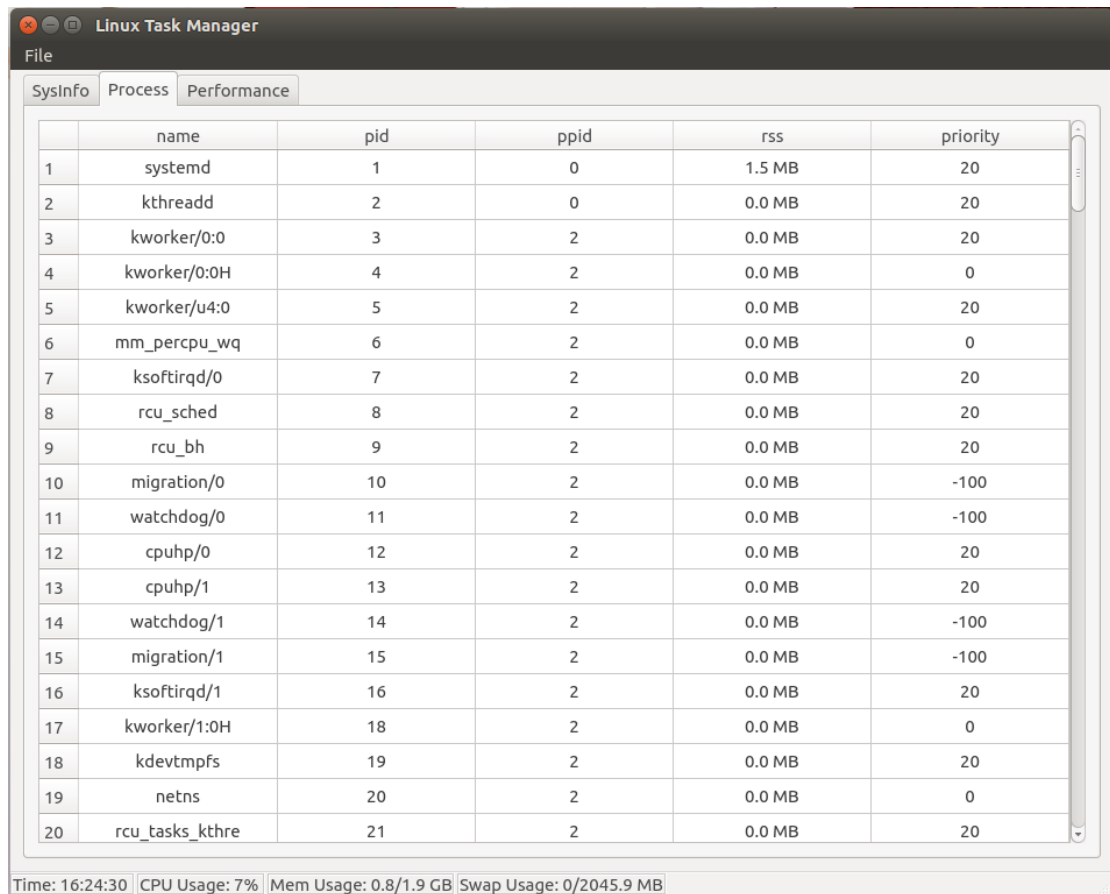


图 4-5 显示系统基本信息

2.显示系统所有进程的一些信息



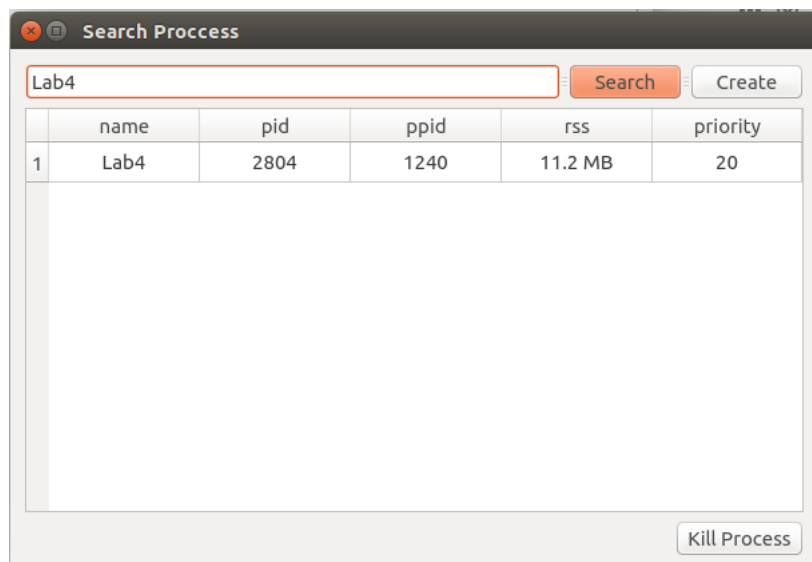
The screenshot shows the 'Linux Task Manager' application with the 'Process' tab selected. It displays a table of running processes with columns for name, pid, ppid, rss, and priority. The status bar at the bottom shows system metrics: Time: 16:24:30, CPU Usage: 7%, Mem Usage: 0.8/1.9 GB, and Swap Usage: 0/2045.9 MB.

	name	pid	ppid	rss	priority
1	systemd	1	0	1.5 MB	20
2	kthreadd	2	0	0.0 MB	20
3	kworker/0:0	3	2	0.0 MB	20
4	kworker/0:0H	4	2	0.0 MB	0
5	kworker/u4:0	5	2	0.0 MB	20
6	mm_percpu_wq	6	2	0.0 MB	0
7	ksoftirqd/0	7	2	0.0 MB	20
8	rcu_sched	8	2	0.0 MB	20
9	rcu_bh	9	2	0.0 MB	20
10	migration/0	10	2	0.0 MB	-100
11	watchdog/0	11	2	0.0 MB	-100
12	cpuhp/0	12	2	0.0 MB	20
13	cpuhp/1	13	2	0.0 MB	20
14	watchdog/1	14	2	0.0 MB	-100
15	migration/1	15	2	0.0 MB	-100
16	ksoftirqd/1	16	2	0.0 MB	20
17	kworker/1:0H	18	2	0.0 MB	0
18	kdevtmpfs	19	2	0.0 MB	20
19	netns	20	2	0.0 MB	0
20	rcu_tasks_kthre	21	2	0.0 MB	20

Time: 16:24:30 | CPU Usage: 7% | Mem Usage: 0.8/1.9 GB | Swap Usage: 0/2045.9 MB

图 4-6 显示进程信息

3.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  
如下图，通过进程名查询进程信息



The screenshot shows the 'Search Process' window. A search bar contains the text 'Lab4'. Below the search bar is a table with one row of process details. At the bottom right, there is a 'Kill Process' button.

	name	pid	ppid	rss	priority
1	Lab4	2804	1240	11.2 MB	20

Kill Process

图 4-7 查询进程

4.杀掉进程

如下图，通过 pid 杀掉新创建的 Lab4 进程。

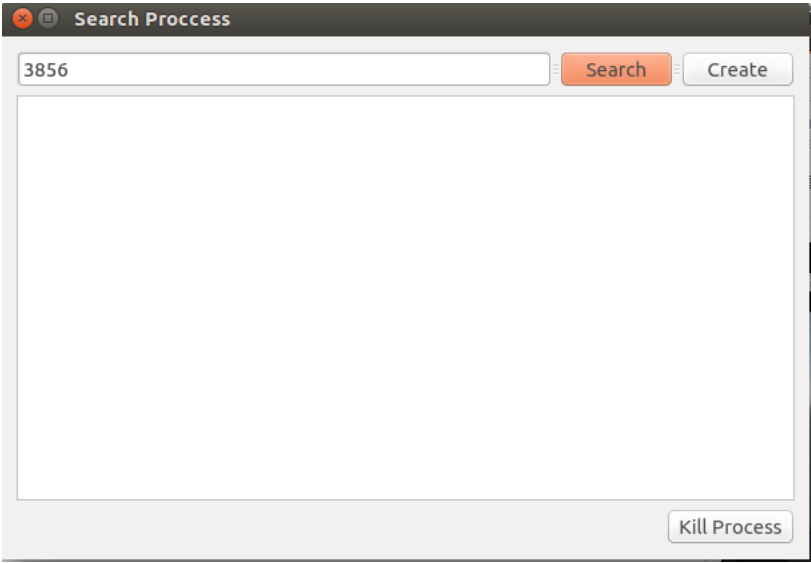


图 4-8 杀掉进程

5.cpu 使用率、内存使用率、swap 使用率的图形化显示

如图 4-8，在正常阶段，没有开什么应用的情况下的曲线。图 4-9 为开了视频之后的曲线情况，可以看到 swap 的使用情况。

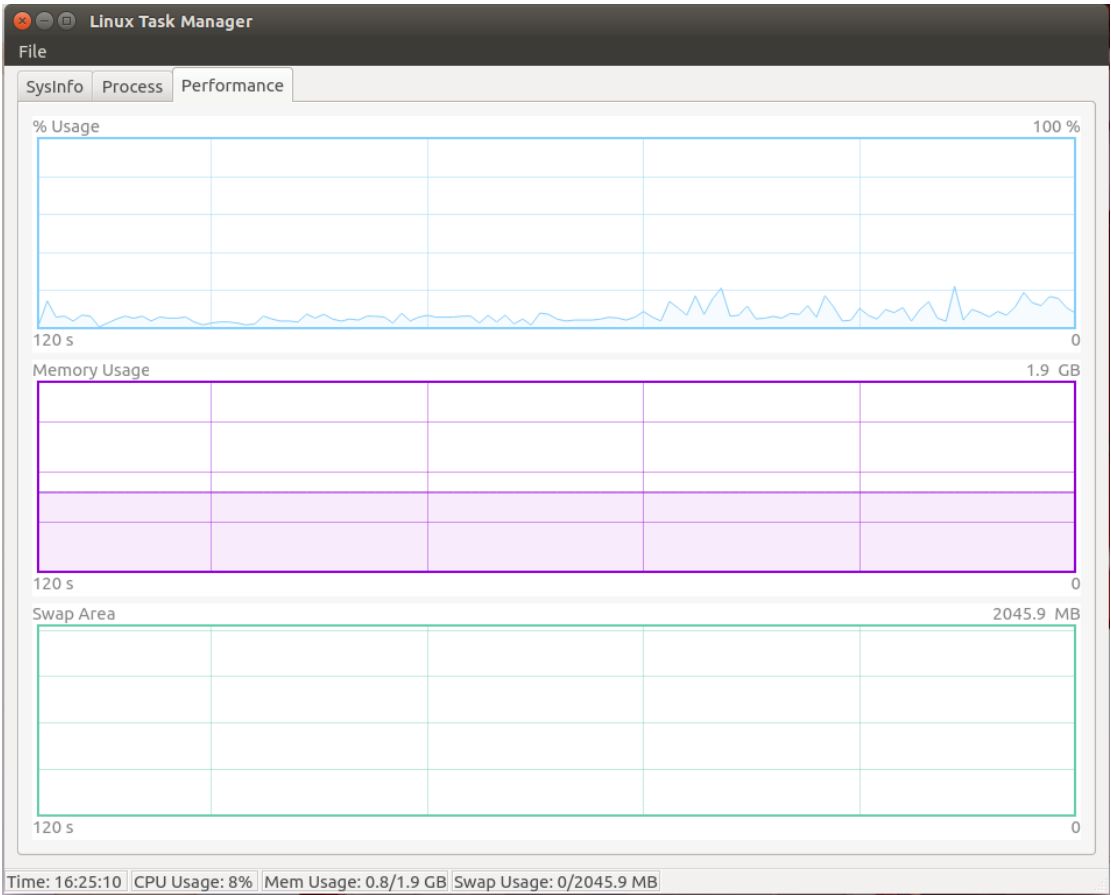


图 4-9 图形化显示

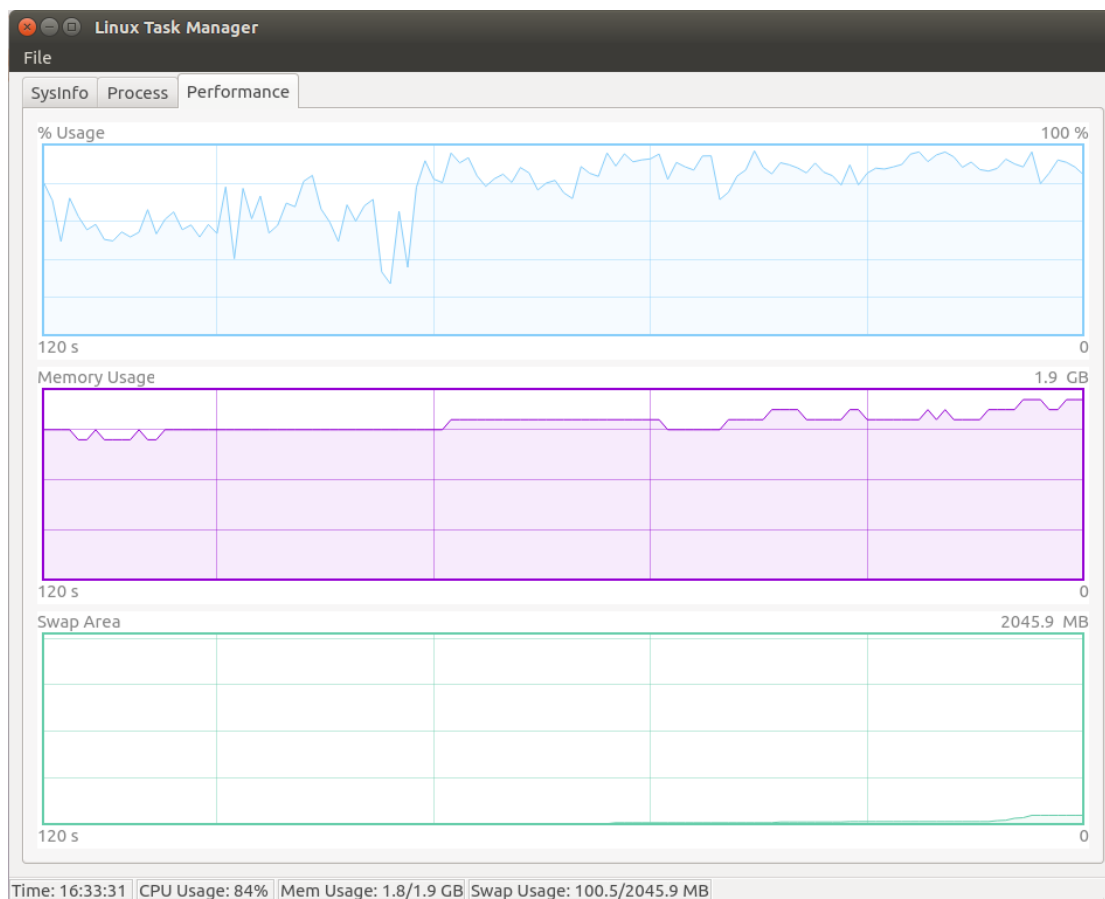


图 4-10 图形化显示

6. 状态栏显示当前时间、cpu 使用率、内存使用情况、swap 使用情况  
如下图，在状态栏可以看到显示情况。

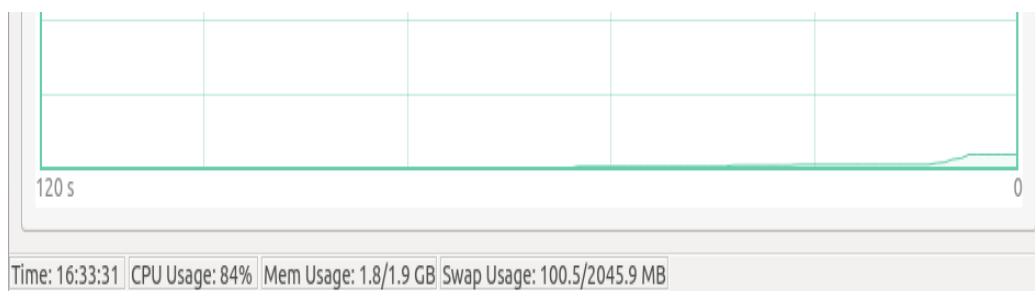


图 4-11 状态栏显示

7. 用新线程运行一个其他程序

如下图，根据进程名来创建新的进程。在这里还可以通过输入路径名来创建，在此没有展示。

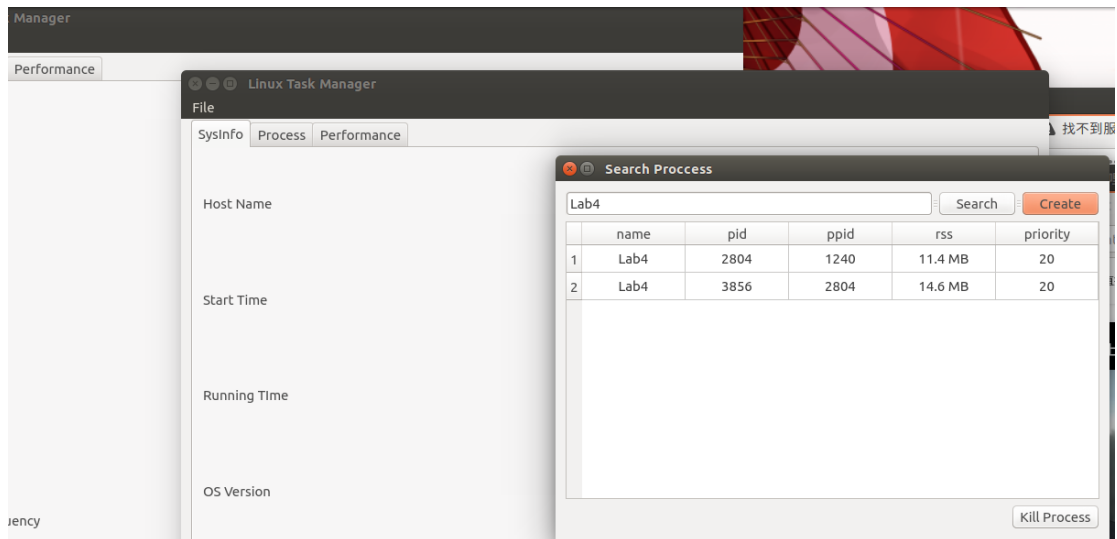


图 4-12 用新线程运行其他程序

## 8. 关机功能

可以看到有消息提示窗，接着是成功关机的界面。

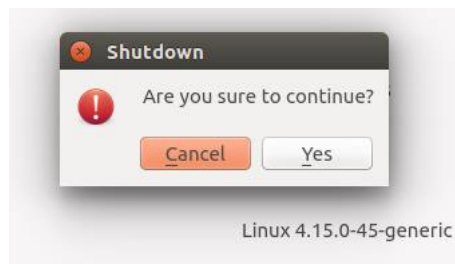


图 4-13 关机功能



图 4-14 关机功能

## 4.4.2 实验调试及心得

### 1.实验调试

(1) 故障：读取进程名问题，ppid 显示异常，为 S

原因：有些进程名有特例，比如 ((sd-pam)) 以及 (Web Content)

解决方案：经过逻辑思考，解决了特例问题。具体的处理方法就是利用 string.find 方法,procName.substr(procName.find('(',1)==temp.npos?1:2,这段代码中 procName.find('(',1)==temp.npos?1:2,表明从第一个位置搜索',如果没有搜索到,子串从第一个位置开始,如果搜索到了,子串从第二个位置开始。范括号的处理方法也是类似的,在此不再赘述。

(2) 故障：搜索同名问题只能显示一个

原因：没有搜索完就跳出循环了

解决方案：由于之前思考问题没有很完善，找到一个进程就直接跳出循环了。后来全部遍历一遍找到所有进程即可。

(3) 故障：创建进程问题

原因：不能满足输入的多样性

解决方案：一开始只是根据文件的绝对路径名来创建进程，后来查阅了资料，加入了根据进程名来创建进程的功能。解决方案就是将路径名设置为进程中的 cwd 目录，这样就可以根据这个链接目录建立进程。

(4) 故障：绘图问题

原因：不熟悉 QCustomPlot 类

解决方案：多多熟悉文档，查找有没有可以实现自己想要功能的函数。

### 2.实验心得：

在本次实验中，我了解了/proc 文件的特点以及使用方法。由 ppt 上了解到，PROC 文件系统是进程文件系统和内核文件系统的组成的复合体,它便于用户监控内核。而这些信息就呈现在一些特殊的文件中，有助于使用者观察。

在进行本次实验之前，由于之前并不是很熟悉 QT，所以学习 QT 花了一段时间。

感觉这次实验比较简单的就是获取系统基本信息，因为系统信息的/proc 文件都很清晰地显示出了各种数据，所以直接读取数据，将数据显示在 label 上就可以完成这一部分功能。除了系统的启动时间不能直接读取外，其他都可以。要获



取系统的启动时间，就需要获取系统当前时间以及系统运行的时间，将这两个时间相减，然后进行转换显示即可。实时显示就利用了信号和槽技术，这一技术在实验一的时候就已经掌握了，因此这里并没有太大的困难。

接下来的工作就比较有难度了，显示进程的信息，一开始并不知道怎么遍历全部文件。后来想到了在之前操作系统的实验课上做过遍历文件的实验，因此借鉴了方法，顺利地读取了全部进程的信息。然后通过进程里的 `stat` 文件，可以获取需要的信息，这一部分就上网查阅了资料，直到了每个数据的含义，之后显示全部信息就变得十分简单了。在这里比较难的是，一开始并不知道有 `tabelwidget` 这个方便的工具，对于显示信息感到不知如何下手，后来经过学习一段时间的 QT，就知道该如何合理运用各种工具了。

然后就是需要查询进程，杀死进程，创建进程的功能了。查询进程一开始想的就是遍历，因为对象是全局的，因此直接读取里面的信息就好。这里我学会了如何将不同类型的数据进行比较。杀死进程并不是简单的 `kill` 命令就能解决。在这里要考虑用户输入的多样性，根据不同的输入来选择不同的命令结束进程才是比较好的解决办法。创建进程也是一个需要考虑多样性的功能。一般来说需要输入文件的路径名来建立进程，这当然是一种比较规范的方法，这样就可以直接根据路径名找到执行文件，并利用 `QProcess` 即可。但是如果想直接输入已有进程的名字来创建进程，该怎么做呢。经过查找资料，发现进程文件夹里有一个 `cwd` 链接，链接到一个工作目录，因此我想到直接把路径名定成这里就可以解决问题了，后来发现真的可以解决。

接下来就是最难的部分，绘制曲线图。对于 QT 的不熟悉导致这一部分困难重重，后来好在上网查阅到一个类 `QCustomPlot` 可以较好实现绘图。查阅它的帮助手册，发现它的功能十分丰富。首先要绘制图形，就要获取相应的信息，`cpu` 利用率是需要计算的，其他的只需要直接读取就行。`Cpu` 利用率的计算也是上网查了资料的，发现只需要用总 `cpu` 时间减去 `cpu` 空闲时间然后除以总 `cpu` 时间即可。

然后设置曲线图的外观，详细的介绍已经在实验设计中介绍的很详细了。在这里不再赘述，由于时间紧迫加上学习成本，在这里尽可能实现地美观，与 `windows task manager` 尽可能相似。

在这次实验中主要就是熟悉了 `proc` 文件，了解到了各种文件携带的信息。其次最重要的就是学习了一段时间的 QT，经过一段时间，掌握了一些基本开发方法，增加了自己的经验，最后做出来这样一个程序，还是比较开心的。

## 附录 实验代码

```
#ifndef DATASTRUCTURE_H
#define DATASTRUCTURE_H

#include <string>

using namespace std;

class SystemInfo{
public:
    string hostName;
    string sysStartTime;
    string sysRunningTime;
    string kernVersion;
    string cpuModel;
    string coreNums;
};

class ProcInfo{
public:
    string name;
    string pid;
    string ppid;
    string rss;
    string priority;
};

#endif // DATASTRUCTURE_H

#ifndef GETCURVE_H
#define GETCURVE_H

#include <QObject>
```

```

#include "qcustomplot.h"

class myCurve : public QCustomPlot
{
    Q_OBJECT
private:
    QVector<double> time, usage;
    QList<QLabel *> infoLabel;
protected:
    virtual void resizeEvent(QResizeEvent *event);
public:
    myCurve(QWidget * parent = 0);
    void setCurveColor(const QColor & curveColor);
    void setPlotName(const QString & name);
    void setMaximumTime(unsigned int max);
    void setMaximumUsage(double max);
    void setUsageCalibration(const QString & calibration);
    void addData(double data);
    void replot();
};

#endif // GETCURVE_H

#include "getcurve.h"

myCurve::myCurve(QWidget * parent )
:QCustomPlot(parent), time(120), usage(120)
{
    axisRect()->setMinimumMargins(QMargins(0, 20, 0, 20));
    //labels
    for (int i = 0; i < 4; i++)
    {
        QLabel * label = new QLabel(this);
        label->show();
        label->setFixedWidth(100);
    }
}

```

```

        label->setFixedHeight(20);
        label->setStyleSheet("color:rgba(0,0,0,120);");
        infoLabel.append(label);
    }

    //axis
    infoLabel[0]->setAlignment(Qt::AlignLeft);
    infoLabel[2]->setAlignment(Qt::AlignLeft);
    infoLabel[1]->setAlignment(Qt::AlignRight);
    infoLabel[3]->setText("0");
    infoLabel[3]->setAlignment(Qt::AlignRight);
    xAxis2->setVisible(true);
    yAxis2->setVisible(true);
    //no tick lables
    xAxis->setTickLabels(false);
    xAxis2->setTickLabels(false);
    yAxis->setTickLabels(false);
    yAxis2->setTickLabels(false);
    //range reverse
    xAxis->setRangeReversed(true);
    //set tick/subtic transparent
    xAxis->setTickPen(QPen(QColor(255, 255, 255, 0)));
    xAxis->setSubTickPen(QPen(QColor(255, 255, 255, 0)));
    xAxis2->setTickPen(QPen(QColor(255, 255, 255, 0)));
    xAxis2->setSubTickPen(QPen(QColor(255, 255, 255, 0)));
    yAxis->setTickPen(QPen(QColor(255, 255, 255, 0)));
    yAxis->setSubTickPen(QPen(QColor(255, 255, 255, 0)));
    yAxis2->setTickPen(QPen(QColor(255, 255, 255, 0)));
    yAxis2->setSubTickPen(QPen(QColor(255, 255, 255, 0)));
    addGraph();
    int index = 0;
    graph(index)->addData(time, usage);
}

void myCurve::resizeEvent(QResizeEvent *event)
{

```

```

QCustomPlot::resizeEvent(event);//resize
//set labels
infoLabel[0]->move(0, 0);
infoLabel[1]->move(event->size().width() - infoLabel[1]->width(), 0);
infoLabel[2]->move(0, event->size().height() - infoLabel[2]->height());
infoLabel[3]->move(event->size().width() - infoLabel[3]->width(),
event->size().height() - infoLabel[3]->height());
}

```

```

void myCurve::setPlotName(const QString & name)
{
    infoLabel[0]->setText(name);
}

```

```

void myCurve::setUsageCalibration(const QString & calibration)
{

```

```

    infoLabel[1]->setText(QString("%1 %2").arg(yAxis2->range().upper).arg(calibration)
);
}

```

```

void myCurve::setMaximumTime(unsigned int max)
{
    time.clear();
    for (unsigned int i = 0; i <= max; i++){
        time.append(i);
    }
    //make time&usage same size
    usage.resize(max + 1);
    xAxis->setRange(0, max);
    xAxis2->setRange(0, max);
    infoLabel[2]->setText(QString("%1 s").arg(max));
}

```

```

void myCurve::setMaximumUsage(double max)

```

```

{
    yAxis->setRange(0, max);
    yAxis2->setRange(0, max);
}

void myCurve::setCurveColor(const QColor & curveColor)
{
    QColor color = curveColor;
    //transparent
    color.setAlpha(100);
    xAxis->grid()->setPen(QPen(color));
    yAxis->grid()->setPen(QPen(color));
    color.setAlpha(255);
    xAxis->setBasePen(QPen(color, 2));
    xAxis2->setBasePen(QPen(color, 2));
    yAxis->setBasePen(QPen(color, 2));
    yAxis2->setBasePen(QPen(color, 2));
    //set pen's color
    int index =0;
    color.setAlpha(255);
    graph(index)->setPen(QPen(color));
    //set brush's color
    color.setAlpha(20);
    graph(index)->setBrush(QBrush(color));
    replot();
}

void myCurve::addData(double data)
{
    usage.removeLast();
    usage.prepend(data);
    int index=0;
    graph(index)->setData(time, usage);
    replot();
}

```

```

void myCurve::replot()
{
    QCustomPlot::replot();
}

```

```

#include "mainwindow.h"
#include <QApplication>

```

```

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("Linux Task Manager");
    w.show();

    return a.exec();
}

```

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

```

```

#include <QMainWindow>
#include <QMenu>
#include <QMenuBar>
#include <QStatusBar>
#include <procdlg.h>
#include <QLabel>
#include "getcurve.h"
#include "showcurve.h"
#include "qcustomplot.h"

```

```

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    void paintCurve(const QVariantList & property);

private:
    Ui::MainWindow *ui;
    QMenu* menu[10];
    QAction* act[10];
    QMenuBar* menuBar ;
    QStatusBar* status ;
    QLabel *timeStatusLabel;
    QLabel *cpuStatusLabel;
    QLabel *memStatusLabel;
    QLabel *swapStatusLabel;
    curveObject performanceObject;
    void getSystemInfo();
    ProcDlg *procDlg;//new dialog

private slots:
    void updateSystemInfo();
    void updateProcInfo();
    void setInfoLabels();
    void refreshCurve();
    void triggerMenu(QAction* act);
};

```



```
#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"  
#include "ui_mainwindow.h"  
#include <QLabel>  
#include <QTimer>  
#include <iostream>  
#include <sstream>  
#include <fstream>  
#include <string>  
#include <time.h>  
#include <dirent.h>  
#include <QtDebug>  
#include <QDateTime>  
#include <unistd.h>  
#include <QPainter>  
#include "datastructure.h"
```

```
using namespace std;
```

```
SystemInfo systemInfo;  
ProcInfo procInfo[20000];
```

```
int getProcInfo();
```

```
MainWindow::MainWindow(QWidget *parent) :  
    QMainWindow(parent),  
    ui(new Ui::MainWindow)  
{  
    ui->setupUi(this);  
  
    timeStatusLabel = new QLabel(this);  
    cpuStatusLabel = new QLabel(this);
```

```

memStatusLabel = new QLabel(this);
swapStatusLabel = new QLabel(this);

//SystemInfo
getSystemInfo();
ui->hN_txt->setText(systemInfo.hostName.c_str());
ui->sT_txt->setText(systemInfo.sysStartTime.c_str());
ui->rT_txt->setText(systemInfo.sysRunningTime.c_str());
ui->oV_txt->setText(systemInfo.kernVersion.c_str());
ui->cpu_txt->setText(systemInfo.cpuModel.c_str());
ui->core_txt->setText(systemInfo.coreNums.c_str());

//Update information
QTimer *timer = new QTimer(this);

//system information
connect(timer,SIGNAL(timeout()),this,SLOT(updateSystemInfo()));
//cpu mem swap
connect(timer,SIGNAL(timeout()),this,SLOT(setInfoLabels()));
connect(&performanceObject,                &curveObject::updateCurve,this,
&MainWindow::paintCurve);
connect(timer,SIGNAL(timeout()),this,SLOT(refreshCurve()));
//ProInfo
connect(timer,SIGNAL(timeout()),this,SLOT(updateProcInfo()));

timer->start(1000);

//menubar
menu[0] = new QMenu("File");
menu[0]->addAction("Search Proccess");
menu[0]->addSeparator();
menu[0]->addAction("Shutdown");
menuBar = new QMenuBar(this);

```

```

        menuBar->addMenu(menu[0]);
        menuBar->setGeometry(0,0,2000,25);

connect(menuBar,SIGNAL(triggered(QAction*)),this,SLOT(triggerMenu(QAction*)))
; //react to click
}

void MainWindow::triggerMenu(QAction* act)
{
    if(act->text() == "Search Proccess")
    {
        procDlg = new ProcDlg(this);
        procDlg->setWindowTitle("Search Proccess");
        procDlg->show();
    }
    else if(act->text() == "Shutdown")
    {
        if(QMessageBox::Yes == QMessageBox::warning(this,"Shutdown","Are
you sure to continue?",QMessageBox::Yes|QMessageBox::Cancel)){
            QStringList arguments;
            arguments << "-h" << "now";
            QProcess::execute("shutdown", arguments);
        }
    }
}

/**update system information
*/
void MainWindow::updateSystemInfo(){
    getSystemInfo();
    ui->hN_txt->setText(systemInfo.hostName.c_str());
    ui->sT_txt->setText(systemInfo.sysStartTime.c_str());
    ui->rT_txt->setText(systemInfo.sysRunningTime.c_str());
    ui->oV_txt->setText(systemInfo.kernVersion.c_str());
}

```

```

        ui->cpu_txt->setText(systemInfo.cpuModel.c_str());
    }

/**update proc information
 */
void MainWindow::updateProcInfo(){
    QStringList headers;
    QTableWidgetItem *nameItem;
    QTableWidgetItem *pidItem;
    QTableWidgetItem *ppidItem;
    QTableWidgetItem *rssItem;
    QTableWidgetItem *priorityItem;
    int totalProc;//total num of proccesses
    totalProc=getProcInfo();
    ui->tableWidget->setColumnCount(5);
    ui->tableWidget->setRowCount(totalProc);
    headers<<"name"<<"pid"<<"ppid"<<"rss"<<"priority";
    ui->tableWidget->setHorizontalHeaderLabels(headers);
    for(int i=0;i<totalProc;i++)
    {
        nameItem=new
QTableWidgetItem(QString::fromStdString(procInfo[i].name));
        ui->tableWidget->setItem(i,0,nameItem);
        nameItem->setTextAlignment(Qt::AlignCenter);
        pidItem=new QTableWidgetItem(QString::fromStdString(procInfo[i].pid));
        ui->tableWidget->setItem(i,1,pidItem);
        pidItem->setTextAlignment(Qt::AlignCenter);
        ppidItem=new
QTableWidgetItem(QString::fromStdString(procInfo[i].ppid));
        ui->tableWidget->setItem(i,2,ppidItem);
        ppidItem->setTextAlignment(Qt::AlignCenter);
        rssItem=new QTableWidgetItem(QString::fromStdString(procInfo[i].rss));
        ui->tableWidget->setItem(i,3,rssItem);
    }
}

```

```

        rssItem->setTextAlignment(Qt::AlignCenter);
        priorityItem=new
QTableWidgetItem(QString::fromStdString(procInfo[i].priority));
        ui->tableWidget->setItem(i,4,priorityItem);
        priorityItem->setTextAlignment(Qt::AlignCenter);
    }
    ui->tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);//
cannot edit
    ui->tableWidget->setSelectionBehavior
( QAbstractItemView::SelectRows);//select a row

ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch
);;//fit automatically
}

/**get system information
*/
void MainWindow::getSystemInfo(){
    string info;
    string temp;

    //hostname
    ifstream if1("/proc/sys/kernel/hostname");
    if1>>info;
    systemInfo.hostName=info;
    if1.close();

    //start time
    time_t curTime;
    time_t startTime;
    long upTime;
    struct tm *ptm=NULL;
    char message[100];

```

```

ifstream if2("/proc/uptime");
if2>>upTime;
curTime = time(NULL);
if(curTime > upTime){
    startTime=curTime-upTime;
}
else{
    startTime=upTime-curTime;
}
ptm=gmtime(&startTime);
sprintf(message,"%d-%d-%d %d:%d:%d",ptm->tm_year + 1900,ptm->tm_mon
+ 1, ptm->tm_mday, ptm->tm_hour+8, ptm->tm_min, ptm->tm_sec);
info =message;
systemInfo.sysStartTime=info;
if2.close();

//running time
ifstream if3("/proc/uptime");
long runTime;
if3>>runTime;
sprintf(message,"%ld d %ld h %ld m %ld s",runTime / 86400,(runTime %
86400)/3600,(runTime % 3600)/60,runTime % 60);
info = message;
systemInfo.sysRunningTime=info;
if3.close();

//kernel version
ifstream if4("/proc/sys/kernel/ostype");
if4>>info;
if4.close();
ifstream if5("/proc/sys/kernel/osrelease");
if5>>temp;
if5.close();
systemInfo.kernVersion=info+" "+temp;

```

```

//cpu info
ifstream if6("/proc/cpuinfo");
while(if6>>info)
{
    if(info=="name")
    {
        if6>>info>>info;
        systemInfo.cpuModel = info;
        while(if6>>info&&info!="stepping")
        {
            systemInfo.cpuModel += " "+info;
        }
        continue;
    }
    if(info=="MHz")
    {
        if6>>info>>info;
        systemInfo.cpuModel += "/" + info + "MHz";
        break;
    }
}
if6.close();

//get the amount of cpu core
ifstream if7("/proc/cpuinfo");
int core_temp=0;
while(if7>>info)
{
    if(info=="processor")
    {
        core_temp++;
        continue;
    }
}
sprintf(message,"%d",core_temp);

```

```

        systemInfo.coreNums=message;
        if7.close();
    }

/**get proc information
 */
int getProcInfo(){
    DIR *dir;
    struct dirent *dirP;
    int i=0;
    if(!(dir=opendir("/proc"))){
        return 0;
    }
    while((dirP=readdir(dir))!=false)
    {
        if(dirP->d_name[0]>='1' && dirP->d_name[0]<='9')
        {
            string temp,procName;
            char pFileAddr[20];
            sprintf(pFileAddr,"/proc/%s/stat",dirP->d_name);
            ifstream infile(pFileAddr);
            infile>>(procInfo[i].pid);
            infile>>temp;
            procName=temp;
            if(temp.find('')==temp.npos){//solve problem like (Web Content)
                infile>>temp;
                procName=procName+" "+temp;
            }
            //solve problem like ((sd-pam))

            procInfo[i].name=procName.substr(procName.find('(',1)==temp.npos?1:2,procName.
            find('')-(procName.find('(',1)==temp.npos?1:2));
            infile>>temp;

```





```

{
    QString content(meminfo.readAll());
    temp.setPattern("MemTotal: (.*) kB\n");
    double totalMem = temp.match(content).captured(1).toDouble();
    totalMem = totalMem / 1024 / 1024;
    totalMem = (int)(totalMem * 10) / 10.0;
    ui->memUsgCurve->setMaximumUsage(totalMem);

    temp.setPattern("SwapTotal: (.*) kB\n");
    double totalSwap = temp.match(content).captured(1).toDouble();
    totalSwap = totalSwap / 1024 ;
    totalSwap = (int)(totalSwap * 10) / 10.0;
    ui->swapCurve->setMaximumUsage(totalSwap);
}
ui->memUsgCurve->setUsageCalibration(" GB");

//swap curve
ui->swapCurve->setCurveColor(QColor("#66CDAA"));
ui->swapCurve->setPlotName("Swap Area");
ui->swapCurve->setMaximumTime(120);
ui->swapCurve->setUsageCalibration(" MB");

}

void MainWindow::paintCurve(const QVariantList &property)
{
    //Cpu usage
    ui->cpuUsgCurve->addData(property[curveObject::CpuUsage].toDouble());
    //Mem info
    double totalMem = property[curveObject::MemoryTotal].toDouble();
    double availableMem = property[curveObject::MemoryAvailable].toDouble();
    double usedMem = totalMem - availableMem;
    usedMem /= 1024*1024;
    usedMem = (int)(usedMem*10)/10.0;

```

```

totalMem/= 1024*1024;
totalMem = (int)(totalMem*10)/10.0;
ui->memUsgCurve->addData(usedMem);
//swap info
double totalSwap = property[curveObject::SwapTotal].toDouble();
totalSwap /= 1024;
totalSwap = (int)(totalSwap*10)/10.0;
double freeSwap = property[curveObject::SwapFree].toDouble();
freeSwap /= 1024;
freeSwap = (int)(freeSwap*10)/10.0;
ui->swapCurve->addData(totalSwap - freeSwap);
//status bar
QTime time=QTime::currentTime();
QString currentTime;
currentTime=time.toString("hh:mm:ss");
timeStatusLabel->setText("Time: " + currentTime + " ");
cpuStatusLabel->setText("CPU          Usage:          "          +
QString::number(property[curveObject::CpuUsage].toUInt()) + "% ");
memStatusLabel->setText("Mem      Usage:      "      +      QString("%1/%2
GB").arg(usedMem).arg(totalMem));
swapStatusLabel->setText("Swap      Usage:      "+      QString("%1/%2
MB").arg(totalSwap - freeSwap).arg(totalSwap));
ui->statusBar->addWidget(timeStatusLabel);
ui->statusBar->addWidget(cpuStatusLabel);
ui->statusBar->addWidget(memStatusLabel);
ui->statusBar->addWidget(swapStatusLabel);
}

void MainWindow::refreshCurve()
{
    performanceObject.refresh();
}

MainWindow::~MainWindow()

```

```

{
    delete ui;
}

#ifndef PROCDLG_H
#define PROCDLG_H

#include <QDialog>
#include <datastructure.h>

namespace Ui {
class ProcDlg;
}

class ProcDlg : public QDialog
{
    Q_OBJECT

public:
    explicit ProcDlg(QWidget *parent = 0);
    ~ProcDlg();

private:
    Ui::ProcDlg *ui;
private slots:
    void searchProcess();
    void killProcess();
    void createProcess();
    void on_search_proc_clicked();
    void on_killProc_clicked();
    void on_createProc_clicked();
};

#endif // PROCDLG_H

```

```

#include "procdlg.h"
#include "ui_procdlg.h"
#include <iostream>
#include <QTimer>
#include <QProcess>
#include <fstream>
#include <sstream>
#include <QtDebug>
#include <QString>

using namespace std;

extern int getProcInfo();
extern ProcInfo procInfo[20000];

ProcDlg::ProcDlg(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::ProcDlg)
{
    ui->setupUi(this);

    ui->lineEdit->setPlaceholderText("Input pid(name) to search or path to create
process");
    ui->splitter->setStretchFactor(0,8);//set strechfactor

    QTimer *timer = new QTimer(this);

    connect(timer,SIGNAL(timeout()),this,SLOT(searchProcess()));

    timer->start(1000);
}

```

```

void ProcDlg::searchProcess()
{
    QString inMessage;
    QStringList headers;
    QTableWidgetItem *nameItem;
    QTableWidgetItem *pidItem;
    QTableWidgetItem *ppidItem;
    QTableWidgetItem *rssItem;
    QTableWidgetItem *priorityItem;
    inMessage=ui->lineEdit->text();
    int totalProc;
    totalProc=getProcInfo();
    bool flag = false;
    int i=0;
    int j=0;
    int temp[20];
    for(i=0;i<totalProc;i++)
    {
        if(inMessage==QString::fromStdString(procInfo[i].name)           ||
inMessage==QString::fromStdString(procInfo[i].pid)){
            flag=true;//find
            temp[j++]=i;
        }
    }
    if(!flag){
        i=10000;
        ui->tableWidget->setColumnCount(0);//clear
        ui->tableWidget->setRowCount(0);
    }
    else{
        ui->tableWidget->setColumnCount(5);
        ui->tableWidget->setRowCount(j);
        headers<<"name"<<"pid"<<"ppid"<<"rss"<<"priority";
        ui->tableWidget->setHorizontalHeaderLabels(headers);
        for(int k=0;k<j;k++){

```

```

        nameItem=new
QTableWidgetItem(QString::fromStdString(procInfo[temp[k]].name));
        ui->tableWidget->setItem(k,0,nameItem);
        nameItem->setTextAlignment(Qt::AlignCenter);
        pidItem=new
QTableWidgetItem(QString::fromStdString(procInfo[temp[k]].pid));
        ui->tableWidget->setItem(k,1,pidItem);
        pidItem->setTextAlignment(Qt::AlignCenter);
        ppidItem=new
QTableWidgetItem(QString::fromStdString(procInfo[temp[k]].ppid));
        ui->tableWidget->setItem(k,2,ppidItem);
        ppidItem->setTextAlignment(Qt::AlignCenter);
        rssItem=new
QTableWidgetItem(QString::fromStdString(procInfo[temp[k]].rss));
        ui->tableWidget->setItem(k,3,rssItem);
        rssItem->setTextAlignment(Qt::AlignCenter);
        priorityItem=new
QTableWidgetItem(QString::fromStdString(procInfo[temp[k]].priority));
        ui->tableWidget->setItem(k,4,priorityItem);
        priorityItem->setTextAlignment(Qt::AlignCenter);
    }
    ui->tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);
    ui->tableWidget->setSelectionBehavior
( QAbstractItemView::SelectRows);

    ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch
);;//fit automatically
    }
}

void ProcDlg::killProcess()
{
    QString inMessage;
    inMessage=ui->lineEdit->text();
    int totalProc;

```

```

totalProc=getProcInfo();
int killCmd=-1;

for(int i=0;i<totalProc;i++)
{
    if(inMessage==QString::fromStdString(procInfo[i].name)){
        killCmd=0;
        break;
    }
    if(inMessage==QString::fromStdString(procInfo[i].pid)){
        killCmd=1;
        break;
    }
}

QString command;
switch (killCmd) {
case 0:
    command=QString("pkill %1").arg(inMessage);
    system(command.toLatin1().data());
    break;
case 1:
    command=QString("kill %1").arg(inMessage);
    system(command.toLatin1().data());
    break;
default:
    break;
}

}

void ProcDlg::createProcess()
{
    QProcess *pro=new QProcess;
    QString newProc;
    newProc=ui->lineEdit->text();

```



```

char message[100];
int totalProc=getProcInfo();
int i;
for(i=0;i<totalProc;i++)
{
    if(newProc==QString::fromStdString(procInfo[i].name)           ||
newProc==QString::fromStdString(procInfo[i].pid))
        break;
}
if(i>=totalProc){
    i=10000;
    string temp;
    temp=newProc.toStdString();
    sprintf(message,"%s",temp.c_str());
}
else{

sprintf(message,"/proc/%s/cwd/%s",procInfo[i].pid.c_str(),procInfo[i].name.c_str());

}

newProc=QString::fromStdString(message);

pro->start(newProc);
}

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

void ProcDlg::on_search_proc_clicked()
{
    searchProcess();
}

```

```

void ProcDlg::on_killProc_clicked()
{
    killProcess();
}

void ProcDlg::on_createProc_clicked()
{
    createProcess();
}

```

```

#ifndef SHOWCURVE_H
#define SHOWCURVE_H

```

```

#include <QObject>
#include <QVariantList>
#include <QFile>
#include <QRegularExpression>
#include <QTime>

```

```

class curveObject : public QObject
{
    Q_OBJECT
private:
    QVariantList itemList;
    unsigned long oldCpuTotal;
    unsigned long newCpuTotal;
    unsigned long oldCpuIdle;
    unsigned long newCpuIdle;
    void refreshCpuUsage();
    void refreshMemoryInfo();

public:
    enum CurveInfo
    {

```

```
CpuUsage, MemoryTotal, MemoryAvailable, SwapTotal, SwapFree,  
infoNums
```

```
};  
curveObject(QObject * parent = 0);  
void refresh();
```

```
signals:  
    void updateCurve(const QVariantList & property);  
};  
#endif // SHOWCURVE_H
```

```
#include "showcurve.h"  
#include <QDebug>  
#include <iostream>  
#include <sstream>  
#include <fstream>
```

```
using namespace std;
```

```
curveObject::curveObject(QObject * parent)  
:QObject(parent)  
{  
    for(int i = 0; i < infoNums; i++){  
        itemList.append(0);  
    }  
    oldCpuTotal = newCpuTotal = 0;  
    oldCpuIdle = newCpuIdle = 0;  
}
```

```
void curveObject::refresh()  
{  
    refreshCpuUsage();  
    refreshMemoryInfo();
```

```

    unsigned long total = newCpuTotal - oldCpuTotal;
    unsigned long idle = newCpuIdle - oldCpuIdle;
    itemList[CpuUsage] = 100 * (float)(total - idle) / total;
    updateCurve(itemList);
}

```

```

void curveObject::refreshCpuUsage()

```

```

{
    oldCpuTotal = newCpuTotal;
    oldCpuIdle = newCpuIdle;
    newCpuTotal = 0;
    ifstream if1("/proc/stat");
    char temp[50];
    if1>>temp;
    if1>>temp;
    newCpuTotal += atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
    newCpuIdle = atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
    if1>>temp;
    newCpuTotal += atoi(temp);
}

```

```

        if l.close();
    }

void curveObject::refreshMemoryInfo()
{
    QRegularExpression matchMessage;
    QFile meminfo("/proc/meminfo");
    if(meminfo.open(QIODevice::ReadOnly))
    {
        QString content(meminfo.readAll());
        matchMessage.setPattern("MemTotal:(.*) kB\n");
        itemList[MemoryTotal] =
matchMessage.match(content).captured(1).toUInt();
        matchMessage.setPattern("MemAvailable:(.*) kB\n");
        itemList[MemoryAvailable] =
matchMessage.match(content).captured(1).toUInt();
        matchMessage.setPattern("SwapTotal:(.*) kB\n");
        itemList[SwapTotal] = matchMessage.match(content).captured(1).toUInt();
        matchMessage.setPattern("SwapFree:(.*) kB\n");
        itemList[SwapFree] = matchMessage.match(content).captured(1).toUInt();
    }
}

```