



华中科技大学

操作系统原理课程设计报告

姓 名：陈 一 超
学 院：计算机科学与技术
专 业：计算机科学与技术
班 级：CS1608
学 号：U201614719
指导教师：阳 富 民

分数	
教师签名	

2019 年 3 月 28 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
1.4 实验调试	2
附录 实验代码	4
2 实验二 增加系统调用	19
2.1 实验目的	19
2.2 实验内容	19
2.3 实验设计	19
2.4 实验调试	20
附录 实验代码	21
3 实验三 增加字符设备驱动	24
3.1 实验目的	24
3.2 实验内容	24
3.3 实验设计	24
3.4 实验调试	25
附录 实验代码	26
4 实验四 系统监视器(QT)	30
4.1 实验目的	30
4.2 实验内容	30
4.3 实验设计	30
4.4 实验调试	31
附录 实验代码	35

1 实验一 LINUX 用户界面的使用

1.1 实验目的

掌握 Linux 操作系统的使用方法，包括键盘命令、系统调用；
熟悉 Linux 下的编程环境。

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(使用系统调用 open/read/write...)；

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 实验设计

1.3.1 开发环境

操作系统：Windows10 系统 + VMware Workstation 下的 Ubuntu 虚拟机；
Ubuntu 版本号为 18.10，64 位；系统内核版本为 4.18.0-10-generic。

编程环境：

实验一在 vim 编辑器下编程，使用 Linux 内的 g++ 编译器编译。

实验二使用 Qt Creator 编程并编译运行。

1.3.2 实验设计

文件拷贝

- 1) 使用 read 函数进行取数据到一个数组 buf 里。
- 2) 使用 write 函数将 buf 里的数据取到文件里。
- 3) 用 access 函数判断是否存在某个文件。
- 4) 当文件存在时，则询问用户是否取消操作以及强制性操作。

三个并发进程：

- 1) 第一个进程计时器和显示当前系统时间。
- 2) 第二个进程显示日历。
- 3) 第三个进程显示钟表（带秒针）、每秒进行动态更新，使用 Qt 的绘画机制，对钟表进行绘画。并监听键盘输入，键盘输入 ESC 键退出进程。
- 4) 在 main.cpp 文件里 fork 出三个进程。

1.4 实验调试

1.4.1 实验步骤

文件拷贝

- 1) 安装 Ubuntu 虚拟机。
- 2) 使用 vim 编写程序代码，用 g++ 编译，并调试程序直至成功
- 3) 运行编译生成的文件拷贝程序，试图拷贝一个文件，并用 cmp 指令比较两个文件是否相同，若相同则实验完成，否则继续修改代码。

实验流程图如图 1-1 所示：

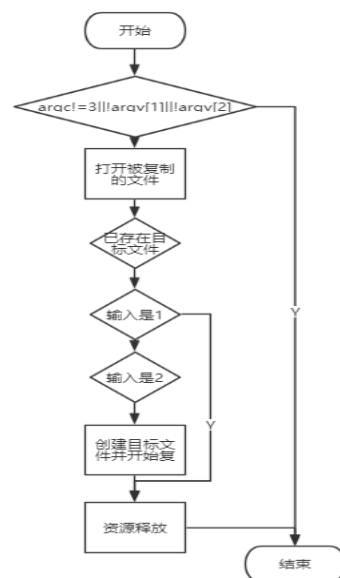


图 1-1 copy 文件流程图

三个并发进程：

- 1) 安装 Qt-Creator，配置 qt5 开发环境，创建 qt widget 项目。
- 2) 编写代码，用集成开发环境进行编译。

1.4.2 实验调试及心得

1. 文件拷贝

- 1) 用 g++ 编译程序代码，没有错误。
- 2) 复制成功：

```
cychust@ubuntu ~/u/p/o/p/p/labs> ./main tmp/a tmp/a_tmp  
cychust@ubuntu ~/u/p/o/p/p/labs>
```

图 1-2: 复制结果演示

按 1 取消操作：

```
cmake-build-debug/ CMakeLists.txt main* main.cpp tmp/  
cychust@ubuntu ~/u/p/o/p/p/labs> ./main tmp/a tmp/a_tmp  
Hello, World!  
file is exist  
1.取消操作  
2.强制删除并复制  
1  
cychust@ubuntu ~/u/p/o/p/p/labs>
```

图 1-3 取消操作结果演示

按 2 强制删除并复制：

```
cychust@ubuntu ~/u/p/o/p/p/labs> ./main tmp/a tmp/a_tmp  
file is exist  
1.取消操作  
2.强制删除并复制  
2
```

图 1-4 强制删除并复制操作结果演示

2. 三个并行进程实验

编译程序并运行结果如下：

结果：

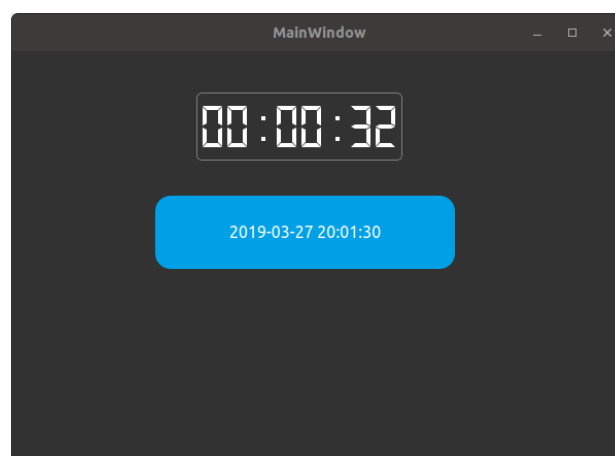


图 1-5 进程 1 的窗口结果演示

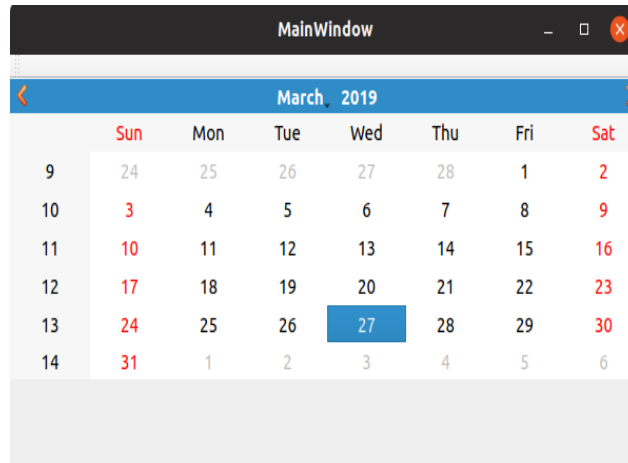


图 1-6 进程 2 的窗口结果演示



图 1-7 进程 3 的窗口结果演示

实验心得:

本次实验在设计过程中, 学习不同 Linux 不同的文件格式, 以及了解软链接和硬链接背后的实现过程。创建三个窗口, 即直接利用 `fork` 进行进程创建, 体验了进程的并发效果。

附录 实验代码

文件 copy:

```
#include <iostream>
#include <unistd.h> //read
```

```
#include <sys/types.h>
```

```

#include <sys/stat.h>
#include <fcntl.h>    //open
#include <cstring>

#define MAX_LINE_LEN 1000

#define OPEN_ERROR(x) ((x)===-1)

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

    char *file_path_from;
    char *file_path_to;

    char buf[MAX_LINE_LEN + 1];

    int fd_from;
    int fd_to;
    int flag;

    int select;

    if (argc != 3 || !argv[1] || !argv[2]) {
        fprintf(stderr, "Usage: %s <source file path> <target file path>\n", argv[0]);
        exit(-1);
    }
    file_path_from = argv[1];
    file_path_to = argv[2];

    fd_from = open(file_path_from, O_RDONLY);
    if (OPEN_ERROR(fd_from)) {
        fprintf(stderr, "open %s happen error:\n", argv[1]);
        perror("open");
        fprintf(stderr, "\t %s with %d\n", strerror(errno), errno);
        exit(-1);
    }

    if (!access(file_path_to, F_OK)) {        //文件存在
        printf("file is exist\n");
        printf(" 1 .取消操作\n2.强制删除并复制\n");
        scanf("%d", &select);
        while (select != 1 && select != 2) {
            printf(" 1 .取消操作\n2.清空文件并复制\n");
            scanf("%d", &select);
        }
    }
}

```

```

    }
    switch (select) {
        case 1:
            return 0;

        case 2:
            fd_to = open(file_path_to, O_TRUNC); //若文件存在并且以可写
            的方式打开时，此旗标会令文件长度清为 0，而原来存于该文件的资料也会消失.
            if (OPEN_ERROR(fd_to)) {
                fprintf(stderr, "create %s happen error\n", argv[2]);
                fprintf(stderr, "\t %s with %d\n", strerror(errno), errno);
                exit(-1);
            }
            break;
        default:
            break;
    }
} else {
    fd_to = open(file_path_to, O_RDONLY | O_WRONLY | O_CREAT);
    //新建文件
    if (OPEN_ERROR(fd_to)) {
        fprintf(stderr, "create %s happen error\n", argv[2]);
        // perror("create");
        fprintf(stderr, "\t %s with %d\n", strerror(errno), errno);
        exit(-1);
    }
}
while ((flag = read(fd_from, buf, MAX_LINE_LEN)) > 0) {
    write(fd_to, buf, flag);
}
close(fd_from);
close(fd_to);

return 0;
}

```

CommonHelper.h 文件代码:

```

#ifndef COMMONHELPER_H
#define COMMONHELPER_H

```



```

#include<QFile>
#include<QApplication>

class CommonHelper
{
public:
    static void setStyle(const QString &style) {
        QFile qss(style);
        qss.open(QFile::ReadOnly);
        qApp->setStyleSheet(qss.readAll());
        qss.close();
    }
};
#endif // COMMONHELPER_H

```

Mainwindows.h 文件代码

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:

```

```

        Ui::MainWindow *ui;
    };

#endif // MAINWINDOW_H

```

mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

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

```

mainwindow2.h 文件代码

```

#ifndef MAINWINDOW2_H
#define MAINWINDOW2_H

```

```

#include <QMainWindow>
#include<QTime>
#include<QTimer>
namespace Ui {
class MainWindow2;
}

```

```

class MainWindow2 : public QMainWindow

```

```

{
    Q_OBJECT

public:
    explicit MainWindow2(QWidget *parent = nullptr);
    ~MainWindow2();

private:
    Ui::MainWindow2 *ui;
    void initTime();
    void initStyleSheets();
    QTimer* timer;
    QTime *timeRecord;
public slots:

    void updateTime();

};

```

#endif // MAINWINDOW2_H

mainwindow2.cpp 文件代码

```
#include "mainwindow2.h"
```

```
#include "ui_mainwindow2.h"
```

```

MainWindow2::MainWindow2(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow2)
{
    ui->setupUi(this);
    this->ui->centralwidget->setFixedHeight(400);
    this->ui->centralwidget->setFixedWidth(600);
    initTime();
}

```

```

MainWindow2::~MainWindow2()
{

    delete ui;
    delete(this->timer);
    delete (this->timeRecord);
}

void MainWindow2::initTime(){
    this->timer=new QTimer();
    this->timeRecord=new QTime(0,0,0);
    this->ui->timer->setDigitCount(8);
    ui->timer->setSegmentStyle(QLCDNumber::Flat);
    ui->timer->display(timeRecord->toString("hh:mm:ss"));
    connect(timer,SIGNAL(timeout()),this,SLOT(updateTime()));

    timer->start(1000);
}

void MainWindow2::updateTime(){
    *timeRecord = timeRecord->addSecs(1);
    ui->timer->display(timeRecord->toString("hh:mm:ss"));
    QDateTime time = QDateTime::currentDateTime();
    QString str = time.toString("yyyy-MM-dd hh:mm:ss");
    ui->radiusBlueLabel->setText(str);
}

```

mainwindow3.h 文件代码

```

#ifndef MAINWINDOW3_H
#define MAINWINDOW3_H

#include <QMainWindow>
#include<QTimer>

```

```

#include<QDateTime>
#include<QPainter>
#include<QPainter>
#include<QKeyEvent>
namespace Ui {
class MainWindow3;
}

class MainWindow3 : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow3(QWidget *parent = nullptr);
    ~MainWindow3();

private:
    Ui::MainWindow3 *ui;
    int clockTick;
    void initTime();
    void initStyleSheets();
    QTimer* timer;
public slots:
    void updateTime();
protected:
    void paintEvent(QPaintEvent *);
    void keyPressEvent(QKeyEvent *);
};

#endif // MAINWINDOW3_H

```

mainwindow3.cpp 文件代码

```

#include "mainwindow3.h"
#include "ui_mainwindow3.h"

```

```

MainWindow3::MainWindow3(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow3)
{
    ui->setupUi(this);
    this->ui->centralwidget->setFixedHeight(300);
    this->ui->centralwidget->setFixedWidth(600);
    this->ui->centralwidget->hide();
    initTime();
}

MainWindow3::~MainWindow3()
{
    delete ui;
}

void MainWindow3::initTime(){
    this->timer=new QTimer();
    //    connect(timer,SIGNAL(timeout()),this,SLOT(updateTime()));
    connect(timer, SIGNAL(timeout()), this, SLOT(update()));
    this->timer->start(1000);
}

void MainWindow3::updateTime(){
    //update time
    //    QDateTime time = QDateTime::currentDateTime();
    //    QString str = time.toString("yyyy-MM-dd hh:mm:ss");
    //    ui->radiusBlueLabel->setText(str);
}

void MainWindow3::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event);

    clockTick++;
}

```

```

/* 时分秒所描绘不规则形状和长度 */
// 时针
static const QPoint hourHand[4] =
{
    QPoint(7, 8),    //右
    QPoint(0, 13),   //上
    QPoint(-7, 8),   //左
    QPoint(0, -40)   //下
};
// 分针
static const QPoint minuteHand[4] =
{
    QPoint(7, 8),    //右
    QPoint(0, 13),   //上
    QPoint(-7, 8),   //左
    QPoint(0, -70)   //下
};
// 秒针
static const QPoint secondHand[4] =
{
    QPoint(7, 8),    //右
    QPoint(0, 13),   //上
    QPoint(-7, 8),   //左
    QPoint(0, -80)   //下
};

//-----
/* 时分秒针描绘的颜色 */
// 时针
QColor hourColor(127, 0, 127);
// 分针
//QColor minuteColor(0, 127, 127, 192);      //RGB(三原色)(0, 127,
127);alpha(透明度)(192).
QColor minuteColor(0, 127, 127);
// 秒针

```

```

QColor secondColor(0, 0, 127);

//-----
int side = qMin(this->width()-40, this->height()-40);
QTime time = QTime::currentTime();           //当前系统时间

//-----
QPainter painter(this);
painter.translate(this->width() / 2, this->height() / 2);
                                                    //原点
painter.setRenderHint(QPainter::Antialiasing);
                                                    //反锯齿,好看点
painter.scale(side / 200.0, side / 200.0);    //伸缩

//-----
painter.setPen(Qt::NoPen);                      //设置画笔
painter.setBrush(QColor(0, 0, 0));              //设置画刷(黑色)
painter.drawEllipse(-100, -100, 200, 200);     //画圆(表盘外圆)
painter.setBrush(QColor(255, 255, 255));       //设置画刷(白色)
painter.drawEllipse(-99, -99, 198, 198);       //画圆(表盘内圆)

//-----
// 显示提示信息: "press ESC to exit!"
//-----
QColor escColor(0, 0, 0);                      //黑色
painter.setPen(escColor);                      //设置画笔
painter.rotate(0.0);                          //顺时针旋转角度(0°)
QFont font("宋体", 8, QFont::Normal, false);
//使用字体
painter.setFont(font);

QString escText = "";
// "press ESC to exit!"
if(clockTick % 2)
{

```



```

        escText = tr("press ESC to exit!");
    }

    painter.drawText(
        QRectF(-100, -40, 200, 40),           //画文本的矩形区域
        Qt::AlignHCenter,                       //文本的对齐方式(水平居
中)
        escText                                 //文本的内容
    );

    //-----
    QFont oldfont("宋体", 10, QFont::Normal, false);
    //使用字体
    painter.setFont(oldfont);

    /***** 时钟 *****/
    painter.setPen(Qt::NoPen);                 //设置画笔
    painter.setBrush(hourColor);               //设置画刷
    painter.save();
    painter.rotate((time.hour() * 30.0) + (time.minute() * (30.0 / 60)));
                                                    //顺时针旋转角度(时角
度+分角度)
    painter.drawConvexPolygon(hourHand, sizeof(hourHand) / sizeof(QPoint));
                                                    //描画的形状

    painter.restore();

    //-----
    // 表盘(时)
    //-----
    painter.setPen(hourColor);                 //设置画笔
    for(int i = 0; i < 12; i++)                //描画刻度
    {
        painter.rotate(30.0);                  //顺时针旋转角度
(360 % 12 = 30 °)

```

```

//1,2,4,5,7,8,10,11
if(0 != ((i + 1) % 3))
{
    QFont font("宋体", 10, QFont::Normal, false);
    //使用字体
    painter.setFont(font);

    painter.drawLine(QPointF(80, 0), QPointF(90, 0));    //刻度(时)
    painter.drawText(
        QRectF(-20, -78, 40, 40),                //画文本的矩形区域
        Qt::AlignHCenter | Qt::AlignTop,           //文本的对齐方式
        QString::number(i + 1)                    //文本的内容
    );
}
//3,6,9,12
else
{
    QFont font("宋体", 14, QFont::Bold, false);
    //使用字体
    painter.setFont(font);

    painter.drawLine(QPointF(75, 0), QPointF(90, 0));    //刻度(时)
    painter.drawText(
        QRectF(-20, -73, 40, 40),                //画文本的矩形区域
        Qt::AlignHCenter | Qt::AlignTop,           //文本的对齐方式
        QString::number(i + 1)                    //文本的内容
    );
}
}

//-----
QFont newfont("宋体", 10, QFont::Normal, false);
//使用字体
painter.setFont(newfont);

```

```

/***** 分钟 *****/
painter.setPen(Qt::NoPen);          //设置画笔
painter.setBrush(minuteColor);      //设置画刷
painter.save();
painter.rotate((time.minute() * 6.0) + (time.second() * (6.0 / 60)));
//顺时针旋转角度(分角
度)
painter.drawConvexPolygon(minuteHand, sizeof(minuteHand) / sizeof(QPoint));
//描画的形状

painter.restore();

//-----
// 表盘(分)
//-----
painter.setPen(minuteColor);        //设置画笔
for(int i = 0; i < 60; i++)          //描画刻度
{
    if(0 != (i % 5))                //跳过时刻度
        painter.drawLine(QPointF(82, 0), QPointF(87, 0)); //刻度(分)

    painter.rotate(6.0);              //顺时针旋转角度(360 % 60 = 6 °)
}

/***** 秒钟 *****/
painter.setPen(Qt::NoPen);          //设置画笔
painter.setBrush(secondColor);      //设置画刷
painter.save();
painter.rotate(6.0 * time.second()); //顺时针旋转角度(秒角度)
painter.drawConvexPolygon(secondHand, sizeof(secondHand) / sizeof(QPoint));
//描画的形状

painter.restore();
}

// 键盘事件(由系统自动调用)
void MainWindow3::keyPressEvent(QKeyEvent *event)

```

```
{  
    switch (event->key())  
    {  
        case Qt::Key_Escape:  
            MainWindow3::close();  
            break;  
        default:  
            break;  
    }  
}
```

2 实验二 增加系统调用

2.1 实验目的

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。

2.2 实验内容

内核编译、生成，用新内核启动；

新增系统调用实现：文件拷贝或 P、V 操作。

2.3 实验设计

2.3.1 开发环境

操作系统：Windows10 系统 + VMware Workstation 下的 Ubuntu 虚拟机；
Ubuntu 版本号为 18.10，64 位；系统内核版本为 4.18.0-10-generic。

2.3.2 实验设计

下载官网最新的内核（5.0.0）版本的源码，进行编译安装测试（熟悉内核编译安装的流程），编译成功并且安装新内核成功后，添加文件拷贝的系统调用，再次编译内核，再为新的系统调用写一段测试代码，进行文件拷贝。

2.4 实验调试

2.4.1 实验步骤

下载内核编译依赖软件包：libncurses5-dev、libssl-dev、build-essential 等。

从 <https://www.kernel.org/> 下载 5.0.0 版本的 Linux 内核代码压缩包，并进入解压后的目录，所有的操作都在该目录下进行。

用 gedit 文本编辑器编辑 sys.c 文件,在尾部加入系统调用函数。

```
SYSCALL_DEFINE2(mycopy,const char *,src_file,const char *, dest_file){
    struct kstat k_buf;
    char copy_buf[1024];
    char s_filename[256], t_filename[256];
    int read_fd, write_fd;
    long read_num;

    mm_segment_t fs = get_fs();
    set_fs(get_ds());

    read_num = strncpy_from_user(s_filename, src_file, sizeof(s_filename));
    if (read_num < 0 || read_num == sizeof(s_filename)) {
        set_fs(fs);
        return -EFAULT;
    }
    read_num = strncpy_from_user(t_filename, dest_file, sizeof(t_filename));
    if (read_num < 0 || read_num == sizeof(t_filename)) {
        set_fs(fs);
        return -EFAULT;
    }

    if (vfs_stat(s_filename, &k_buf) != 0) {
        set_fs(fs);
        return -EFAULT;
    }

    if ((read_fd = ksys_open(s_filename, O_RDONLY, S_IRUSR)) == -1) {
        set_fs(fs);
        return -EFAULT;
    }
    if ((write_fd = ksys_open(t_filename, O_WRONLY | O_CREAT | O_TRUNC, k_buf.mode)) == -1) {
        set_fs(fs);
        return -EFAULT;
    }

    while(1) {
        read_num = ksys_read(read_fd, copy_buf, sizeof(copy_buf));
        if (read_num < 0) {
            set_fs(fs);
            return -EFAULT;
        } else if (read_num == 0)
            break;
        ksys_write(write_fd, copy_buf, read_num);
    }

    ksys_close(read_fd);
    ksys_close(write_fd);

    set_fs(fs);

    return 0;
}
```

Show Applications

图 2-1 sys.c 文件添加功能

用 gedit 文本编辑器 syscalls.h 进行编辑，加入系统调用函数的声明。

```
asmlinkage long sys_mycopy(const char* src_file,const char * dest_file);
```

用 gedit 编辑 syscalls/syscall_64.tbl，寻找第一个未被使用的系统调用号，如图 2-1，在第 337 号处加入系统调用。

```
337      64      mycopy      __x64_sys_mycopy|
#
```

图 2-2 分配系统调用号

用 `sudo` 获取 `root` 权限，执行 `make mrproper`、`make clean`、`make menuconfig` 指令清楚上一次编译产生的文件。

执行 `make -j8` 指令（开启 8 个线程），开始编译内核，等待直到完成编译。

编译内核完成后，先执行 `make install` 指令，再执行 `sudo make modules_install`，安装模块和内核。

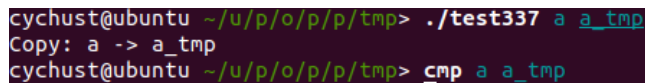
重启虚拟机，在重启时长按 `shift` 键，可以在启动之前选择内核版本，选中 5.0.0 版本。

编写测试代码，查看运行结果。

若结果符合预期，则回到第 4 步，添加文件拷贝的系统调用函数，再次执行一次内核编译并测试。

2.4.2 实验调试及心得

添加文件拷贝系统调用，再次编译内核，由于第一次编译较慢，此次使用 `-j8` 参数加快编译速度，编译完成后重启系统，用测试程序拷贝文件。如图 2-3，新文件与原文件内容完全相同，系统调用添加成功。



```
cychust@ubuntu ~/u/p/o/p/p/tmp> ./test337 a a_tmp
Copy: a -> a_tmp
cychust@ubuntu ~/u/p/o/p/p/tmp> cmp a a_tmp
```

图 2-3 文件拷贝系统调用结果

实验心得：

编译内核是用 `make` 程序进行编译，时间一般耗时很长，运用 `make -j $kernel_num` 在 `make` 执行过程中可以同时执行的指令数目，相当于并行，在有大量独立模块且依赖较上的时候，能够大大加快底层的编译速度。第一次编译内核方式实现系统的调用，收益颇丰，但耗时太长，常常编译内核需要重新编译，耗时几个小时，需要有耐心进行实验。可以考虑增量编译的方式进行节约时间。

附录 实验代码

sys.c 文件增加的代码

```
SYSCALL_DEFINE2(mycopy,const char *,src_file,const char *,dest_file){
    struct kstat k_buf;
    char copy_buf[1024];
    char s_filename[256],t_filename[256];
    int read_fd,write_fd;
    long read_num;
```

```

/* Set memory access range */
mm_segment_t fs = get_fs();
set_fs(get_ds());

/* Get source and target file name */
read_num = strncpy_from_user(s_filename, src_file, sizeof(s_filename));
if (read_num < 0 || read_num == sizeof(s_filename)) {
    set_fs(fs);
    return -EFAULT;
}
read_num = strncpy_from_user(t_filename, dest_file, sizeof(t_filename));
if (read_num < 0 || read_num == sizeof(t_filename)) {
    set_fs(fs);
    return -EFAULT;
}

/* Get source file mode */
if (vfs_stat(s_filename, &k_buf) != 0) {
    set_fs(fs);
    return -EFAULT;
}

/* Open files */
if ((read_fd = ksys_open(s_filename, O_RDONLY, S_IRUSR)) == -1) {
    set_fs(fs);
    return -EFAULT;
}
printk("After open source file");
if ((write_fd = ksys_open(t_filename, O_WRONLY | O_CREAT | O_TRUNC,
k_buf.mode)) == -1) {
    set_fs(fs);
    return -EFAULT;
}

/* Do copy */

```



```

for (;;) {
    read_num = ksys_read(read_fd, copy_buf, sizeof(copy_buf));
    if (read_num < 0) {
        set_fs(fs);
        return -EFAULT;
    } else if (read_num == 0)
        break;
    ksys_write(write_fd, copy_buf, read_num);
}

ksys_close(read_fd);
ksys_close(write_fd);

/* Restore previous memory access */
set_fs(fs);

return 0;
}

```

syscall_64.tbl 增加代码:

```

337      64      mycopy      __x64_sys_mycopy

```

syscalls.h 增加代码为

```

/*
 *mycall
 */
asmlinkage long sys_mycall(pid_t pid,int flag ,int nicevalue,void __user*prio,void
__user*nice);

```

3 实验三 增加字符设备驱动

3.1 实验目的

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。

3.3 实验设计

3.3.1 开发环境

Ubuntu 实体机，Ubuntu 版本号为 18.10，64 位；系统内核版本为 4.18.0-10-generic。

3.3.2 实验设计

新的字符设备驱动程序需要有注册、注销、打开、读和写这些基本操作，分别用 4 个功能函数实现。

注册设备用 `register_chrdev` 函数，需要指定设备名，设备号自动分配；注销设备用 `unregister_chrdev` 函数。

读设备，是从内核态读到用户态，由于内核态和用户态的内存空间不一致，所以需要用 `copy_to_user` 函数将内核空间的字符转到用户空间；同理，写设备时需要使用 `copy_from_user` 函数从用户空间转到内核空间。

3.4 实验调试

3.4.1 实验步骤

编写设备功能函数；

编写 makefile 文件，执行 make 命令，makefile 文件编译设备功能函数并生成驱动程序。

用 insmod 命令将生成的模块加载到内核。

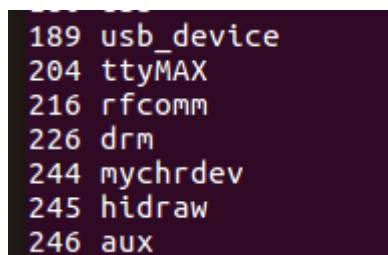
Cat /proc/devices 命令查看所有设备，可以看到新设备的设备号。

用 mknod 命令将新设备加载到内核。

编写测试程序，测试设备的读写字符能力。

3.4.2 实验调试及心得

用 make 编译生成驱动后，将驱动加载到内核，用 cat 命令查看设备，如图 3-1，244 号设备 mychrdev 是新增的设备。



```
189 usb_device
204 ttyMAX
216 rfcomm
226 drm
244 mychrdev
245 hidraw
246 aux
```

图 3-1 查看设备

用 mknod 命令将设备加载到内核，用 ls /dev 查看设备，也能看到新加入的设备。

编译运行测试程序，如图 3-2，输入任意字符串，将该字符串写入设备，然后将其从设备读出，再打印出来，结果与写入的一致，证明字符设备功能达到实验要求。

实验心得：

第一次进行增加系统设备驱动，在调试过程中，发现存在无法卸载驱动的情况，最后通过查阅资料解决问题，增加设备驱动的时间比上次实验少很多，学到了新的技能，收益匪浅。

附录 实验代码

chrdev.c 代码如下:

```
#include "linux/kernel.h"
#include "linux/module.h"
#include "linux/fs.h"
#include "linux/init.h" // dd init and exit
#include "linux/types.h"
#include "linux/errno.h"
#include "linux/uaccess.h"
#include <linux/kdev_t.h>
#include <linux/types.h>
const char * devName="cycDevDriver";
int devNum;
char buf[1024] = "---cyc---";
int myOpen(struct inode *inode, struct file* file);
int myRelease(struct inode*inode, struct file* file);
ssize_t myRead(struct file *file, char __user* user,size_t t, loff_t*f);
ssize_t myWrite(struct file *file, const char __user* user,size_t t, loff_t*f);

struct file_operations structfileOperations={
    owner:THIS_MODULE,
    open: myOpen,
    release:myRelease,
    read: myRead,
    write:myWrite,
};

int init_module(){
    int ret=register_chrdev(0,devName,&structfileOperations);
    if(ret<0){
        printk("regist fail");
        return -1;
    }
    printk("myDevDrive has been registered!\n");
}
```

```

        devNum = ret;
        // debug information
        printk("myDevDrive's id: %d\n",ret);
        printk("usage: mknod /dev/myDevDrive c %d 0\n",devNum);
        printk("delete device\n\t usage: rm /dev/%s ",devName);
        printk("delete module\n\t usage: rmmod device ");
        return 0;
    }

void unregister_module(void)
{
    unregister_chrdev(devNum,devName);
    printk("unregister success !\n");
}

int myOpen(struct inode * inode,struct file * file){
    printk("open myDrive OK ! \n");
    try_module_get(THIS_MODULE);
    return 0;
}

int myRelease(struct inode*inode,struct file*file){
    printk("Device  release !\n");
    module_put(THIS_MODULE);
    return 0;
}

ssize_t myRead(struct file *file, char __user *user, size_t t, loff_t *f){
    if (copy_to_user(user ,buf,sizeof(buf)))
    {
        return -2;
    }
    return sizeof(buf);
}

```

```

ssize_t myWrite(struct file *file, const char __user *user, size_t t, loff_t *f)
{
    if(copy_from_user(buf,user,sizeof(buf)))
    {
        return -3;
    }
    return sizeof(buf);
}

```

test.c 文件代码如下:

```

#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#define MAX_SIZE 1024
int main()
{
    int fd;
    char buf[MAX_SIZE];
    char get[MAX_SIZE]; // to be written
    char devName[20];
    char dir[50] = "/dev/cycDevDriver";
    //system("ls /dev/");
    //printf("Please input the device's name: ");
    //gets(devName);
    //strcat(dir, devName);
    fd = open("/dev/cycDevDriver", O_RDWR | O_NONBLOCK,
S_IRUSR|S_IWUSR);
    if( fd>0 )
    {
        // get str from buf
        read( fd , buf , sizeof(buf) );

```

```

printf( "%s\n" , buf);
    // read
printf( "Please input a string : ");
gets(get);
write( fd , get , sizeof(get) );

// read back
read(fd, buf, sizeof(buf));
printf("device Message : %s\n", buf);

close(fd);
return 0;
}
else
{
    printf("Device open failed !\n");
    return -1;
}
}

```

Makefile 文件代码如下：

```

ifeq ($(KERNELRELEASE),)
KERNELDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
modules:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
modules_install:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules_install
clean:
    rm -rf *.o .depend *.cmd *.ko *.mod.c .tmp_versions modules.*
else
obj-m := chrdev.o
endif

```

4 实验四 系统监视器(QT)

4.1 实验目的

使用 GTK/QT 实现系统监控器。

了解/proc 文件的特点和使用方法；

监控系统中进程运行情况；

用图形界面实现系统资源的监控。

4.2 实验内容

- (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

4.3 实验设计

4.3.1 开发环境

操作系统： Ubuntu 18.04 系统，内核版本 4.18.0-10-generic。

编程环境：使用 Qt Creator 编程并编译运行。

4.3.2 实验设计

当前时间调用 Qt 的函数 `QTimer` 可以获得。

关机功能用 `system` 函数，用 `shutdown` 命令实现关机。

Cpu 等系统信息通过从 `/proc` 文件系统获取数据：

- 1) 系统迄今为止持续运行的时间从 `/proc/uptime` 文件中可以获取
- 2) 系统启动时间需要用当前时间减去运行时间；
- 3) 系统版本号从 `/proc/sys/kernel/ostype` 文件中获取；
- 4) cpu 型号在 `/proc/cpuinfo` 文件中获取；
- 5) 进程信息在 `/proc/pid/stat` 文件中获取；
- 6) 内存占用信息在 `/proc/meminfo` 中获取；
- 7) cpu 占用信息在 `/proc/stat` 文件中获取。

用调用 `qt` 的函数打开各个文件，从中读取字符数据，再根据需要，从字符串中获取有用的数据。

主机名、系统启动时间、系统版本号、CPU 型号和主频等信息对于一次启动计算机来说是固定的，所以只需要获取一次，永久显示在图形界面上；系统当前时间、系统持续运行时间 1 秒刷新一次；cpu 使用率图像可以设计 100 个点，2 秒刷新一次。所有的刷新都使用 `Qtimer` 计时器和信号槽实现。

启动新进程需要用到 `system` 函数，也就是相当于系统终端下的命令行，将进程名作为参数调用 `system` 函数，即可开启进程。

4.4 实验调试

4.4.1 实验步骤

创建 Qt Widget 工程，`TabWidget` 的四个页面分别显示系统参数、进程管理和 CPU 使用率信息、Swap 使用率和 temp 使用率；

系统参数一页，静态显示主机名、系统版本号、启动时间、cpu 型号、cpu 主频，动态显示系统持续运行时间，信息的显示用 `textedit` 和 `linedit` 控件实现。

进程管理一页，用 `Qtable Widget` 类绘制表格，表格中显示所有进程的进程号、进程名、进程状态、父进程号、动态优先级和内存占用情况。设置单机选中

进程，按确认键可结束该进程。

使用率的三个组件使用 QCharts 组件进行绘画。

状态栏显示当前时间，当前 cpu 使用率和当前内存使用率。

4.4.2 实验调试及心得

完成后的系统功能演示如下：

1) 显示系统参数，其中除了系统运行时间可变，其他的都是固定信息，如图 4-1。

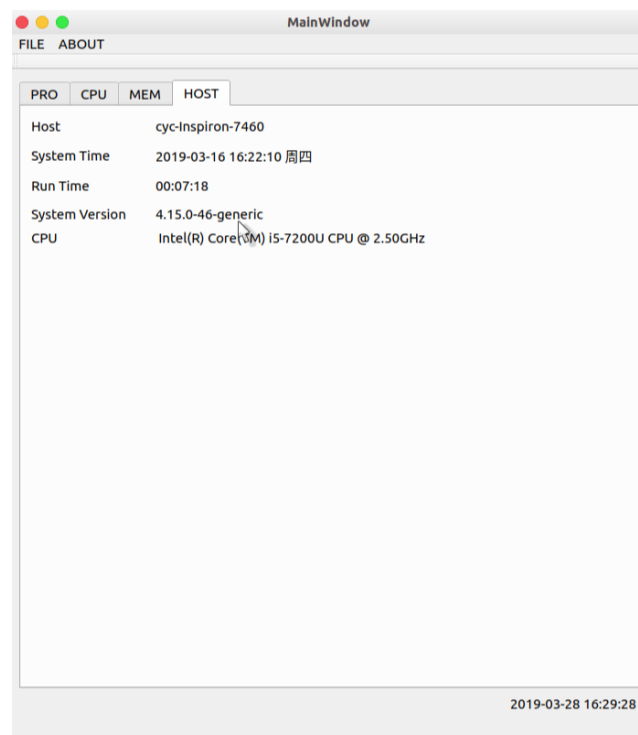


图 4-1 系统参数信息

2) 显示进程信息，如图 4-2。

1	2	3	4	5	6
251 888	wpa_suppli...	S	1	20	1853
252 890	NetworkM...	S	1	20	4648
253 892	thermald	S	1	20	2378
254 893	ModemMa...	S	1	20	2292
255 894	avahi-daemon	S	1	20	832
256 895	accounts-daemon	S	1	20	1809
257 898	acpid	S	1	20	197
258 9	rcu_bh	I	2	20	0
259 939	polkitd	S	1	20	2478
260 940	polipo	S	1	20	35
261 943	avahi-daemon	S	894	20	84
262 944	cups-browsed	S	1	20	2696
263 956	courierlogger	S	1	20	19
264 957	courierlogger	S	1	20	19
265 958	couriertcpd	S	956	20	453
266 959	couriertcpd	S	957	20	473
267 99	kthrotld	I	2	0	0
268 995	unattended-	S	1	20	4045

图 4-2 进程信息列表

- 1) 查询进程，可以用进程号进行精确查找，也可以用进程名进行模糊查询，如图 4-3。

1	2	3	4	5	6
1 4202	chrome	S	1	20	51603
2 4222	chrome	S	4202	20	13979
3 4235	chrome	S	4222	20	3632
4 4274	chrome	S	4202	20	19122
5 4329	chrome	S	4202	20	21700
6 4402	chrome	S	4235	20	23121
7 4523	chrome	S	4235	20	12208
8 4525	chrome	S	4235	20	17953
9 4544	chrome	S	4235	20	20613
10 4550	chrome	S	4235	20	18270
11 4551	chrome	S	4235	20	20281
12 4562	chrome	S	4235	20	18888
13 4568	chrome	S	4235	20	18199
14 4569	chrome	S	4235	20	20658
15 4570	chrome	S	4235	20	17833
16 4581	chrome	S	4235	20	17791
17 4592	chrome	S	4235	20	19670
18 4599	chrome	S	4235	20	18593

图 4-3 查找 chrome 进程列表显示结果

- 2) Cpu 使用率图像如图 4-8 所示，多数情况下使用率都很低，在启动某个

新进程时，使用率会陡增，不过很快就会恢复。

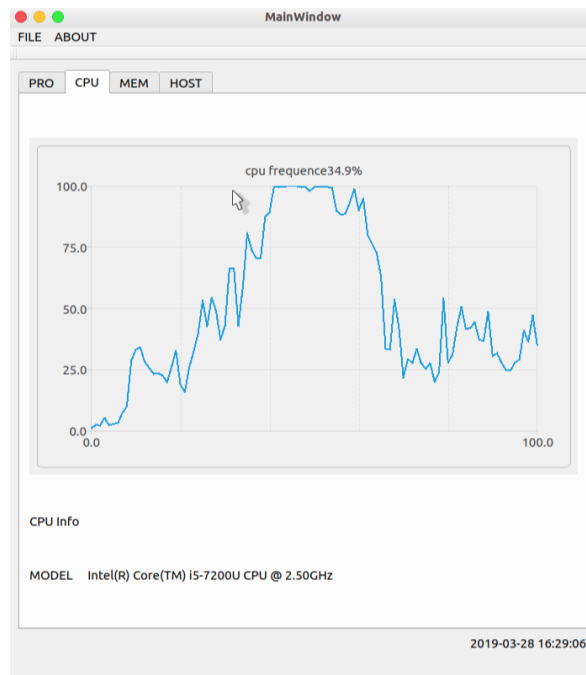


图 4-4 cpu 使用率图像

- 3) 内存使用率图像如图 4-9，内存的占用一直都很高，而且维持稳定。Swap 交换率一直显示 0%

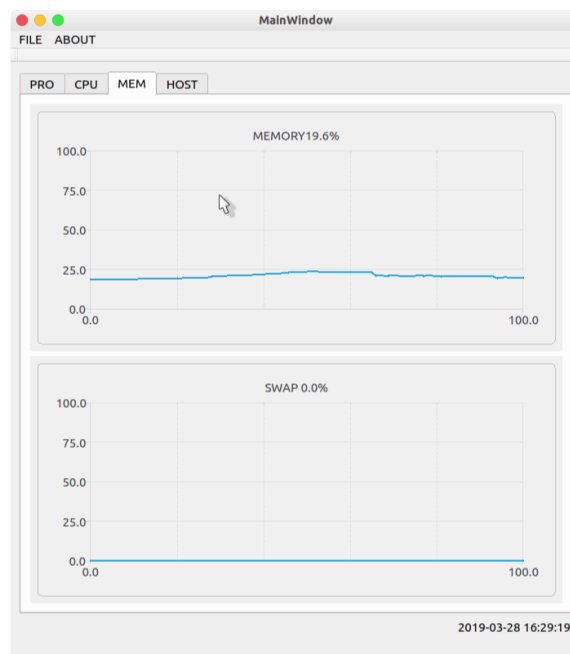


图 4-9 内存使用率和 swap 交换率图像

实验心得：

实验使用 QT 进行，总体只有一个 QWidget，省去了页面跳转的逻辑，更新机制上，获取数据部分和绘图部分分离，可以采用不同的定时器时长，在绘图的

间隙中间进行数据的获取，减少卡顿。文件读取进行字符串比较，截取需要的结果，实验尽量进行代码复用，实验的难度较小，但项目量比较大，也有收获。

附录 实验代码

Part.pro 文件代码:

```
#-----  
#  
# Project created by QtCreator 2019-03-12T10:13:47  
#  
#-----  
  
QT      += core gui  
  
QT += charts  
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets  
  
TARGET = part3  
TEMPLATE = app  
  
# The following define makes your compiler emit warnings if you use  
# any feature of Qt which has been marked as deprecated (the exact warnings  
# depend on your compiler). Please consult the documentation of the  
# deprecated API in order to know how to port your code away from it.  
DEFINES += QT_DEPRECATED_WARNINGS  
  
# You can also make your code fail to compile if you use deprecated APIs.  
# In order to do so, uncomment the following line.  
# You can also select to disable deprecated APIs only up to a certain version of Qt.  
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables  
all the APIs deprecated before Qt 6.0.0
```

```
CONFIG += c++11
```

```
SOURCES += \  
    main.cpp \  
    mainwindow.cpp \  
    util.cpp \  
    mythread.cpp \  
    global.cpp
```

```
HEADERS += \  
    mainwindow.h \  
    util.h \  
    mythread.h \  
    global.h
```

```
FORMS += \  
    mainwindow.ui
```

```
## Default rules for deployment.
```

```
#qnx: target.path = /tmp/${TARGET}/bin
```

```
#else: unix:!android: target.path = /opt/${TARGET}/bin
```

```
#!isEmpty(target.path): INSTALLS += target
```

```
#CONFIG += console
```

```
main.cpp 文件代码:
```

```
#include "mainwindow.h"
```

```
#include <QApplication>
```

```
#include "QtDebug"
```

```
#include <global.h>
```

```
// #include "QChartView"
```

```
// using namespace QtCharts;
```

```
QStringList Global::cpuRates;
```

```

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

    return a.exec();
}

```

mainwindow.h 文件代码

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include "qtimer.h"
#include <QMainWindow>
#include "util.h"
#include "mythread.h"
#include <QtCharts>
#include <qlist.h>
QT_CHARTS_USE_NAMESPACE
namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;

```

```

Util util;
QTimer *timer1;
QTimer *timer3;
MyThread* myThread;

double total=0.0;
double idle=0.0;
double rate=0.0;

bool searchMode=false;

QValueAxis *axisYProcess;
QValueAxis *axisXProcess;

QValueAxis *axisYMem;
QValueAxis *axisXMem;
QValueAxis *axisYSwap;
QValueAxis *axisXSwap;

QLineSeries * linesProcess;
QLineSeries * linesMem;
QLineSeries * linesSwap;

QList<QMap<QString,QString>>processMapList;
QList<double> cpuStates;

QList<double> memRates;

int rowSelect=-1;

int i=0;

QChart*processLineChart;
QChart*memLineChart;
QChart*swapLineChart;

```



```

        QStringList header;
        void initWidgets();
        void initVariable();
        void initTableWidget();
        void          chartViewPainted(QList<double>          pointList,QLineSeries*
lines,QChartView*    widget,
                                QChart * lineChart,QValueAxis* axisY,QValueAxis*
axisX,QString title);

public slots:
    void initDynamicWidgets();
    void dynamicProcessRate();
    void dynamicMemRate();
    void dynamicSwapRate();
    void tableClicked(int row,int column);
    void killBtnClicked();
    void searchBtnClicked();
    void shutdown();
};

```

```

#endif // MAINWINDOW_H

```

Mainwindows.cpp 文件代码

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "util.h"
#include "qabstractitemview.h"
#include "qstringlist.h"
#include "QChartView"
#include "global.h"
MainWindow::MainWindow(QWidget *parent) :

```

```

MainWindow(parent),
ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    timer1=new QTimer();
    timer3=new QTimer();
    Util util;
    qWarning() << "**** " <<util.host_info ();
    qWarning() << "**** " <<util.sys_version ();
    qWarning() << "**** " <<util.CPU_model();
    qWarning() << "**** " <<util.CPU_frequence();
    util.process_stat ();
    initWidgets ();
    initDynamicWidgets ();
    initVariable ();
//    iniDynamicVariable ();
    dynamicMemRate ();
    dynamicProcessRate ();
    dynamicSwapRate ();
}

```

```

MainWindow::~MainWindow()
{
    delete timer1;
    delete timer3;
    delete axisXProcess;
    delete axisYProcess;
    delete axisXMem;
    delete axisYMem;
    delete axisXSwap;
    delete axisYSwap;
    delete myThread;
    delete ui;
}

```

```

void MainWindow::initVariable(){
    this->timer1=new QTimer();
    this->timer3=new QTimer();
    connect(timer1,SIGNAL (timeout()),this,SLOT(initDynamicWidgets ());
//    connect(timer3,SIGNAL (timeout()),this,SLOT(iniDynamicVariable()));

    connect(timer3,SIGNAL (timeout()),this,SLOT(dynamicProcessRate()));
    connect(timer3,SIGNAL (timeout()),this,SLOT(dynamicMemRate()));
    connect(timer3,SIGNAL (timeout()),this,SLOT(dynamicSwapRate()));

    connect          (ui->tableWidget,SIGNAL(cellClicked(int,int)),this,SLOT
(tableClicked(int,int)));
    connect (ui->killer_btn,SIGNAL (clicked()),this,SLOT (killBtnClicked()));
    connect (ui->search_btn,SIGNAL (clicked()),this,SLOT (searchBtnClicked()));
    connect (ui->actionsshutdown,SIGNAL (clicked()),this,SLOT(shutdown()));
    timer1->start (1000);
    timer3->start (1000);
//    connect (ui->tabWidget,SIGNAL(current))

//    myThread=new MyThread(this);
//    myThread->start ();

//    lineChart=new QChart;
//    memLineChart=new QChart;
//    swapLineChart=new QChart;

    axisYProcess=new QValueAxis();
    axisXProcess=new QValueAxis();
    axisYProcess->setRange (0,100);
    axisXProcess->setRange (0,100);
    axisXProcess->setTickCount(2); //主分隔个数: 0 到 10 分成 20 个单位
    axisXProcess->setMinorTickCount(4); //每个单位之间绘制了多少虚网线

    axisYMem=new QValueAxis();

```

```

axisXMem=new QValueAxis();
axisYMem->setRange (0,100);
axisXMem->setRange (0,100);
axisXMem->setTickCount(2); //主分隔个数： 0 到 10 分成 20 个单位
axisXMem->setMinorTickCount(4); //每个单位之间绘制了多少虚网线

axisYSwap=new QValueAxis();
axisXSwap=new QValueAxis();
axisYSwap->setRange (0,100);
axisXSwap->setRange (0,100);
axisXSwap->setTickCount(2); //主分隔个数： 0 到 10 分成 20 个单位
axisXSwap->setMinorTickCount(4); //每个单位之间绘制了多少虚网线

for(int i=0;i<=100;i++){
    cpuStates.push_back(-1.0);
    memRates.push_back (-1.0);
}

}

void MainWindow::shutdown (){
    system ("halt");
}

void MainWindow::tableClicked(int row,int column){
    rowSelect=row;
    qDebug()<<rowSelect;
}

void MainWindow::killBtnClicked (){
    if(rowSelect==-1){
        return;
    }
    if(rowSelect>=processMapList.size ()){

```

```

        return;
    }
    QMap<QString,QString> process=processMapList.at(rowSelect);
    long pid=process["pid"].toLong ();
    if(pid>0){
        if(!kill(pid,SIGKILL))qDebug()<<"kill success";
    }
}

void MainWindow::dynamicProcessRate(){
    QStringList tmp=util.cpu_rates ();
    double total2=tmp[0].toDouble ();
    double idle2=tmp[1].toDouble();
    double rate2;
    rate2=(1-(idle2-idle)/(total2-total))*100.0;
    if(rate2<0)rate2-=rate2;
    if(rate2!=0&&rate2!=100){
        rate=rate2;
        total=total2;
        idle=idle2;
    }
    cpuStates.pop_front();
    cpuStates.push_back(rate);
    processLineChart=new QChart();
    chartViewPainted (cpuStates,linesProcess,
ui->widget,processLineChart,axisYProcess,axisXProcess,"cpu frequence");
}

void MainWindow::dynamicMemRate(){
    memRates.pop_front();
    double memRate=util.mem_rate ();
    memRates.push_back (memRate);
    memLineChart=new QChart();
    chartViewPainted (memRates,linesMem,

```

```

ui->mem_widget,memLineChart,axisYMem,axisXMem,"MEMORY");

}

void MainWindow::dynamicSwapRate(){
    memRates.pop_front();
    double memRate=util.mem_rate ();
    memRates.push_back (memRate);
//    auto swapLineChart=new QChart();
    swapLineChart=new QChart();
    chartViewPainted (memRates,linesSwap,
ui->swap_widget,swapLineChart,axisYSwap,axisXSwap,"SWAP ");
}

void MainWindow::chartViewPainted(QList<double> pointList,QLineSeries* lines,
QChartView* widget,QChart * lineChart,
                                QValueAxis* axisY,QValueAxis*
axisX,QString title){
    lines = new QLineSeries();
    for(int i=0;i<cpuStates.size ();i++){
        lines->append(i,cpuStates[i]);
    }
    lineChart->setTitle(title+QString::number(cpuStates[cpuStates.size
()-1], 'f', 1)+"%");
    lineChart->legend()->setVisible(false);
    lineChart->addSeries(lines);
    lineChart->setAxisY(axisY,lines);
    lineChart->setAxisX(axisX,lines);
//    lineChart->setGeometry(10, 4, 300, 260); // the method of QGraphicsWidget,
move && resize
    lineChart->setBackgroundVisible(true);
    lineChart->setBackgroundPen(QPen(Qt::lightGray)); // the frame
    lineChart->setBackgroundBrush(QBrush(QColor(240, 240, 240)));
    widget->setChart (lineChart);
}

void MainWindow::initWidgets(){
    util.setText (ui->cpu_content_label,util.CPU_model ());

```

```

    util.setText (ui->sys_version_content_label,util.sys_version ());
    util.setText (ui->host_content_label,util.host_info ());
    util.setText (ui->systime_content_label,util.sys_time ());

    //tableWidget

ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch
);

    header<<"PID"<<"COMM"<<"STATE"<<"PPID"<<"PRIORITY"<<"RSS";
    ui->tableWidget->setColumnCount (6);
        ui->tableWidget->setHorizontalHeaderLabels(header);
    //

}

void MainWindow::initDynamicWidgets(){
    util.setText (ui->runtime_content_label,util.run_time ());
    //        ui->statusBar->showMessage(QDateTime::currentDateTime    ().toString
("yyyy-MM-dd"));
        ui->status_label->setText        (QDateTime::currentDateTime        ().toString
("yyyy-MM-dd hh:mm:ss"));

    processMapList=util.process_stat ();
    initTableWidget ();
}

void MainWindow::initTableWidget(){
    if(processMapList.isEmpty ())return;
    ui->tableWidget->setRowCount (processMapList.size());
    ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);
    int j=0;
    ui->tableWidget->clear ();
    for(int i=0;i<processMapList.size();i++){
        QMap<QString,QString> map=processMapList.at(i);

```

```

        if(searchMode){
            QString lineEdit=ui->lineEdit->text ();
            if(util.isNumberByString (lineEdit)){
                if(map["pid"]!=lineEdit)
//                qDebug()<<"is numer"<<map["pid"];
                continue;
            }else{
                if(!map["comm"].contains(lineEdit))
                    continue;
            }
        }
        ui->tableWidget->setItem (j,0,new QTableWidgetItem (map["pid"]));
        ui->tableWidget->setItem (j,1,new QTableWidgetItem (map["comm"]));
        ui->tableWidget->setItem (j,2,new QTableWidgetItem (map["task_state"]));
        ui->tableWidget->setItem (j,3,new QTableWidgetItem (map["ppid"]));
        ui->tableWidget->setItem (j,4,new QTableWidgetItem (map["priority"]));
        ui->tableWidget->setItem (j,5,new QTableWidgetItem (map["vsize"]));
        j++;
    }
}

```

```

void MainWindow::searchBtnClicked(){
    if(searchMode){
        searchMode=false;
        ui->search_btn->setText ("SEARCH");
    }else {
        searchMode=true;
        ui->search_btn->setText ("ALL");
    }
    initTableWidget();
}

```

util.h 文件代码:

```

#ifndef UTIL_H

```



```

#define UTIL_H

#include "QString"
#include "qfile.h"

#include "qlabel.h"
#include "qdir.h"
#include "qmap.h"
#include "qthread.h"

#include "QtDebug"
#include "qtextstream.h"
#include "istream"
#include "QDateTime"
#include "QTime"
#include "qabstractitemview.h"
#include <sys/types.h>
#include <signal.h>

class Util
{
public:
    Util();

    static const QString HOST_NAME_PATH;
    static const QString CPU_INFO_PATH;
    static const QString SYSTEM_VERSION_PATH;
    static const QString SYSTEM_TIME_PATH;
    static const QString RUNTIME_PATH;
    static const QString CPU_FREQUENCY_PATH;

    QString sys_time();
    QString run_time();
    QString host_info();
    QString sys_version();

```

```

    QString CPU_model();
    QString CPU_frequence();
    void setText(QLabel* o,QString text);
    QList<QMap<QString,QString>> process_stat();
    QString CPU_Utilization_Rate();
    QStringList cpu_rates();
    bool isNumberByString(QString name);
    double mem_rate();
private:

    QString file_content(QString filePath);
    QString file_content(QString filePath,QString lineContained);
    QString file_content(QString filePath,QStringList argv);
    bool IsExistByPid(QString pid);
};

#endif // UTIL_H

```

util.cpp

```
#include "util.h"
```

```

Util::Util()
{

}

const QString Util::HOST_NAME_PATH="/proc/sys/kernel/hostname";
const QString Util::CPU_INFO_PATH="/proc/cpuinfo";
const QString Util::SYSTEM_VERSION_PATH="/proc/version";
const QString Util::SYSTEM_TIME_PATH="/proc/uptime";
const QString Util::RUNTIME_PATH="";
const QString Util::CPU_FREQUENCY_PATH="";

```

```

QString secondToDay(long seconds){
    long day,hour,minute,second;
    long tmp=seconds;
    day=tmp/(60*60*24);
    tmp%=(60*60*24);
    hour=tmp/(60*60);
    tmp%=tmp%(60*60);
    minute=tmp/60;
    second=tmp%60;
    return  QString("").sprintf("%ld  days  %ld  hours  %ld  minutes  %ld
seconds",day,hour,minute,second);
}

```

```

QString NoNULL_content(QString content){
    if(content==nullptr) return "";
    return content;
}

```

```

QString Util::sys_time(){
    QString content=file_content (SYSTEM_TIME_PATH);
    if(content==nullptr) return nullptr;
    QStringList times = content.split (" ");
    QDateTime currentTime=QDateTime::currentDateTime();
    qint64 runtime=times[0].toDouble();
    currentTime = currentTime.addSecs(-runtime);
    QString result= currentTime.toString ("yyyy-MM-hh hh:mm:ss ddd");
    return result;
}

```

/**

* (3) run time /proc/uptime

* @brief Util::run_time

* @return

*/

```

QString Util::run_time(){
    QString content = file_content(SYSTEM_TIME_PATH);

```

```

        if(content==nullptr) return nullptr;

        QStringList times = content.split (" ");
        if(times.length ()>0){
            QTime runtime(0,0);
            runtime=runtime.addSecs((long long)times[0].toDouble ());
            QString result=runtime.toString ("HH:mm:ss");
            return result;
        }
        return nullptr;
    }
}

/**
 * (1)host info   /proc/host
 * @brief Util::host_info
 * @return
 */
QString Util::host_info(){    //(1)host info
    QString hostName=file_content(HOST_NAME_PATH);
    return  hostName;
}

/**
 * (4) system version /proc/version
 * @brief Util::sys_version
 * @return
 */
QString Util::sys_version(){
    QString fileContent=file_content (SYSTEM_VERSION_PATH);
    if(fileContent==nullptr) return nullptr;
    QStringList elems=fileContent.split (" ");
    if(elems.length ()<3)return nullptr;
    return elems[2];
}

/**
 * (5) cpu model

```

```

* @brief Util::CPU
* @return
*/
QString Util::CPU_model(){
    QString fileContent=file_content (CPU_INFO_PATH,"model name");
    QStringList cpus=fileContent.split ("\n");
    QString result;
    result.append (cpus[0].mid (cpus[0].lastIndexOf (":")+1,cpus[0].length ());
    return result;
}
/**
* cpu frequency
* @brief Util::CPU_frequency
* @return
*/
QString Util::CPU_frequency(){
    QString fileContent=file_content (CPU_INFO_PATH,"cpu MHz");
    QStringList cpus=fileContent.split ("\n");
    QString result;
    for (int i=0;i<cpus.size ();i++) {
        result.append (cpus[i].mid (cpus[i].lastIndexOf (":")+1,cpus[i].length
    ())).append ("\n");
    }
    return result;
}

/**
* get all content of file
* @brief Util::file_content
* @param filePath
* @return
*/
QString Util::file_content(QString filePath){
    QFile file(filePath);
    QString result;

```

```

        if(!file.open(QIODevice::ReadOnly)){
            qDebug()<<"Can't open the file"<<endl;
            return nullptr;
        }
        QTextStream in(&file);
        QString line;
        do{
            line=in.readLine ();
            if(!line.isNull ())
                result.append (line);
        }while(!line.isNull ());

        if(file.isOpen ())
            file.close ();
        return result;
    }
}

/**
 * get the lines of file which contain the specific content
 * @brief Util::file_content
 * @param filePath
 * @param lineContained
 * @return
 */
QString Util::file_content(QString filePath,QString lineContained){
    QFile file(filePath);
    QString result="";
    if(!file.open(QIODevice::ReadOnly)){
        qDebug()<<"Can't open the file"<<endl;
        return nullptr;
    }
    QTextStream in(&file);
    QString line;
    do{
        line=in.readLine();
        if(!line.isNull ()){

```

```

        if(line.contains(lineContained)){
            result.append(line).append("\n");
        }
    }
}while(!line.isNull ());
//    qDebug()<<"*****";
if(file.isOpen ())
    file.close ();
return result;
}

QString Util::file_content(QString filePath,QStringList argv){
    QFile file(filePath);
    QString result;
    if(!file.open(QIODevice::ReadOnly)){
        qDebug()<<"Can't open the file"<<endl;
        return nullptr;
    }
    QTextStream in(&file);
    QString line;
    do{
        line=in.readLine();
        if(!line.isNull ()){
            for (int i=0;i<argv.size();i++) {
                if(line.contains(argv[i])){
                    result.append(line).append("\n");
                    break;
                }
            }
        }
    }while(!line.isNull ());
//    qDebug()<<"*****";
if(file.isOpen ())
    file.close ();
return result;
}

```

```

}
/**
 * file is exist or not
 * @brief Util::IsExist
 * @param pid
 * @return
 */
bool Util::IsExistByPid(QString pid){
    QFile file("/proc/"+pid);
    if(!file.exists ()){
        return false;
    }
    return true;
}

/**
 * Judge a file Name is number or not
 * @brief Util::IsNumberByString
 * @param name
 * @return
 */
bool Util::isNumberByString(QString name){
    QString pattern("[0-9]+");
    QRegExp rx(pattern);
    return rx.exactMatch (name);
}

/**
 * (7)get the list of processes
 * @brief Util::process_stat
 * @return
 */
QList<QMap<QString,QString>> Util::process_stat(){
    QDir dir("/proc");
    dir.setFilter (QDir::Dirs);
    QStringList list = dir.entryList();

```



```

QList<QMap<QString,QString>> mapList;
for (int var = 0; var < list.size (); ++var) {
    if(isNumberByString (list.at(var))) { //is number
        QMap<QString,QString> map;
//        QString statusFilePath="/proc/"+list.at (var)+"/status";
        QString statFilePath="/proc/"+list.at (var)+"/stat";
        QString content=file_content (statFilePath);
        if(content==nullptr) continue;
        QStringList elems=content.split (" ");
        map.insert ("pid",elems[0]); //pid
        map.insert ("comm",elems[1].mid(elems[1].lastIndexOf
("(")+1,elems[1].indexOf (")")-1)); //comm
        map.insert ("task_state",elems[2]); //task_state
        map.insert ("ppid",elems[3]); //ppid
        map.insert ("priority",elems[17]); //priority
        map.insert ("vsize",elems[23]); //rss
//        qDebug()<<elems[0]<<" "<<elems[1]<<" "<<elems[2]<<"
"<<elems[3]<<" "<<elems[16]<<" "<<elems[21];
        mapList.push_back (map);
    }
}
return mapList;
}

```

/**

* cpu rate

* @brief Util::CPU_Utilization_Rate

* @return

*/

```

QString Util::CPU_Utilization_Rate(){
    QString content = file_content ("/proc/stat","cpu");
    QStringList elems=content.split (" ");
    double usage=0;
    double idle2;

```

```

double total2;
double user=elems[2].toDouble ();
double nice=elems[3].toLong ();
double system=elems[4].toLong ();
double idle=elems[5].toLong ();
double iowait=elems[6].toLong ();
double irq=elems[7].toLong ();
double softirq=elems[8].toLong ();
double total = user+nice+system+idle+iowait+irq+softirq;
QThread::msleep(30);
content=file_content ("/proc/stat","cpu");
//   qDebug()<<content;
elems=content.split (" ");
user=elems[2].toLong ();
//   qDebug()<<elems[2];
nice=elems[3].toLong ();
system=elems[4].toLong ();
idle2=elems[5].toLong ();
iowait=elems[6].toLong ();
irq=elems[7].toLong ();
softirq=elems[8].toLong ();
total2 = user+nice+system+idle2+iowait+irq+softirq;
usage=(1-(idle2-idle)/(total2-total))*100.0;

//   qDebug()<<elems[2]<<" "<<elems[3]<<" "<<elems[4]<<" "<<elems[5]<<"
"<<elems[6]<<" "<<elems[7]<<" "<<elems[8]<<" ";
//   qDebug()<<idle2-idle<<" "<<total2-total;
if(usage<0) usage=-usage;
//   qDebug()<<usage;
QString result=QString::number(usage,10,2);
return result;
}

QStringList Util::cpu_rates(){
    QString content = file_content ("/proc/stat","cpu");

```

```

        QStringList elems=content.split (" ");
        double user=elems[2].toDouble ();
        double nice=elems[3].toLong ();
        double system=elems[4].toLong ();
        double idle=elems[5].toLong ();
        double iowait=elems[6].toLong ();
        double irq=elems[7].toLong ();
        double softirq=elems[8].toLong ();
        double total = user+nice+system+idle+iowait+irq+softirq;
        QStringList result;
        result.push_back (QString::number (total));
        result.push_back (QString::number (idle));
        return result;
    }
/**
 * label set text
 * @brief Util::setText
 * @param o
 * @param text
 */
void Util::setText(QLabel* o,QString text){
    o->setText(text==nullptr?"":text);
}

/**
 * memory used rates;
 * @brief Util::mem_rate
 * @return
 */
double Util::mem_rate (){
    QStringList argv={"MemTotal","MemFree","Buffers","Cached"};
    QString content = file_content ("/proc/meminfo",argv);
    QStringList contents=content.split ("\n");
    double MemTotal = contents[0].split (":")[1].trimmed ().split (" ")[0].toDouble
    ();

```

```

double MemFree  = contents[1].split(":")[1].trimmed().split(" ")[0].toDouble
();
double Buffers  = contents[2].split(":")[1].trimmed().split(" ")[0].toDouble ();
double Cached   = contents[3].split(":")[1].trimmed().split(" ")[0].toDouble
();
qDebug()<<MemTotal<<" "<<MemFree<<" "<<Buffers<<" "<<Cached<<" ";
double result=0.0;
result=(MemTotal-MemFree-Buffers-Cached)/MemTotal*100;
qDebug()<<result;
return result;
}

```