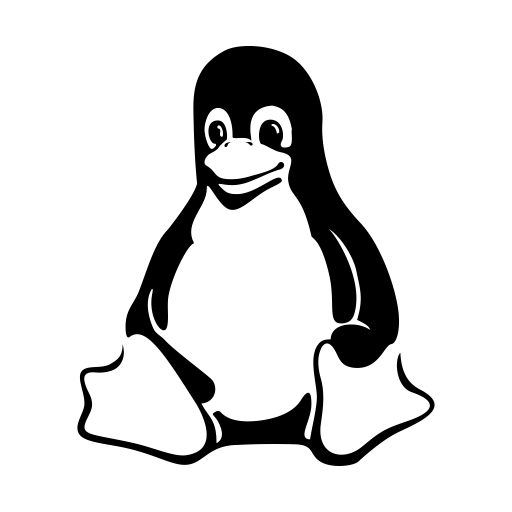
Linux高级程序设计

Qt简易Linux任务管理器



学号：0x091395

姓名：uknowho

年级：2014

目录

[一. 项目概述 - 1 -](#_Toc482962844)

[二. 需求分析 - 1 -](#_Toc482962845)

[三. 功能设计 - 2 -](#_Toc482962846)

[四. 图例示意 - 3 -](#_Toc482962847)

[五. 功能实现 - 4 -](#_Toc482962848)

[六. 程序运行结果 - 8 -](#_Toc482962849)

[七. 源代码 - 9 -](#_Toc482962850)

# 项目概述

QT是一个跨平台的C++应用程序开发框架。广泛用于开发GUI程序，这种情况下又被称为部件工具箱。也可用于开发非GUI程序，比如控制台工具和服务器。Qt使用于OPIE、Skype、VLC media player、Adobe Photoshop Elements、VirtualBox与Mathematica[7]以及被Autodesk [8][9]、欧洲空间局[10]、梦工厂[11][12]、Google、HP[13]、KDE、卢卡斯影业[14]、西门子公司[15]、富豪集团[16], 华特迪士尼动画制作公司[17]、三星集团[18]、飞利浦[19]、Panasonic 所使用。

Qt是自由且开放源代码的软件，在GNU宽通用公共许可证（LGPL）条款下发布。所有版本都支持广泛的编译器，包括GCC的C++编译器和Visual Studio。

Qt Creator 是一款跨平台的集成开发环境，特别针对Qt开发者，是Qt SDK组成的一部分，可运行于Windows, Linux/X11及Mac OS X等桌面操作系统，允许开发者为多桌面环境及移动设备平台创建应用程序。它包括一个可视化调试工具和集成的 GUI 版面和外形设计师。这个编辑器的功能包括语法高亮度显示和自动完成。[4] Qt Creator 在 Linux 上，使用 GCC 的 C++ 编译器。在 Windows，默认安装它可以使用 MinGW 或 MSVC。

常见的任务管理器有

任务管理器，适用于Microsoft Windows

活动监视器，适用于Mac OS X（Mac OS X v10.3称为进程查看器）

ps (Unix)

top (软件)，大多数类Unix系统上的实用工具，及之后其他人模仿的原型。

tasklist，适用于MS-DOS

htop，适用于Linux系统

开发环境：

Ubuntu16.04LST

QT5.8

# 需求分析

我们在使用Windows操作系统的时候，任务管理器是一个方便我们管理系统中正在使用或运行的进程或任务的软件。任任务管理器可以是用于监视计算机上进程和程序活动，以及查看计算机的全局状态的系统监视器软件。也可以仅仅内建于软件之中，用于监视该软件自身活动。

任务管理器可以显示正在运行和已终止运行的后台服务及进程，可以实现终止进程和程序的功能，也能改变进程的优先级。在信息完整的情况下，其可显示服务的相关信息（如进程ID和组）。

而在我们使用Linux操作系统的时候，也希望通过一个管理器，实现对系统进程任务的管理，并且可以实现查看系统内存的使用、CPU的信息、系统的信息等功能，同时也能通过可视化的图形界面实现信息的显示和功能的操作，避免了命令模式的繁杂，提高可用性和易用性。此次开发Qt简易Linux任务管理器，就实现简单的类似Windows任务管理器的一个软件，这样我们在使用Linux操作系统的时候，通过这个管理器，就可以实现对操作系统信息基本的操作和管理。

# 功能设计

任务管理器应该至少包含有以下功能：

1.查看当前系统下的所有进程，在Linux下包含有 进程名称，进程所占内存，进程PID等。

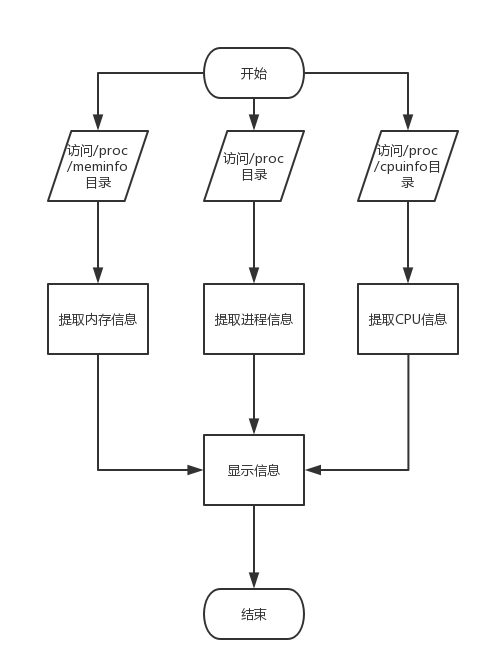
2.能够结束所选进程。

3.查看系统的内存信息。

4.查看系统CPU信息。

5.查看系统信息。

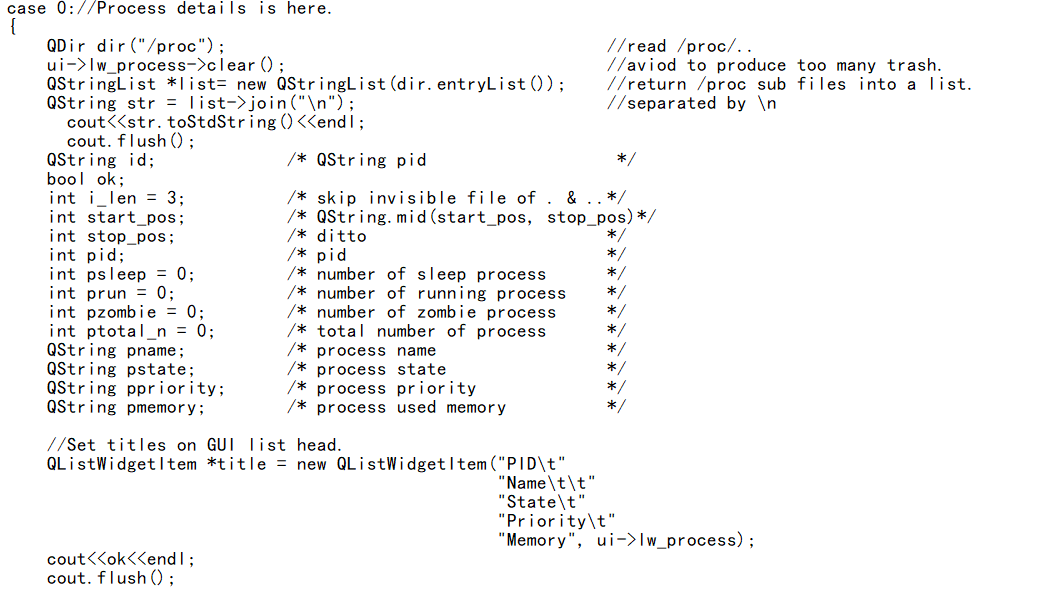
# 图例示意



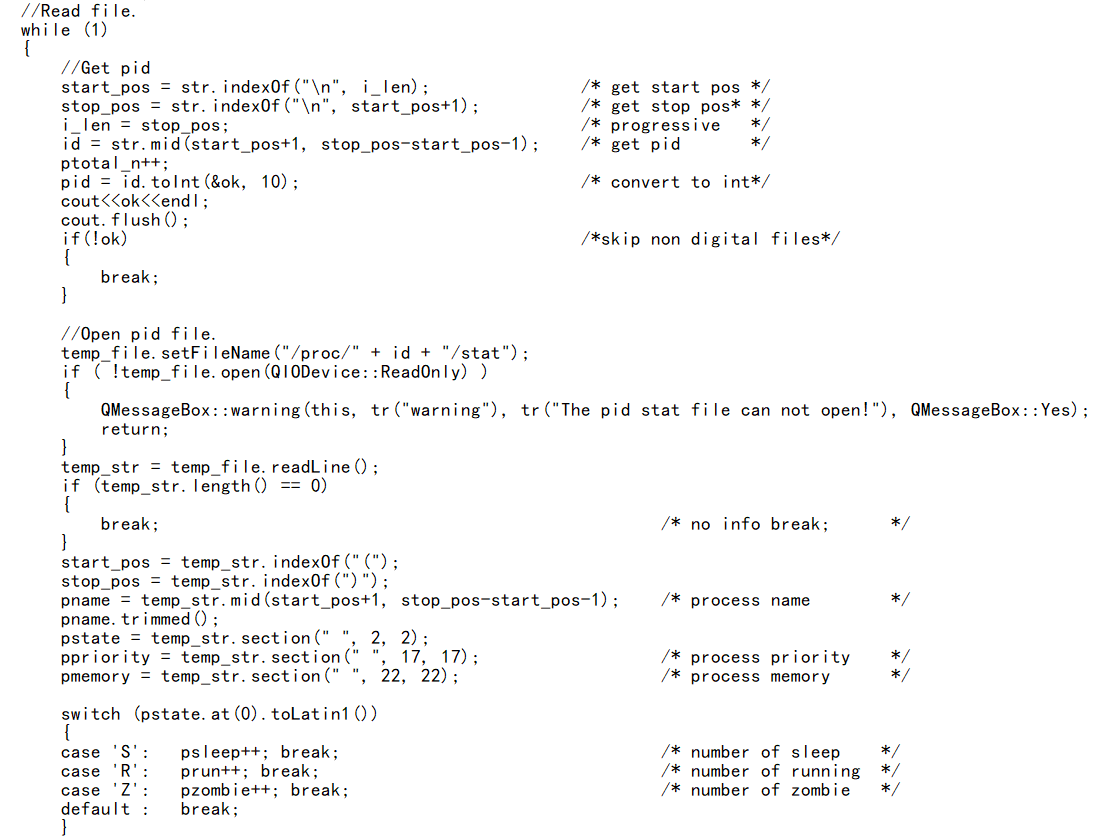
# 功能实现

**显示当前系统下所有进程，包括进程名、所占内存、PID**

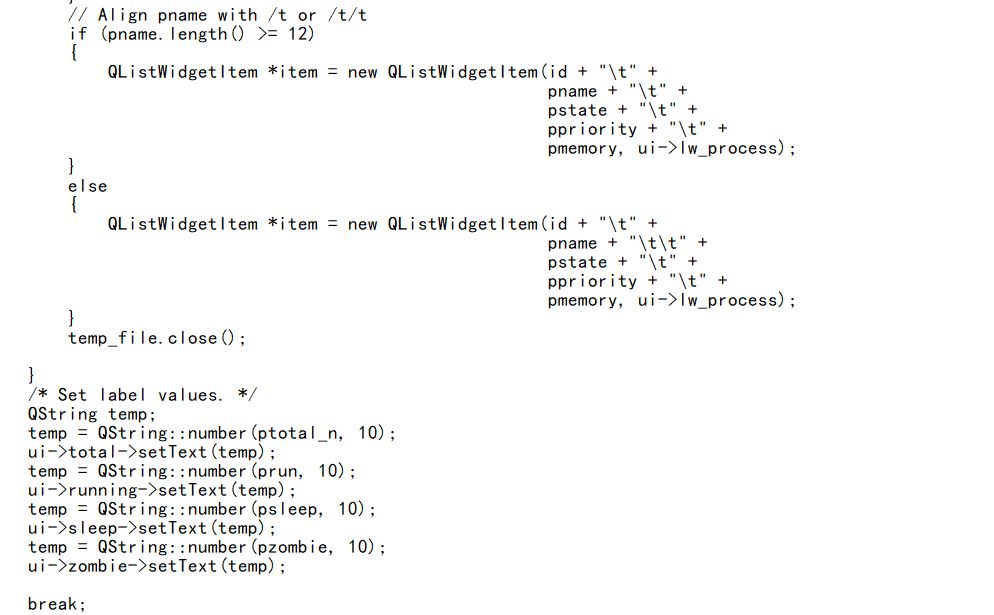
Linux系统的/proc目录下存放着系统所有进程的信息，通过访问此目录，提取出进程信息的内容。类MainWindow中switch语句的case 0中实现此功能，如下图4-1、4-2、4-3所示：



**图4-1 case 0 语句（1）**

****

**图4-2 case 0 语句（2）**

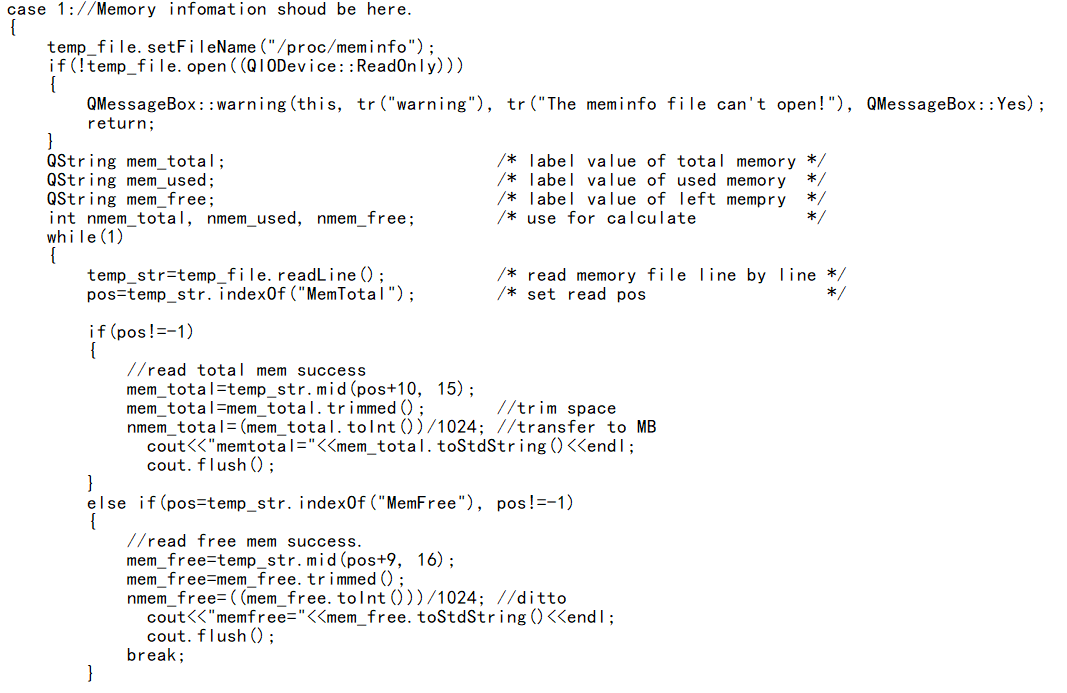
****

**图4-3 case 0 语句（3）**

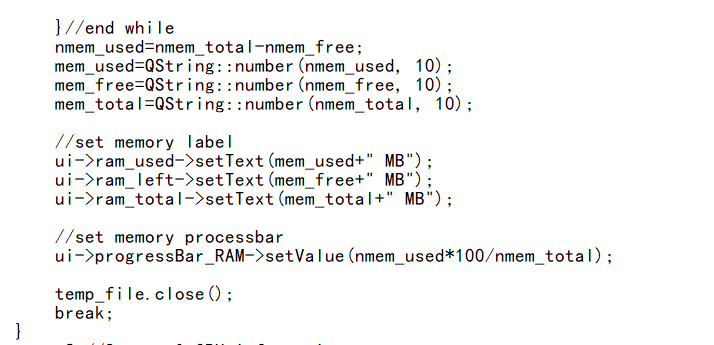
这样就可以在管理器中访问进程的所有信息，通过MainWindow显示出来。

1. **显示系统内存使用信息、CPU占用率**

在/proc目录下读取/meminfo目录中的内容，就可以获取内存的使用信息。在类MainWindow中switch语句的case 1实现此功能，如下图4-4、4-5所示|



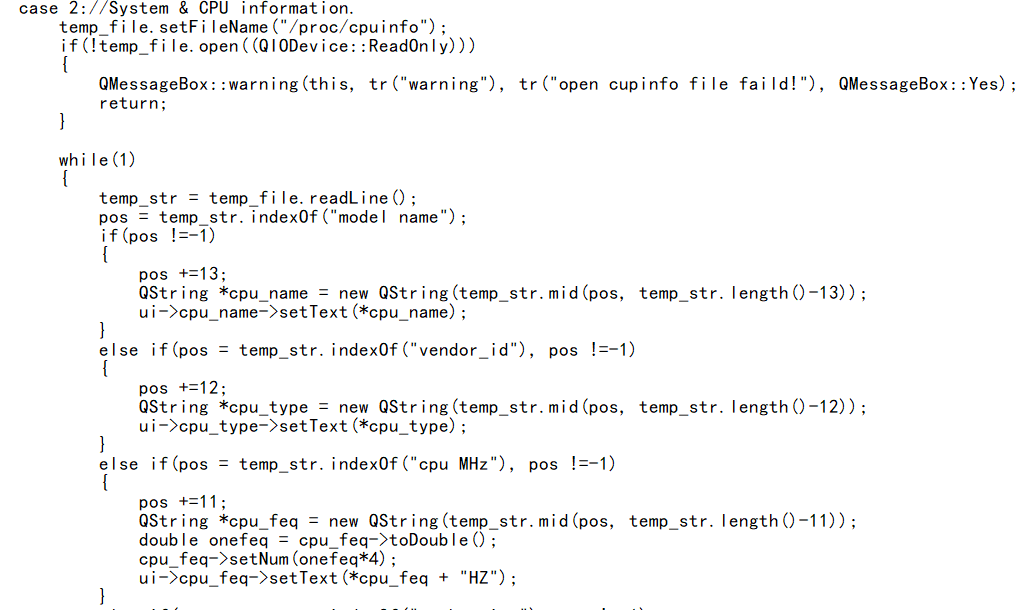
**图4-4 case 1 语句（1）**



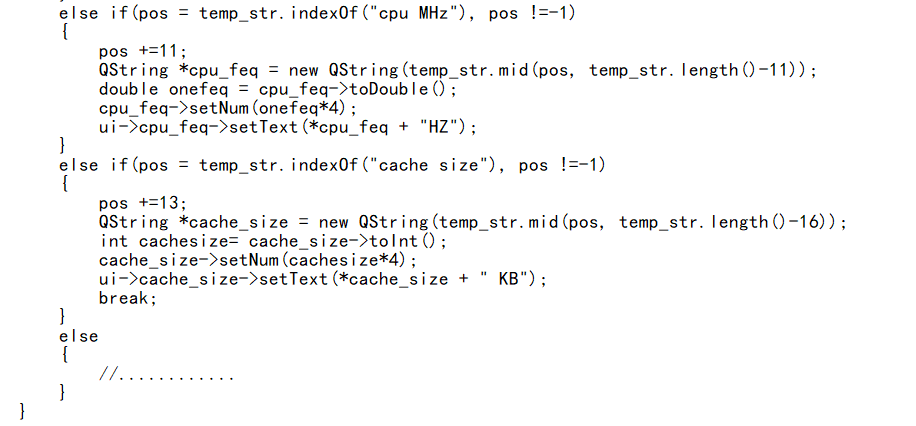
**图4-5 case 1 语句（2）**

1. **显示当前系统版本等信息**

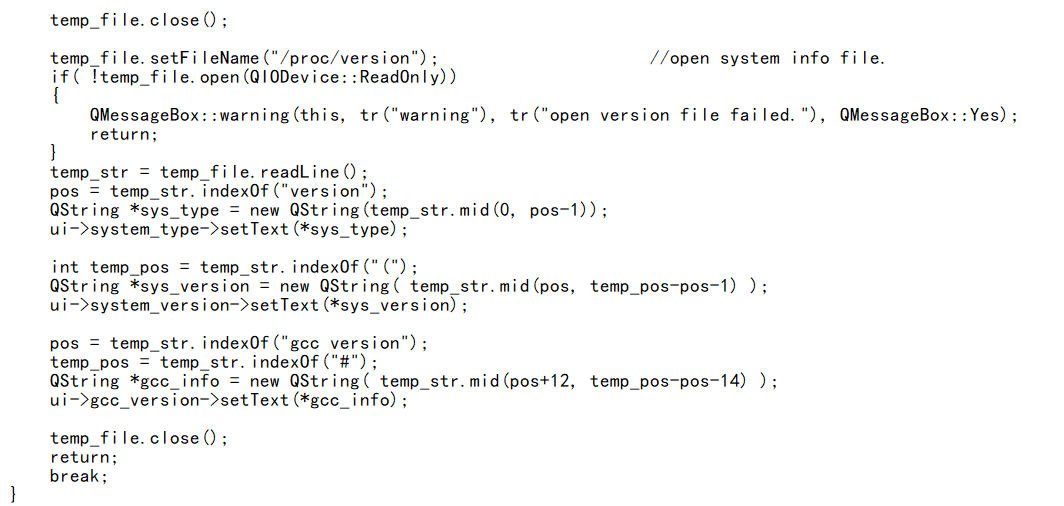
在/proc目录下读取/cpuinfo目录中的内容，就可以获取系统和CPU的信息。在类MainWindow中switch语句的case 2实现此功能，如下图4-6、4-7、4-8所示：



**图4-6 case 2 语句（1）**

****

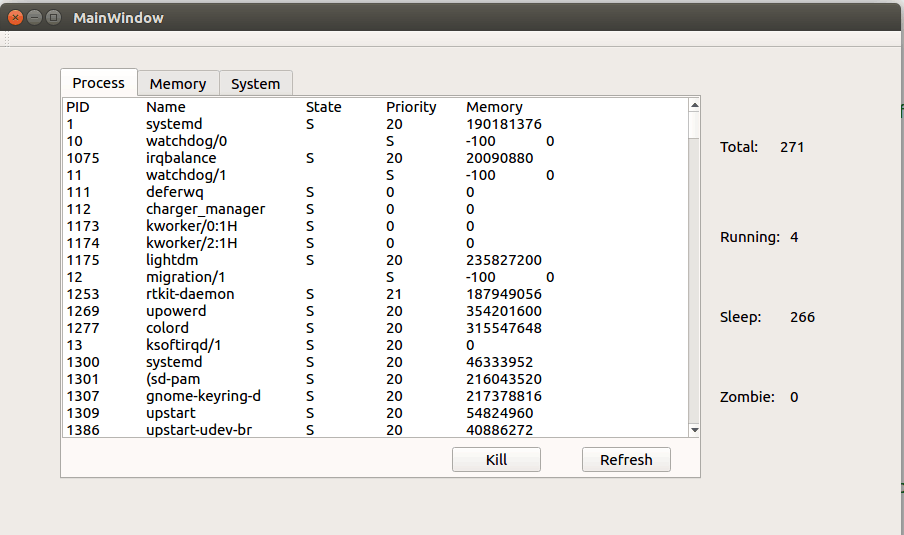
**图4-7 case 2 语句（2）**

****

**图4-8 case 2 语句（3）**

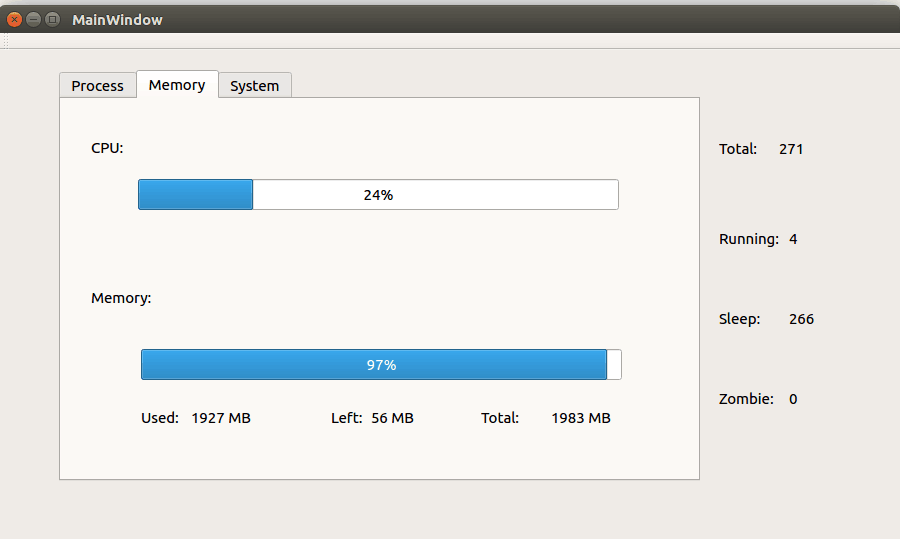
# 程序运行结果

显示进程信息，如图5-1所示：



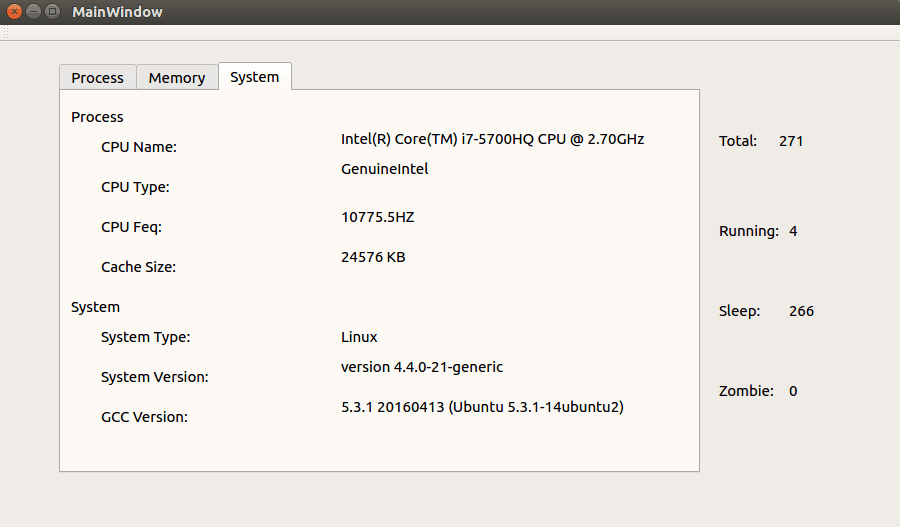
**图5-1 进程信息**

显示内存、CPU信息，如图5-2所示：



**图5-2 内存、CPU信息**

显示当前系统信息，如图5-3所示：



**图5-3 系统信息**

# 源代码

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QFile>

#include <QListWidget>

#include <QListWidgetItem>

#include <QStringList>

#include <QMessageBox>

#include <QDir>

#include <iostream>

using namespace std;

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

QWidget::connect(ui->tabWidget, SIGNAL(currentChanged(int)), this, SLOT(on\_tabWidget\_currentChanged(int)));

show\_tab\_info(0);

show\_tab\_info(1);

show\_tab\_info(2);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::timer\_update()

{

int index = ui->tabWidget->currentIndex();

if (index == 0)

{

show\_tab\_info(index);

}

}

void MainWindow::show\_tab\_info(int index)

{

QString temp\_str; /\* read a line of temp file \*/

QFile temp\_file; /\* open file \*/

int pos; /\* pos of reading file \*/

switch (index) {

case 0://Process details is here.

{

QDir dir("/proc"); //read /proc/..

ui->lw\_process->clear(); //aviod to produce too many trash.

QStringList \*list= new QStringList(dir.entryList()); //return /proc sub files into a list.

QString str = list->join("\n"); //separated by \n

// cout<<str.toStdString()<<endl;

// cout.flush();

QString id; /\* QString pid \*/

bool ok;

int i\_len = 3; /\* skip invisible file of . & ..\*/

int start\_pos; /\* QString.mid(start\_pos, stop\_pos)\*/

int stop\_pos; /\* ditto \*/

int pid; /\* pid \*/

int psleep = 0; /\* number of sleep process \*/

int prun = 0; /\* number of running process \*/

int pzombie = 0; /\* number of zombie process \*/

int ptotal\_n = 0; /\* total number of process \*/

QString pname; /\* process name \*/

QString pstate; /\* process state \*/

QString ppriority; /\* process priority \*/

QString pmemory; /\* process used memory \*/

//Set titles on GUI list head.

QListWidgetItem \*title = new QListWidgetItem("PID\t"

"Name\t\t"

"State\t"

"Priority\t"

"Memory", ui->lw\_process);

cout<<ok<<endl;

cout.flush();

//Read file.

while (1)

{

//Get pid

start\_pos = str.indexOf("\n", i\_len); /\* get start pos \*/

stop\_pos = str.indexOf("\n", start\_pos+1); /\* get stop pos\* \*/

i\_len = stop\_pos; /\* progressive \*/

id = str.mid(start\_pos+1, stop\_pos-start\_pos-1); /\* get pid \*/

ptotal\_n++;

pid = id.toInt(&ok, 10); /\* convert to int\*/

cout<<ok<<endl;

cout.flush();

if(!ok) /\*skip non digital files\*/

{

break;

}

//Open pid file.

temp\_file.setFileName("/proc/" + id + "/stat");

if ( !temp\_file.open(QIODevice::ReadOnly) )

{

QMessageBox::warning(this, tr("warning"), tr("The pid stat file can not open!"), QMessageBox::Yes);

return;

}

temp\_str = temp\_file.readLine();

if (temp\_str.length() == 0)

{

break; /\* no info break; \*/

}

start\_pos = temp\_str.indexOf("(");

stop\_pos = temp\_str.indexOf(")");

pname = temp\_str.mid(start\_pos+1, stop\_pos-start\_pos-1); /\* process name \*/

pname.trimmed();

pstate = temp\_str.section(" ", 2, 2);

ppriority = temp\_str.section(" ", 17, 17); /\* process priority \*/

pmemory = temp\_str.section(" ", 22, 22); /\* process memory \*/

switch (pstate.at(0).toLatin1())

{

case 'S': psleep++; break; /\* number of sleep \*/

case 'R': prun++; break; /\* number of running \*/

case 'Z': pzombie++; break; /\* number of zombie \*/

default : break;

}

// Align pname with /t or /t/t

if (pname.length() >= 12)

{

QListWidgetItem \*item = new QListWidgetItem(id + "\t" +

pname + "\t" +

pstate + "\t" +

ppriority + "\t" +

pmemory, ui->lw\_process);

}

else

{

QListWidgetItem \*item = new QListWidgetItem(id + "\t" +

pname + "\t\t" +

pstate + "\t" +

ppriority + "\t" +

pmemory, ui->lw\_process);

}

temp\_file.close();

}

/\* Set label values. \*/

QString temp;

temp = QString::number(ptotal\_n, 10);

ui->total->setText(temp);

temp = QString::number(prun, 10);

ui->running->setText(temp);

temp = QString::number(psleep, 10);

ui->sleep->setText(temp);

temp = QString::number(pzombie, 10);

ui->zombie->setText(temp);

break;

}

case 1://Memory infomation shoud be here.

{

temp\_file.setFileName("/proc/meminfo");

if(!temp\_file.open((QIODevice::ReadOnly)))

{

QMessageBox::warning(this, tr("warning"), tr("The meminfo file can't open!"), QMessageBox::Yes);

return;

}

QString mem\_total; /\* label value of total memory \*/

QString mem\_used; /\* label value of used memory \*/

QString mem\_free; /\* label value of left mempry \*/

int nmem\_total, nmem\_used, nmem\_free; /\* use for calculate \*/

while(1)

{

temp\_str=temp\_file.readLine(); /\* read memory file line by line \*/

pos=temp\_str.indexOf("MemTotal"); /\* set read pos \*/

if(pos!=-1)

{

//read total mem success

mem\_total=temp\_str.mid(pos+10, 15);

mem\_total=mem\_total.trimmed(); //trim space

nmem\_total=(mem\_total.toInt())/1024; //transfer to MB

// cout<<"memtotal="<<mem\_total.toStdString()<<endl;

// cout.flush();

}

else if(pos=temp\_str.indexOf("MemFree"), pos!=-1)

{

//read free mem success.

mem\_free=temp\_str.mid(pos+9, 16);

mem\_free=mem\_free.trimmed();

nmem\_free=((mem\_free.toInt()))/1024; //ditto

// cout<<"memfree="<<mem\_free.toStdString()<<endl;

// cout.flush();

break;

}

}//end while

nmem\_used=nmem\_total-nmem\_free;

mem\_used=QString::number(nmem\_used, 10);

mem\_free=QString::number(nmem\_free, 10);

mem\_total=QString::number(nmem\_total, 10);

//set memory label

ui->ram\_used->setText(mem\_used+" MB");

ui->ram\_left->setText(mem\_free+" MB");

ui->ram\_total->setText(mem\_total+" MB");

//set memory processbar

ui->progressBar\_RAM->setValue(nmem\_used\*100/nmem\_total);

temp\_file.close();

break;

}

case 2://System & CPU information.

temp\_file.setFileName("/proc/cpuinfo");

if(!temp\_file.open((QIODevice::ReadOnly)))

{

QMessageBox::warning(this, tr("warning"), tr("open cupinfo file faild!"), QMessageBox::Yes);

return;

}

while(1)

{

temp\_str = temp\_file.readLine();

pos = temp\_str.indexOf("model name");

if(pos !=-1)

{

pos +=13;

QString \*cpu\_name = new QString(temp\_str.mid(pos, temp\_str.length()-13));

ui->cpu\_name->setText(\*cpu\_name);

}

else if(pos = temp\_str.indexOf("vendor\_id"), pos !=-1)

{

pos +=12;

QString \*cpu\_type = new QString(temp\_str.mid(pos, temp\_str.length()-12));

ui->cpu\_type->setText(\*cpu\_type);

}

else if(pos = temp\_str.indexOf("cpu MHz"), pos !=-1)

{

pos +=11;

QString \*cpu\_feq = new QString(temp\_str.mid(pos, temp\_str.length()-11));

double onefeq = cpu\_feq->toDouble();

cpu\_feq->setNum(onefeq\*4);

ui->cpu\_feq->setText(\*cpu\_feq + "HZ");

}

else if(pos = temp\_str.indexOf("cache size"), pos !=-1)

{

pos +=13;

QString \*cache\_size = new QString(temp\_str.mid(pos, temp\_str.length()-16));

int cachesize= cache\_size->toInt();

cache\_size->setNum(cachesize\*4);

ui->cache\_size->setText(\*cache\_size + " KB");

break;

}

else

{

//............

}

}

temp\_file.close();

temp\_file.setFileName("/proc/version"); //open system info file.

if( !temp\_file.open(QIODevice::ReadOnly))

{

QMessageBox::warning(this, tr("warning"), tr("open version file failed."), QMessageBox::Yes);

return;

}

temp\_str = temp\_file.readLine();

pos = temp\_str.indexOf("version");

QString \*sys\_type = new QString(temp\_str.mid(0, pos-1));

ui->system\_type->setText(\*sys\_type);

int temp\_pos = temp\_str.indexOf("(");

QString \*sys\_version = new QString( temp\_str.mid(pos, temp\_pos-pos-1) );

ui->system\_version->setText(\*sys\_version);

pos = temp\_str.indexOf("gcc version");

temp\_pos = temp\_str.indexOf("#");

QString \*gcc\_info = new QString( temp\_str.mid(pos+12, temp\_pos-pos-14) );

ui->gcc\_version->setText(\*gcc\_info);

temp\_file.close();

return;

break;

}

}

// Refresh every tab when it's changed.

void MainWindow::on\_tabWidget\_currentChanged(int index)

{

show\_tab\_info(index);

}

//Manual refresh

void MainWindow::on\_btn\_refresh\_clicked()

{

show\_tab\_info(0);

}

//Kill a process

void MainWindow::on\_btn\_kill\_clicked()

{

QListWidgetItem \*item = ui->lw\_process->currentItem(); /\* get selected item \*/

QString pro = item->text(); /\* get item's content \*/

pro = pro.section("\t", 0, 0); /\* get process name \*/

system("kill " + pro.toLatin1()); /\* Oh yes! \*/

QMessageBox::warning(this, tr("kill"), QString::fromUtf8("Process has been killed"), QMessageBox::Yes);

//refresh process list

show\_tab\_info(0);

}