**西南大学**

**计算机与信息科学学院**

操作系统课程设计报告

**题 目：**  基于proc文件系统的任务管理器

**年级、专业：** 2019 **级**  网络工程 **专业**  2 **班**

**学生姓名：**  周亮

**指导教师：**  廖剑伟

**提交日期：**  2022 **年** 1 **月** 7 **日**

|  |
| --- |
| **操作系统课程设计报告评定：** |
| **成绩：**  **指导教师（签字）：廖剑伟**  **2022年 1月 15 日** |

**目录**

[一、问题描述与需求分析 1](#_Toc90383533)

[1.1 问题描述 1](#_Toc90383534)

[1.2 需求分析 1](#_Toc90383535)

[二、设计原理 1](#_Toc90383536)

[2.1 /proc下文件分析 1](#_Toc90383537)

[2.2 进程管理 6](#_Toc90383538)

[2.3 性能监控 6](#_Toc90383539)

[2.4 图形化处理 7](#_Toc90383540)

[三、总体设计 7](#_Toc90383541)

[3.1 设计思路 7](#_Toc90383542)

[3.2 设计方案 8](#_Toc90383543)

[四、算法分析 12](#_Toc90383544)

[4.1 程序流程图 12](#_Toc90383545)

[4.2 算法流程图及分析 12](#_Toc90383546)

[五、代码分析 17](#_Toc90383547)

[5.1 函数功能 17](#_Toc90383548)

[5.2 函数实现 18](#_Toc90383549)

[5.3 程序流程分析 25](#_Toc90383550)

[六、运行结果分析 26](#_Toc90383551)

[6.1 编译器与运行环境 26](#_Toc90383552)

[6.2 功能演示 26](#_Toc90383553)

[七、总结 34](#_Toc90383554)

[八、参考资料 35](#_Toc90383555)

[附录：源代码 36](#_Toc90383556)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **基于proc文件系统的任务管理器** 一、问题描述与需求分析1.1 问题描述 Windows操作系统中，自带一个图形化任务管理器，可以查看、管理进程，监控系统硬件状态等。在Linux操作系统中对于查看不同系统资源使用，比如进程、CPU、内存等，需要使用不同命令来进行查看，和Windows相比比较麻烦。现需要设计一个Linux下的任务管理器，并满足以下条件：   1. 了解/proc文件的特点和使用方法； 2. 监控系统状态，显示系统中若干部件的使用情况； 3. 用图形界面实现系统监控状态。  1.2 需求分析 随着互联网的普及，越来越多的互联网从业者涌入Linux这个大家庭；在Linux操作系统中，所有功能都是靠命令来使用，随着图形化桌面的使用，各类Windows下使用的软件也能跨平台在Linux下使用，但Linux始终没有一款属于自己的图形化的任务管理器，为使用Linux图形化桌面的用户提供更加便捷的服务。本课程设计基于Linux下/proc文件系统设计了类似于Windows任务管理器的简易Linux任务管理器，能为非专业Linux使用者提供基础的监控Linux进程、CPU、内存、磁盘和网络的任务管理工具。 二、设计原理2.1 /proc下文件分析 Linux系统上的/proc目录是一种文件系统，即proc文件系统。与其它常见的文件系统不同的是，/proc是一种伪文件系统（也即虚拟文件系统），存储的是当前内核运行状态的一系列特殊文件，用户可以通过这些文件查看有关系统硬件及当前正在运行进程的信息，甚至可以通过更改其中某些文件来改变内核的运行状态，其内的文件也常被称作虚拟文件，并具有一些独特的特点。例如，其中有些文件虽然使用查看命令查看时会返回大量信息，但文件本身的大小却会显示为0字节。此外，这些特殊文件中大多数文件的时间及日期属性通常为当前系统时间和日期，这跟它们随时会被刷新有关。 2.1.1 进程 /proc/[pid]/cmdline: 一个只读文件，包含进程的完整命令行信息  /proc/[pid]/comm: 包含进程的命令名  /proc/[pid]/cwd: 是进程当前工作目录的符号链接  /proc/[pid]/environ: 显示进程的环境变量  /proc/[pid]/exe: 为实际运行程序的符号链接  /proc/[pid]/fd: 一个目录，包含进程打开文件的情况  /proc/[pid]/latency: 显示哪些代码造成的延时比较大  /proc/[pid]/maps: 显示进程的内存区域映射信息  /proc/[pid]/root: 是进程根目录的符号链接  /proc/[pid]/stack: 示当前进程的内核调用栈信息  /proc/[pid]/statm: 显示进程所占用内存大小的统计信息  /proc/[pid]/status: 包含进程的状态信息  /proc/[pid]/syscall: 显示当前进程正在执行的系统调用  /proc/[pid]/wchan: 显示当进程 sleep 时，kernel 当前运行的函数 2.1.2 CPU **/proc/cpuinfo文件** 该文件中存放了有关 cpu的相关信息(型号，缓存大小等)。   * processor 逻辑处理器的id。 * physical id 物理封装的处理器的id。 * core id 每个核心的id。 * cpu cores 位于相同物理封装的处理器中的内核数量。 * siblings 位于相同物理封装的处理器中的逻辑处理器的数量。   **/proc/stat文件**  该文件包含了所有CPU活动的信息，该文件中的所有值都是从系统启动开始累计到当前时刻。   | **cpu指标** | **含义** | **单位** | **备注** | | --- | --- | --- | --- | | user | 用户态时间 | jiffies | 一般/高优先级，仅统计nice<=0 | | nice | nice用户态时间 | jiffies | 低优先级，仅统计nice>0 | | system | 内核态时间 | jiffies |  | | idle | 空闲时间 | jiffies | 不包含IO等待时间 | | iowait | I/O等待时间 | jiffies | 硬盘IO等待时间 | | irq | 硬中断时间 | jiffies |  | | softirq | 软中断时间 | jiffies |  | | steal | 被盗时间 | jiffies | 虚拟化环境中运行其他操作系统上花费的时间 | | guest | 来宾时间 | jiffies | 操作系统运行虚拟CPU花费的时间 | | guest\_nice | nice来宾时间 | jiffies | 运行一个带nice值的guest花费的时间 |  2.1.3 内存 **/proc/meminfo文件**   * MemTotal：所有可用的内存大小，物理内存减去预留位和内核使用 * MemFree：表示系统尚未使用的内存(free命令看到的没有这个准确) * MemAvailable：真正的系统可用内存，系统中有些内存虽然已被使用但是可以回收的，比如cache/buffer、slab（MemAvailable=可回收+MemFree） * Buffers：用来给块设备做缓存的内存（文件系统的 metadata、pages) * Cached：分配给文件缓冲区的内存（比如未保存的文件就是写到该缓冲区） * SwapCached：被高速缓冲存储用的交换空间（硬盘的swap）的大小 * Active：经常使用的高速缓冲存储器页面文件大小 * Inactive：不经常使用的高速缓冲存储器文件大小 * Active(anon)：活跃的匿名内存 * Inactive(anon)：不活跃的匿名内存 * Active(file)：活跃的文件使用内存 * Inactive(file)：不活跃的文件使用内存 * Unevictable：不能被释放的内存页 * Mlocked：系统调用 mlock 家族允许程序在物理内存上锁住它的部分或全部地址空间。这将阻止Linux 将这个内存页调度到交换空间（swap space），即使该程序已有一段时间没有访问这段空间 * HighTotal：所有在 860MB（0x35C00000）以上的空间。主要是用户空间程序或缓存页 * HighFree:： 860MB 以下的空间。如果该空间用完了，系统可能会异常 * LowTotal：860MB 以上空间的可用空间 * LowFree：860MB 以下空间的可用空间 * SwapTotal：交换空间总内存 * SwapFree：交换空间空闲内存 * Dirty：等待被写回到磁盘的 * Writeback：正在被写回的 * AnonPages：未映射页的内存/映射到用户空间的非文件页表大小 * Mapped：映射文件内存 * Shmem：已经被分配的共享内存 * Slab：内核数据结构缓存 * SReclaimable：可收回slab内存 * SUnreclaim：不可收回slab内存 * KernelStack：内核消耗的内存 * PageTables：管理内存分页的索引表的大小 * NFS\_Unstable：不稳定页表的大小 * Bounce：在低端内存中分配一个临时buffer作为跳转，把位于高端内存的缓存数据复制到此处消耗的内存 * WritebackTmp：FUSE用于临时写回缓冲区的内存 * CommitLimit：系统实际可分配内存 * Committed\_AS：系统当前已分配的内存 * VmallocTotal：预留的虚拟内存总量 * VmallocUsed：已经被使用的虚拟内存 * VmallocChunk：可分配的最大的逻辑连续的虚拟内存 * CmaTotal：连续可用内存总量 * CmaFree：空闲连续可用内存  2.1.4 磁盘 **/proc/diskstats文件**  输出每列对应值：  设备号 编号 设备 读完成次数 合并完成次数 读扇区次数 读操作花费毫秒数 写完成次数 合并写完成次数 写扇区次数 写操作花费的毫秒数 正在处理的输入/输出请求数 输入/输出操作花费的毫秒数 输入/输出操作花费的加权毫秒数。  注意：除正在处理的输入/输出请求数这项是非累积值外，其他磁盘统计都是累积值。 2.1.5网络 **proc/net/dev文件**  网络适配器及统计信息   * bytes:接口发送或接收的数据的总字节数 * packets:接口发送或接收的数据包总数 * errs:由设备驱动程序检测到的发送或接收错误的总数 * drop:设备驱动程序丢弃的数据包总数 * fifo: FIFO缓冲区错误的数量 * frame:分组帧错误的数量 * colls:接口上检测到的冲突数 * compressed:设备驱动程序发送或接收的压缩数据包数 * carrier:由设备驱动程序检测到的载波损耗的数量 * multicast:设备驱动程序发送或接收的多播帧数   **/proc/net/arp文件**  每个网络接口的arp表中dev包的统计  每列字段含义：  IP address HW type Flags HW address Mask Device   * IP address：IP地址（直连） * HW type：硬件类型 * 23=0x17 strip (Metricom Starmode IP) * 01=0x01 ether (Ethernet) * 15=0xf dlci (Frame Relay DLCI) * Flags： * HW address：MAC 地址 * Mask： * Device：所在网络接口  2.2 进程管理 /proc目录中包含许多以数字命名的子目录，这些数字表示系统当前正在运行进程的进程号，子目录里面包含对应进程相关的多个信息文件，包括进程名、进程使用内存大小，进程状态、进程占用CPU情况等等。程序可以读取每个进程相应文件，计算该进程各项状态，最后存储统计。 2.3 性能监控 性能监控包括CPU、内存、磁盘和网络的使用情况，通过对/proc目录下cpuinfo文件读取可以获取到当前计算机CPU型号、性能等等；通过周期性读取/proc目录下stat文件，可以计算出当前CPU的使用情况。通过对/proc目录下meminfo文件读取，可以获取当前内存使用情况，需要注意显示单位为页或者Byte，后需要换算单位；通过对/proc目录下diskstats文件读取，可以获取磁盘使用情况，显示的是一个累计值，需要换算后才能得出当前磁盘使用情况；通过对/proc/net目录下dev文件读取可以获取当前网卡信息；/proc/net目录下arp文件保存了用于地址解析的内核ARP表的ASCII可读转储，将显示动态学习和预编程的ARP条目，读取文件可以获取到通过网卡接收和发送的包的字节数，从而计算当前网络使用情况。 2.4 图形化处理 使用QcustomPlot绘制系统性能的曲线图。QCustomPlot是一个用于绘图和数据可视化的Qt C++构件。 它没有进一步的依赖关系，并有很好的文档记录。 该绘图库专注于制作美观，出版品质的2D图表，图表和图表，以及为实时可视化应用程序提供高性能。 查看“设置”和“基本绘图”教程以开始。QCustomPlot可以导出为各种格式，如矢量化的PDF文件和光栅化图像，如PNG，JPG和BMP。 QCustomPlot是在应用程序内部显示实时数据以及为其他媒体生成高质量图的解决方案。本课程设计里简单使用QcustomPlot通过实时计算的系统CPU、内存、磁盘和网络使用情况绘制相应曲线图，简单实现了类似Windows操作系统的任务管理器的样式。 三、总体设计3.1 设计思路 程序主要分为六个模块：一是对进程进行监控，包括显示进程基本信息、杀死指定进程等；二是对CPU性能进行监控，通过曲线图反应实时CPU使用率，还包括系统进程数、运行时间等；三是对内存使用情况监控，实时反应内存使用情况；四是对磁盘读写情况的监控，实时反应磁盘IO使用情况；五是对网络使用情况的监控，根据接收、发送数据包大小实时反应网络使用情况；六是一些系统管理功能，包括杀死进程、关机和重启计算机等。   3.2 设计方案 **进程监控模块：**  启动程序显示进程页面，通过选项卡可以切换为监控进程还是性能；进程页面使用列表显示当前进程，并会显示进程PID号、进程状态、CPU占用率、内存占用率、优先级和进程名称。    **CPU监控模块：**  点击性能下CPU选项卡进入CPU监控页面，标题显示CPU型号等信息，中间部分为CPU实时使用率的曲线图，能够直观的反映出CPU的使用情况，下面显示CPU利用率、运行时间、进程数等等。    **内存监控模块：**  点击性能下内存选项卡进入内存监控页面，标题为“内存”，提示当前监控页面，中间部分为一个曲线图和进度条，都能直观反映出当前系统内存使用情况，末尾显示已使用内存、剩余内存等等。    **磁盘监控模块：**  点击性能下磁盘选项卡进入磁盘监控页面，标题为“磁盘”，提示当前页面监控信息，主题仍为一个曲线图反应实时磁盘使用情况。    **网络监控模块：**  点击性能下内存选项卡进入内存监控页面，主体设计基本同其它选项卡。  **系统管理模块：**  在程序主页面（进程页面）底部有三个系统管理按钮，点击“结束进程”按钮会结束当前选中的进程，点击“关机”和“重启”按钮计算机会执行关机和重启操作。   四、算法分析4.1 程序流程图  4.2 算法流程图及分析 进程定时器流程图：  先循环读取/proc目录下文件，找到文件名为数字的文件（进程），进入目录读取CPU使用时间，并将该进程PID号与时间作映射存入Map中，等待下次（1s后）读取进程后根据PID取出时间与本次事件做差，计算出CPU利用率。    CPU定时器流程图：  首先初始化时，需要将全局变量（记录的上一次函数执行CPU时间）备份一次，然后读取cpuinfo文件获取到当前系统CPU参数并存储，再一次读取stat文件每一行，获取CPU相应参数，获取CPU时间时赋值给全局变量（更新，始终保持全局变量为上一次CPU使用时间），最后更改两次时间计算这一秒CPU使用时间从而计算利用率。    内存定时器流程图：  首先初始化变量，依次读取memeinfo文件每行，匹配相应字符串再分割出占用内存和空余内存大小，最后计算并更新UI。    磁盘定时器流程图：  首先初始化拷贝全局变量（上一次IO操作时间，也是一个累计值），使用C++流的方式读取diskstats文件，获取到IO操作的时间，并赋值给全局变量，最后计算出磁盘使用率并更新UI。    网络定时器流程图：  首先初始化变量，拷贝全局变量（发送数据字节数和接收数据字节数），通过流的方式读取/proc/net/dev文件，获取到网卡接收和发送数据累计值，赋值给全局变量，然后计算出使用率，最后读取/proc/net/arp文件获取网卡信息并更新UI。   五、代码分析5.1 函数功能  |  |  | | --- | --- | | **函数声明** | **函数功能** | | void Init(); | 绘图初始化、安装定时器和启动定时器 | | int IsPid(const struct dirent \*entry); | 判断目录是否为Pid（数字） | | void DispProc(); | 计算、显示进程页面函数 | | void PaintGraph(QCustomPlot \*customPlot); | 绘制曲线图函数 | | void DispCpuStat(); | 计算、显示CPU性能状况 | | void DispMemStat(); | 计算、显示内存使用情况 | | void DispDiskStat(); | 计算、显示磁盘IO使用情况 | | void DispNetStat(); | 计算、显示网络使用情况 | | void on\_listWidget\_itemClicked(QListWidgetItem \*item); | 槽函数，当某个进程被点击时触发 | | void on\_pushButton\_clicked(); | 槽函数，“结束进程”按钮点击后触发 | | void on\_tabWidget\_currentChanged(int index); | 槽函数，选项卡切换时触发 | | void on\_poweroffButton\_clicked(); | 槽函数，“关机”按钮点击后触发 | | void on\_restartButton\_clicked(); | 槽函数，“重启”按钮点击后触发 |  5.2 函数实现 **void PaintGraph(QCustomPlot \*customPlot);**  使用”qcustomplot.h”头文件里库函数，为传入的customplot参数控件添加曲线图，设置颜色、图列、刻度、填充等。    **void Init();**  从ui中取到曲线图控件作为参数传入PaintGraph中，初始化绘图；通过connect函数，将定时器与对应处理函数绑定，其中进程显示为单独一个定时器，因为进程定时器可能会因为用户操作而暂停，这时性能监控定时器不能暂停；最后启动两个定时器并设置间隔时间为1秒。    **void DispProc();**  使用一个全局Map存储进程Pid和当前CPU使用时间的映射并每次读取后更新Map。  首先定义变量，定义一个Map，存储进程状态简写与进程状态的映射，然后为List设置标题。通过"dirent.h"头文件里的opendir函数打开/proc目录，readdir函数依次读取目录，如果该目录不是一个Pid号则跳过下面代码，否则读取对应Pid目录下的stat文件获取有关CPU的参数；通过indexOf函数索引到括号位置，通过Mid函数截取括号内字符串从而获取到进程名；从全局变量Map中取出上次存储的该进程的CPU使用时间，与这次读取到时间做差，求出在1秒内该进程CPU使用时间，从而计算CPU使用率；最后更新UI时判断进程状态是否为运行状态，如果是则添加到List头。      **void DispCpuStat();**  使用两个全局变量存储当前时刻CPU的总时间和闲置时间。  首先初始化变量，将全局变量里的时间拷贝，打开/proc/cpuinfo文件，读取CPU型号等参数；再打开/proc/stat文件，还是通过indexOf函数索引关键词位置，mid函数分割字符串取出对应参数；计算总时间和闲置时间并重新赋值给全局变量；再计算这一秒CPU使用时间并换算单位为秒，算出使用率；最后更新UI并传递数值给曲线图控件显示曲线。        **void DispMemStat();**  首先初始化变量，打开/proc/meminfo文件，逐行读取，通过indexOf函数索引关键词位置，通过mid函数切割参数并存储；最后计算内存使用率并更新UI显示。      **void DispDiskStat();**  使用全局变量存储当前时刻磁盘IO占用时间。  首先初始化变量并拷贝全局变量，通过C++流的方式打开/proc/diskstats文件，获取到IO花费的毫秒数并重新赋值给全局变量；然后将拷贝的全局变量（上一次的IO花费时间）与当前IO花费时间做差，求出这一秒IO使用时间，从而计算出IO使用率；最后更新UI。    **void DispNetStat();**  使用全局变量存储当前时刻通过网卡发送和接收的字节数。  首先初始化变量并拷贝全局变量，通过流的方式读取/proc/net/dev文件，索引到enp0s3网卡（以太网卡），获取到当前累计的发送字节数和接收字节数。再通过与上一次的值做差，计算出这一秒当前网卡的数据传输大小；再读取/proc/net/arp文件，获取当前网卡型号，最后更新UI。      **事件处理函数**  点击List中某个进程时，关闭进程计时器，不然会每隔一秒更新UI。  点击“结束进程”按钮时，首先获取选中条目，切割出进程PID，再调用system函数发送kill命令来杀死进程。切换选项卡后重新启动定时器。  点击“关机”和“重启”按钮后，使用system函数向shell发送”poweroff”和”restart”命令来关机或重启计算机。   5.3 程序流程分析 主函数中调用Init函数    Init函数中启动定时器1秒执行一次函数。procTimer定时器执行DisProc函数，每秒更新进程页面的进程显示；timer定时器每秒执行DispMemStat(),DispCpuStat(),  DispDiskStat(),DispNetStat()函数，动态更新CPU、内存、磁盘IO和网络性能状态。   六、运行结果分析6.1 编译器与运行环境 编译器版本：Qt 5.10.0  运行环境：Linux Ubuntu 18.0.4 6.2 功能演示 **进程管理功能：**  启动程序首先进入进程管理页面，进程会每隔一秒刷新一次，每个条目显示了进程的PID号、状态、CPU占用率、内存使用情况、优先级和进程名称（命令）。    选中其中一个进程会停止刷新，点击“结束进程”按钮会杀死选中进程；当切换页面时，会重新刷新进程。    **CPU性能监控：**  切换到性能页面，首先显示的是对CPU的性能监控。标题显示当前系统CPU型号，这里为Intel的九代i5，还有主频等一些参数。中间图表显示了最近CPU使用情况的曲线图，可以发现最近一段时间CPU使用很低；下方以文字形式给出了当前CPU利用率、系统运行时间、正在运行进程数、已创建进程数和被挂起进程数。    启动FireFox浏览器，会发现CPU利用率激增，并且正在运行进程数增多，已创建进程数增加；待浏览器启动完后CPU曲线重新变得平缓。    **内存性能监控：**  切换内存监控界面，标题显示内存总量和使用量；中间图表反映了最近一段时间内存使用情况，程序正常运行并且没有启动其他程序时，可以发现内存使用情况基本不变；下面进度条和文字也显示了使用百分比。    这时打开浏览器，发现内存使用上升并且最后平稳下来，进度条也显示内存使用率多了5个百分点，证明浏览器占用了总内存的5%。    **磁盘性能监控：**  磁盘仍用图表显示最近一段时间磁盘IO的使用情况，可以发现仅运行本程序磁盘大部分时间处于待机状态，偶尔会出现一个峰值。    **网络性能监控：**  切换到网络监控界面，图表显示最近一段时间网络使用情况；下方显示发送和结束数据速率，单位为Kbps，即每秒接收多少KB数据；右边显示了当前网卡名称、连接类型和当前IP地址。    同样打开浏览器并访问网页，发现曲线起伏并显示当前接收、发送数据速率。   七、总结 **心得**  1. 通过本次课程设计，深入理解和掌握了Linux操作系统下proc文件系统的原理和使用；  2.了解和掌握了Linux进程、性能管理的底层原理，和查看Linux进程、系统性能的另一种方式；  3.深入理解和探究了操作系统进程管理和性能监控的原理和方法，增强了对操作系统的学习和理解。  4.自主学习了Qt开发框架，熟悉并掌握了使用Qt编写简单的桌面应用程序；  **优点**  1.仿造Windows任务管理器设计，页面较为简洁易懂；  2.使用图表监控性能状态，形象直观；  3.将进程显示进行人性化处理，相比于Linux下top命令更加直观、简单；  **缺点**  1.总体方面，代码比较简单，很多细节没有考虑完全；  2.进程方面，界面显示不够美观，没有像Windows一样区分类别和显示总使用量；  3.内存监控方面，显示信息略微有点单调；  4.磁盘监控方面，只显示了磁盘使用情况的图表，因为其他信息从/proc中不好获取；  5.网络监控方面，由于代码逻辑的原因，网卡配置仅能获取一块网卡信息，缺乏通用性。 八、参考资料 [1] [ljheee](https://ljheee.blog.csdn.net/). 《自己实现Linux系统任务管理器》，CSDN博客，2017.  [2] 烨子. 《简析Linux中 /proc/[pid] 目录的各文件》，linuxprobe，2017.  [3] lgstudyvc. 《linux /proc/cpuinfo文件分析》，CSDN博客，2012.  [4] coldice0521. 《/proc/meminfo参数详细解释》，CSDN博客，2020.  [5] 哪是什么大佬. 《linux下/proc/diskstats文件详解》，博客园，2018.  [6] weichanghu\_. 《Linux /proc/net/ 下文件用途》，CSDN博客，2018. |
| 附录：源代码 **mainwindow.h**  #ifndef MAINWINDOW\_H  #define MAINWINDOW\_H  #include <QMainWindow>  #include "qcustomplot.h"  #include <QTimer>  namespace Ui {  class MainWindow;  }  class MainWindow : public QMainWindow  {  Q\_OBJECT  public:  int key =0 ;  explicit MainWindow(QWidget \*parent = 0);  ~MainWindow();  private:  int curTime,curTotal,curIdle,curTranTime,curRecvByte,curSendByte;  QMap<int,int> procPreTimeMap;  QTimer timer,procTimer;  Ui::MainWindow \*ui;  void Init();  int IsPid(const struct dirent \*entry);  void PaintGraph(QCustomPlot \*customPlot);  private slots:  void DispProc();  void DispCpuStat();  void DispMemStat();  void DispDiskStat();  void DispNetStat();  void on\_listWidget\_itemClicked(QListWidgetItem \*item);  void on\_pushButton\_clicked();  void on\_tabWidget\_currentChanged(int index);  void on\_poweroffButton\_clicked();  void on\_restartButton\_clicked();  };  #endif // MAINWINDOW\_H  **mainwindow.cpp**  #include "mainwindow.h"  #include "ui\_mainwindow.h"  #include "dirent.h"  #include "QFileDialog"  #include "iostream"  #include "fstream"  #include "QVector"  #include "QTimer"  #include "unistd.h"  using namespace std;  MainWindow::MainWindow(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  ui->setupUi(this);  Init();  }  MainWindow::~MainWindow()  {  delete ui,timer;  }  void MainWindow::Init(){  //绘图  PaintGraph(ui->memCustomPlot);  PaintGraph(ui->cpuCustomPlot);  PaintGraph(ui->diskCustomPlot);  PaintGraph(ui->netCustomPlot);  //安装定时器  connect(&procTimer,SIGNAL(timeout()),this,SLOT(DispProc()));  connect(&timer,SIGNAL(timeout()),this,SLOT(DispMemStat()));//mem  connect(&timer,SIGNAL(timeout()),this,SLOT(DispCpuStat()));//cpu  connect(&timer,SIGNAL(timeout()),this,SLOT(DispDiskStat()));//disk  connect(&timer,SIGNAL(timeout()),this,SLOT(DispNetStat()));//net  //启动定时器  procTimer.start(1000);  timer.start(1000);  }  //传入一个目录结构体，取出目录名查看是否为数字  int MainWindow::IsPid(const struct dirent \*entry){  const char \*p;  for(p=entry->d\_name;\*p;p++){ //d\_name是一个字符数组  if(!isdigit(\*p))  return 0;  }  return 1;  }  void MainWindow::DispProc(){  DIR \*procDir;  QFile file;  QString str;  int cRow,par1,par2,procTime,preProcTime;  string comm;  QString qComm;  double usage;  char task\_stat;  int pid,ppid,pgid,sid,tty\_nr,tty\_pgrp,task\_flag,min\_flt,cmin\_flt,maj\_flt,cmaj\_flt,utime,stime,  cutime,cstime,priority,nice,num\_threads,it\_real\_value,start\_time,vsize,rss;  int curProcTime;  struct dirent \*entry;  char path[5+256+5]; // /proc + d\_name + /stat  QMap<char,QString> statMap;  //init map  statMap.insert('R',"running");  statMap.insert('S',"sleeping");  statMap.insert('D',"disk sleep");  statMap.insert('T',"stopped");  statMap.insert('Z',"zombie");  statMap.insert('X',"dead");  statMap.insert('I',"idle");  ui->listWidget->clear();  QListWidgetItem \*title = new QListWidgetItem("PID\t"+QString::fromUtf8("状态")+'\t'+QString::fromUtf8("CPU")  +'\t'+QString::fromUtf8("内存")+'\t'+QString::fromUtf8("优先级")  +'\t'+QString::fromUtf8("名称"));  QFont font;  font.setBold(true);  title->setFont(font);  ui->listWidget->addItem(title);  //打开/proc目录  procDir = opendir("/proc");  while(entry=readdir(procDir)){  if(!IsPid(entry))  continue;  snprintf(path,sizeof(path),"/proc/%s/stat",entry->d\_name);  //打开对应pid目录的stat文件  ifstream fin(path,ios::in);  fin >> pid;  preProcTime = procPreTimeMap.value(pid);//先从Map中取出上一次cpu时间  fin>>comm>>task\_stat>>ppid>>pgid>>sid>>tty\_nr>>tty\_pgrp>>task\_flag>>min\_flt>>cmin\_flt>>maj\_flt>>cmaj\_flt>>utime>>stime>>  cutime>>cstime>>priority>>nice>>num\_threads>>it\_real\_value>>start\_time>>vsize>>rss;  //删除括号  qComm = QString::fromStdString(comm);  par1 = qComm.indexOf('(');  par2 = qComm.indexOf(')');  qComm = qComm.mid(par1+1,par2-par1-1);  qComm.trimmed();  //计算cpu使用率  //将当前进程pid和time存入Map中，下次执行根据pid找到上次时间  curProcTime = utime + stime + cutime + cstime;  procPreTimeMap.insert(pid,curProcTime);  procTime = curProcTime - preProcTime;//当前进程使用cpu总时间  usage = procTime\*1.0 / curTime \* 100; //进程使用时间/cpu总时间  QListWidgetItem \*item = new QListWidgetItem(QString::number(pid)+"\t"+statMap.value(task\_stat)  +'\t'+QString::number(usage,'f',1)+'%'+'\t'  +QString::number(rss\*4/1024)+'M'+'\t'  +QString::number(priority)+"\t"+qComm);  if(pid % 2){  item->setBackgroundColor(QColor(248,248,255));  }  //将正在运行的进程放在前面  if(task\_stat == 'R'){  ui->listWidget->insertItem(1,item);  }else{  ui->listWidget->addItem(item);  }  fin.close();  }  }  void MainWindow::PaintGraph(QCustomPlot \*customPlot){  customPlot->addGraph();  customPlot->graph(0)->setPen(QPen(QColor(0,0,255))); //曲线颜色  customPlot->legend->setVisible(false); //不显示图例  customPlot->axisRect()->setupFullAxesBox();//四边轴显示  customPlot->xAxis->setTickLabels(false);  customPlot->graph(0)->setBrush(QBrush(QColor(0,0,255,50)));//填充  customPlot->yAxis->setRange(0,100);  }  void MainWindow::DispCpuStat(){  string tmp;  double usage;  int preTotal, user, nice, system, preIdle,iowait, irq, softirq, stealstolen, guest, guest\_nice;  QString curStr,cpuInfo,processes,procRun,procBlock;  QFile file;  int pos,runTime;  //将上一次的CPU时间拷贝  preTotal = curTotal;  preIdle = curIdle;  //cpu信息  file.setFileName("/proc/cpuinfo");  if(!file.open(QIODevice::ReadOnly)){  cout << "cpuinfo can not open!" << endl;  return;  }  while(1){  curStr = file.readLine();  pos = curStr.indexOf("model name");  if(pos != -1){  cpuInfo = curStr.mid(pos+12,40);  break;  }  }  file.close();  file.setFileName("/proc/stat");  if(!file.open(QIODevice::ReadOnly)){  cout << "stat can not open!" << endl;  return;  }  while(1){  curStr = file.readLine();  pos = curStr.indexOf("processes");  if(pos != -1){  processes = curStr.mid(pos+10,curStr.length()-10);  processes.trimmed();  }  else if(pos = curStr.indexOf("procs\_running"),pos != -1){  procRun = curStr.mid(pos+13,5);  procRun.trimmed();  }  else if(pos = curStr.indexOf("procs\_blocked"),pos != -1){  procBlock = curStr.mid(pos+14,curStr.length()-14);  procBlock.trimmed();  break;  }  }  file.close();  //计算CPU利用率  ifstream fin("/proc/stat", ios::in);  fin >> tmp >> user >> nice >> system >> curIdle >> iowait >> irq >> softirq >> stealstolen >> guest >> guest\_nice;  fin.close();  //重新赋值给全局变量  curTotal = user + nice + system + curIdle + iowait + irq + softirq + stealstolen + guest + guest\_nice;  curTime = curTotal - preTotal;  usage = (curTotal + preIdle - preTotal - curIdle) \* 1.0 / curTime \*100;  runTime = (user + nice +system +curIdle + iowait + irq + softirq)/100;  //second to std  int H = runTime / (60\*60);  int M = (runTime- (H \* 60 \* 60)) / 60;  int S = (runTime - (H \* 60 \* 60)) - M \* 60;  QString hour = QString::number(H);  if (hour.length() == 1) hour = "0" + hour;  QString min = QString::number(M);  if (min.length() == 1) min = "0" + min;  QString sec = QString::number(S);  if (sec.length() == 1) sec = "0" + sec;  QString qTime = hour + ":" + min + ":" + sec;  //UI  ui->cpuName->setText(cpuInfo);  ui->cpuUse->setText(QString::number(usage,'f',1)+'%');  ui->cpuProcRun->setText(procRun);  ui->cpuProcCreate->setText(processes);  ui->cpuProcBlock->setText(procBlock);  ui->cpuTime->setText(qTime);  ui->cpuCustomPlot->graph(0)->addData(key,usage);  ui->cpuCustomPlot->graph(0)->removeDataBefore(key-20);//显示20s内  ui->cpuCustomPlot->xAxis->setRange(key,20,Qt::AlignRight);  ui->cpuCustomPlot->replot();  }  void MainWindow::DispMemStat(){  QString curStr; //当前读取字符串  QFile file;  int pos;  QString memTotal,memFree;  float nMemTotal,nMemFree,nMemUsed;  float value;  file.setFileName("/proc/meminfo");  if(!file.open(QIODevice::ReadOnly)){  cout << "meminfo can not open!" << endl;  return;  }  while(1){  curStr = file.readLine();  pos = curStr.indexOf("MemTotal");  if(pos != -1){  memTotal = curStr.mid(pos+10,curStr.length()-13); //mid(start pos,num)  memTotal = memTotal.trimmed(); //清除空格  nMemTotal = memTotal.toFloat() / 1024 / 1024; //M  }  else if(pos = curStr.indexOf("MemFree") , pos != -1){  memFree = curStr.mid(pos+9,curStr.length()-12);  memFree = memFree.trimmed();  nMemFree = memFree.toFloat() / 1024 /1024;  break;  }  }  nMemUsed = nMemTotal - nMemFree;  value = nMemUsed\*100 / nMemTotal;  //update UI  key++;  ui->memCustomPlot->graph(0)->addData(key,value);  ui->memCustomPlot->graph(0)->removeDataBefore(key-20);//显示20s内  ui->memCustomPlot->xAxis->setRange(key,20,Qt::AlignRight);  ui->memCustomPlot->replot();  ui->memProgressBar->setValue((int)value);  ui->memProgressBar->setMaximum(100);  ui->memAll->setText(QString::number(nMemTotal,'f',2)+'G');  ui->memAvailable->setText(QString::number(nMemFree,'f',2)+'G');  ui->memUsing->setText(QString::number(nMemUsed,'f',2)+'G');  ui->memUse->setText(QString::number(nMemUsed,'f',2)+'G');  }  //loop:挂载镜像 sr0光驱 sda:磁盘 sda5:第一逻辑分区  void MainWindow::DispDiskStat(){  string temp;  int preTranTime;  float usage;  preTranTime = curTranTime;  //设备号 编号 设备 读完成次数 合并完成次数 读扇区次数 读操作花费毫秒数 写完成次数 合并写完成次数 写扇区次数  //写操作花费的毫秒数 正在处理的输入/输出请求数 输入/输出操作花费的毫秒数 输入/输出操作花费的加权毫秒数。  ifstream fin("/proc/diskstats", ios::in);  while(!fin.eof()){  fin >> temp >> temp >> temp;  if(temp.size()==3 && temp == "sda"){  fin >> temp >> temp >> temp >> temp >> temp >> temp >> temp  >> temp >> temp >> curTranTime;  getline(fin,temp);//将当前字符串赋值给temp  }else{  getline(fin,temp);  }  }  fin.close();  usage = (curTranTime - preTranTime) \*1.0 / 1 / 1000 \* 100;  //UI  ui->diskCustomPlot->graph(0)->addData(key,usage);  ui->diskCustomPlot->graph(0)->removeDataBefore(key-20);//显示20s内  ui->diskCustomPlot->xAxis->setRange(key,20,Qt::AlignRight);  ui->diskCustomPlot->replot();  }  void MainWindow::DispNetStat(){  string temp;  int preRecvByte,preSendByte;  float recvRate,sendRate,throughput;  preRecvByte = curRecvByte;//曾经的当前就是以前  preSendByte = curSendByte;  ifstream fin("/proc/net/dev", ios::in);  while(1){  fin >> temp;  if(temp == "enp0s3:"){  fin >> curRecvByte >> temp >> temp >> temp >> temp >> temp >> temp >> temp >> curSendByte;  break;  }else{  getline(fin,temp);  }  }  fin.close();  recvRate = (curRecvByte - preRecvByte)\*1.0 / 1024 \* 8; // kbps  sendRate = (curSendByte - preSendByte)\*1.0 / 1024 \* 8;  throughput = recvRate + sendRate;  string ipAddr,device,type;  fin.open("/proc/net/arp",ios::in);  getline(fin,temp);  fin >> ipAddr >> type >> temp >> temp >> temp >> device;  fin.close();  if(type == "0x1"){  type = "Etherney";  }else if(type == "0x17"){  type = "Metricom starmode IP";  }  else{  type = "Frame Relay DLCI";  }  //UI  ui->netRecv->setText(QString::number(recvRate,'f',1)+"Kbps");  ui->netSend->setText(QString::number(sendRate,'f',1)+"Kbps");  ui->netDev->setText(QString::fromStdString(device));  ui->netIP->setText(QString::fromStdString(ipAddr));  ui->netType->setText(QString::fromStdString(type));  ui->netCustomPlot->graph(0)->addData(key,throughput);  ui->netCustomPlot->graph(0)->removeDataBefore(key-20);//显示20s内  ui->netCustomPlot->xAxis->setRange(key,20,Qt::AlignRight);  ui->netCustomPlot->yAxis->setRange(0,500);  ui->netCustomPlot->replot();  }  //事件处理  void MainWindow::on\_listWidget\_itemClicked(QListWidgetItem \*item){  procTimer.stop();  }  void MainWindow::on\_pushButton\_clicked(){  QListWidgetItem \*item = ui->listWidget->currentItem();  QString proMsg = item->text();  proMsg = proMsg.section("\t",0,0);//切割出PID  system("kill " + proMsg.toLatin1());  QMessageBox::warning(this,tr("kill"),QString::fromUtf8("该进程已被结束！"),QMessageBox::Yes);  procTimer.start(1000);  }  void MainWindow::on\_tabWidget\_currentChanged(int index){  procTimer.start(1000);  }  void MainWindow::on\_poweroffButton\_clicked(){  system("poweroff");  }  void MainWindow::on\_restartButton\_clicked(){  system("restart");  } |