|  |
| --- |
| **操作系统课程设计报告评定：** |
|  |

**目录**

[1 设计原理 1](#_Toc26477)

[1.1 概述 1](#_Toc13025)

[1.2 调度功能 1](#_Toc8542)

[1.3 调度算法 1](#_Toc13260)

[2 需求分析 2](#_Toc12677)

[3 总体设计 2](#_Toc27967)

[3.1 设计思路 3](#_Toc24471)

[3.2 设计方案 3](#_Toc2925)

[4 算法设计分析 4](#_Toc25857)

[4.1 调度算法比较 4](#_Toc20462)

[4.2 调度算法基本流程 6](#_Toc15192)

[4.3 调度算法演示 8](#_Toc31484)

[4.4 调度算法评价指标 9](#_Toc25365)

[5 程序总流程图 9](#_Toc28585)

[6 源代码分析 10](#_Toc3459)

[6.1 相关头文件简介 10](#_Toc29354)

[6.2 函数功能简介及重要函数的实现 12](#_Toc10948)

[7 运行结果分析 14](#_Toc8371)

[7.1 编译器和运行环境 14](#_Toc9563)

[7.2 功能演示 14](#_Toc24695)

[7.5 分析总结 21](#_Toc7833)

[8 参考资料 22](#_Toc30143)

[附录：源代码 23](#_Toc23518)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 通用处理机调度演示程序 设计原理1.1 概述 在多道程序环境下，主存中有着多个进程，其数目往往多于处理机数目。这就要求系统能按某种算法，动态地把处理机分配给就绪队列中的一个进程，使之执行。分配处理机的任务是由处理机调度程序完成的。由于处理机是最重要的计算机资源，提高处理机的利用率以及改善系统性能（吞吐量、响应时间），在很大程度上取决于处理机调度性能的好坏。处理机调度就是从就绪队列中，按照一定的算法选择一个进程并将处理机分配给它运行，以实现进程并发地执行。 1.2 调度功能 一般情况下，当占用处理机的进程因为某种请求得不到满足而不得不放弃CPU进入等待状态时，或者当时间片到，系统不得不将CPU分配给就绪队列中另一进程的时候，都要引起处理机调度。除此之外，进程正常结束、中断处理等也可能引起处理机的调度。因此，处理机调度是操作系统核心的重要组成部分，它的主要功能如下：   * 记住进程的状态，如进程名称、指令计数器、程序状态寄存器以及所有通用寄存器等现场信息，将这些信息记录在相应的进程控制块中。 * 根据一定的算法，决定哪个进程能获得处理机，以及占用多长时间。 * 收回处理机，即正在执行的进程因为时间片用完或因为某种原因不能再执行的时候，保存该进程的现场，并收回处理机。  1.3 调度算法 在多道程序系统中，一个作业被提交后必须经过处理机调度后，方能获得处理机执行。对于批量型作业而言，通常需要经历作业调度和进程调度两个过程后方能获得处理机执行。作业调度是对成批进入系统的用户作业，根据作业控制块的信息，按一定的策略选取若干个作业使得它们可以去获得处理器运行的一项工作。而对每个用户来说总希望自己的作业的周转时间是最小的，因此可以采用：先来先服务算法、时间片轮转算法、短作业优先算法、静态优先权调度算法、高响应比调度算法等五种算法降低周转时间。   * 先来先服务算法(FCFS)：每次调度是从就绪队列中选择一个最先进入该队列的进程为之分配处理机，使之投入运行。该进程一直运行到完成或发生某事件而阻塞后才放弃处理机。 * 时间片轮转算法(RR)：系统将所有的就绪进程按先来先服务的原则排成一个队列，每次调度时，把CPU 分配给队首进程，并令其执行一个时间片。当执行的时间片用完时，由一个计时器发出时钟中断请求，调度程序便据此信号来停止该进程的执行，并将它送往就绪队列的末尾，然后，再把处理机分配给就绪队列中新的队首进程，同时也让它执行一个时间片。 * 短作业优先算法(SJF)：从后备队列中选择一个或若干个估计运行时间最短的进程，将它们调入内存运行。 * 静态优先权调度算法(PF):把处理机分配给就绪队列中优先权最高的进程。 * 高响应比调度算法(HRRN):把处理机分配给就绪队列中响应比最高的进程。  需求分析  * 编译及运行环境：Qt，Linux * 前提条件：假定初始时CPU空闲无进程运行。 * 界面设计：在主界面中能通过下拉菜单选择某算法，有开始按钮（控制算法执行）、清空按钮（清空主界面进程信息）、添加按钮（逐行添加进程信息）、读取文件按钮（从文件中读入进程信息），设有计时器显示运行时间，在主界面中通过表格显示各个进程的信息，可以显示同一组数据运用各算法的平均周转时间，方便用户使用和阅读。 * 具体功能：编程实现先来先服务算法、时间片轮转算法、短作业优先算法、静态优先权调度算法、高响应比调度算法等五种算法的具体过程，在主界面的表格中动态显示出进程运用各个算法时状态的变化，并计算各算法的平均周转时间，能进行相关比较。 * 进程产生方式：可以手动添加进程信息（进程号、进入时间、服务时间、优先级），也可以从文件中读取。  总体设计3.1 设计思路 根据功能要求，主界面需要如下几个板块：显示进程信息模块（进程号、进入时间、服务时间、优先级、响应时间、结束时间、状态）、显示调度算法模块（先来先服务算法、时间片轮转算法、短作业优先算法、静态优先权调度算法、高响应比调度算法）、显示运行时间模块（计时器）、显示各个算法的平均周转时间模块和按钮模块（开始、清空、添加、读取文件）。大概思路如图所示：  wps  图3.1 1设计思路 3.2 设计方案3.2.1 界面功能模块设计  * 显示进程信息模块：采用表格的形式显示，表头包含七类信息分别为进程号、进入时间、服务时间、优先级、响应时间、结束时间和状态，每行显示一个进程的具体信息。 * 显示算法模块：为了方便用户选择，采用下拉菜单的形式，默认显示先来先服务算法。 * 显示运行时间模块：采用一个计时器显示运行时间在界面右上角，初始时间为0。 * 按钮模块：①开始按钮：当无进程信息时点击开始按钮弹出警示框提醒用户添加数据，避免程序崩溃 ②清空按钮：可以一键清空表格中的所有进程信息 ③添加按钮：用于用户手动添加进程信息，点击后将在表格中添加一行进程号、进入时间、服务时间、优先级均为零的默认进程信息 ④读取文件按钮：用于用户从文件中读取进程信息，点击后将弹出文件选择框选择文件后将在表格中自动添加相应信息。 * 显示平均周转时间模块：在界面下方有五个显示框，每当执行完一个算法后将在相应算法的显示框内显示该组数据的平均周转时间。  3.2.2 设计图示 下图展示了五个界面模块的大致布局，在后续测试中将对界面进行调整修改。    图 3.2.2 1 设计图 算法设计分析4.1 调度算法比较 表 4.1 1 五种调度算法的比较   |  |  |  |  | | --- | --- | --- | --- | | 算法名称 | 概念 | 优点 | 缺点 | | 先来先服务 | 按照进程进入系统的先后次序进行调度，先进入系统者先调度 | 有利于长进程、CPU繁忙型进程，适用于批处理系统 | 不利于短进程、不利于I/O繁忙型作业，不适用于分时系统 | | 时间片轮转 | 系统将所有的就绪进程按先来先服务的原则排成一个队列，每次调度时，把CPU分配给队首进程，并令其执行一个时间片；当执行的时间片用完时，由一个计时器发出时钟中断请求，调度程序便停止该进程的执行，并将其放就绪队列尾；然后，再把处理机分配给就绪队列中新的队首 | 让每个进程在就绪队列中等待的时间和享受服务的时间成正比例 | 紧迫任务响应慢 | | 短作业优先（抢占式） | 从队列中选出一个估计运行时间最短的进程优先调度 | 缩短平均周转时间 | 长作业可能长时间得不到执行 | | 静态优先权调度（抢占式） | 把处理机分配给优先权最高的进程，但在执行期间，只要出现另一个优先权更高的进程，则进程调度程序就立即停止当前进程的执行，并将处理机分配给新到的优先权最高的进程 | 能更好地满足紧迫进程的要求 | 优先权低的进程可能长期得不到执行 | | 高响应比调度 | 把处理机分配给就绪队列中响应比最高的进程 | 既照顾了短作业，又考虑了作业到达的先后次序，又不会使长作业长期得不到服务 | 要进行响应比计算，增加了系统开销 |  4.2 调度算法基本流程 先来先服务：  wps  图4.2 1 先来先服务调度算法流程图  时间片轮转：  wps  图4.2 2 时间片轮转调度算法流程图  短作业优先（抢占式）：  wps  图4.2 3 短作业优先调度算法（抢占式）流程图  静态优先权调度（抢占式）：  wps  图4.2 4 静态优先权调度算法（抢占式）流程图  高响应比调度：  wps  图4.2 5 高响应比调度算法流程图 4.3 调度算法演示 假设有三个进程P1、P2、P3，默认优先级数值越小等级越高   |  |  |  |  | | --- | --- | --- | --- | | 进程 | 到达时间 | 服务时间 | 优先级 | | P1 | 0 | 2 | 1 | | P2 | 1 | 3 | 3 | | P3 | 3 | 1 | 2 |             图 4.3 1 五种调度算法的演示  注：相应比=（等待时间+要求服务时间）/要求服务时间  在0时刻时只有P1到达因此先执行P1，P1执行完后只有P2到达，因此直接执行P2，接着执行P3，在此实例中高响应比调度相当于先来先服务调度；若P1执行结束后P2、P3都到达则计算两者响应比，响应比高的进程先执行。 4.4 调度算法评价指标  * CPU利用率：CPU“忙碌”的时间占总时间的比例，利用率=忙碌的时间/总时间。 * 系统吞吐量：单位时间内完成作业的数量。 * 周转时间：从作业被提交给系统开始，到作业完成为止的这段时间间隔。 * 等待时间：进程建立后等待被服务的时间之和。 * 响应时间：从用户提交请求到首次产生响应所用的时间。  程序总流程图 wps  图 5 总流程图 源代码分析 因代码量比较大，**具体代码放在附录部分**，此处进行代码的相关分析。 6.1 相关头文件简介6.1.1 pcb.h  * PCB的定义：   class PCB  {  public:  PCB();  int pid; //进程号  int arriveTime;//到达时间  int serviceTime;//服务时间  int priority;//优先级  int startTime;//开始运行时间  int finishTime;//完成运行时间  int waitTime;//等待时间  float responseRatio;//响应比  float turnaroundTime;//周转时间  int state[20];//状态 1:就绪 2:执行 3:结束  }; 6.1.2 mainwindow.h  * 全局变量声明   QTimer \*timer;//定义定时器  int second;//秒数  Ui::MainWindow \*ui;  PCB pcbArr[20];//存放进程信息  PCB temp[20];//存放按到达时间先后排序的进程  float fcfsAverageTurnaroundTime;//先来先服务的平均周转时间  float sjfAverageTurnaroundTime;//短作业优先的平均周转时间  float rrAverageTurnaroundTime;//时间片轮转的平均周转时间  float pfAverageTurnaroundTime;//静态优先级调度的平均周转时间  float hrrnAverageTurnaroundTime;//高响应比优先的平均周转时间 6.2 函数功能简介及重要函数的实现 表6.2 mainwindow函数说明   |  |  | | --- | --- | | 函数声明 | 作用与实现 | | void init(); | 用于界面样式初始化 | | void initPcb(); | 将表格中填入的数据存放到数组pcbArr中 | | void clearPcb(int rows); | 清除pcbArr数组中存放的数据 | | void sort(); | 将所有进程按照到达时间的先后顺序存入temp数组中 | | void copyPcb(); | 将通过算法计算后更新的PCB内容从temp数组复制到pcbArr数组中 | | void fcfs(); | 执行先来先服务算法，将temp数组中的进程依次计算响应时间和结束时间 | | void sjf(); | 执行短作业优先抢占式算法，定义变量current作为当前时间，以每秒为单位，循环遍历temp数组，找出当前时间点已到达且服务时间最短的进程执行，若出现服务时间更短的进程则抢占，并在temp.state数组中标记被抢占的进程此刻状态为等待 | | void rr(); | 执行时间片轮转算法，按照进程轮转顺序为进程分配指定的时间片，若在时间片内可以执行完当前进程则执行完后即切换，若在时间片内不能执行完成则被抢占，并在temp.state数组中标记被抢占的进程此刻状态为等待 | | void pf(); | 执行静态优先级抢占式算法，默认优先级数值越小等级越高，定义变量current作为当前时间，以每秒为单位，循环遍历temp数组，找出当前时间点已到达且优先级最高的进程执行，若出现服务时间更短的进程则抢占，并在temp.state数组中标记被抢占的进程此刻状态为等待 | | void hrrn(); | 执行高响应比优先算法，先执行第一个到达的进程，若第一个进程执行完后只有一个进程到达则相当于先来先服务算法，若第一个进程执行结束后有多个进程到达，则分别计算多个进程的响应比，选择当前时刻响应比最高的进程先执行完毕 | | void on\_addButton\_clicked(); | 设置添加按钮的点击事件，点击一次表格增加一行 | | void on\_startButton\_clicked(); | 设置开始按钮的点击事件，当表格中无数据时点击按钮弹出警示框提示添加数据，当表格中有数据时则执行选择的相应算法执行，并开启定时器 | | void updateTime(); | 与定时器绑定，每一秒执行一次该函数，动态更改每一个进程的状态 | | void on\_clearButton\_clicked(); | 设置清空按钮的点击事件，清空表格中的所有数据，和平均周转时间显示框的所有数据，并将右上角显示时间清零 | | void on\_comboBox\_currentIndexChanged(int index); | 设置下拉菜单的点击事件，当切换选项时清除上一次执行的结果为下一次执行做准备，当切换到时间片轮转算法时显示时间片输入框 | | void on\_fileButton\_clicked(); | 设置读取文件按钮的点击事件，弹出文件选择器，将选中文件中的数据填入到表格当中 |  运行结果分析7.1 编译器和运行环境  * 编译器版本：Qt 12.0.0 * 运行环境：Linux Ubuntu 20.0.4  7.2 功能演示 1) 初始主界面：    图7.4 1 初始主界面   1. 手动添加数据：   点击添加按钮逐行添加    图 7.4 2 点击添加按钮效果  双击表格单元格可以修改数据    图 7.4 3 双击单元格效果   1. 读取数据：   test.txt文件内容    图 7.4 4文件内容显示  点击读取文件按钮选择文件    图 7.4 5 点击读取文件按钮效果  双击文件将数据导入表格    图7.4 6 双击文件效果   1. 算法选择     图 7.4 7 算法下拉菜单   1. 执行算法   当表格中无数据时点击开始按钮，显示警示框提醒添加数据    图 7.4 8 无数据时点击开始按钮效果  执行先来先服务算法，执行结束后在下方相应位置显示平均周转时间    图 7.4 9 先来先服务算法结果  执行时间片轮转算法，设置时间片=2s，执行结束后在下方相应位置显示平均周转  时间    图 7.4 10 时间片轮转算法结果  执行短作业优先算法，执行结束后在下方相应位置显示平均周转时间    图 7.4 11 先来先服务算法结果  执行静态优先级调度算法，执行结束后在下方相应位置显示平均周转时间    图 7.4 12 静态优先级调度算法结果  执行高响应比优先算法，执行结束后在下方相应位置显示平均周转时间    图 7.4 13 高响应比调度算法结果  该组数据经过不同算法执行的结果显示，短作业优先调度算法的平均周转时间最  短，其它四种调度算法的平均周转时间相同。   1. 动态更新   每秒更新当前进程的状态（就绪、等待、执行、结束）    图 7.4 14 执行中效果   1. 清空数据   将界面恢复成初始状态    图 7.4 15 点击清空按钮效果 7.5 分析总结  * 问题解决  1. 当同一组数据多次执行算法时需要重新开启定时器，本意是想可以动态显示每秒的结果，但是多次测试发现只有第一次的时候是每秒改变，后几次变成了多秒变动，经过调试发现是多次重新开启定时器但未及时销毁之前的定时器的结果，当在每次算法执行结束后销毁定时器问题解决。 2. 在本程序中，时间片轮转算法、静态优先级调度算法和短作业优先算法均为抢占式算法，为了能够更加直观地显示抢占过程，想在状态栏中显示被抢占的进程状态为等待，但是程序核心是根据计算好的结果再在界面上动态显示，存放被抢占和重新执行的时间点比较困难。最终采取的解决方法是在PCB类中增加一个state数组，下标为秒数，存放被抢占时刻和重新执行时刻的状态。  * 优点  1. 较为直观地展现了通用处理机调度五种算法的运用过程。 2. 可以动态显示实时结果，体现进程的抢占过程。 3. 可以比较五种算法的平均周转时间。 4. 界面结构清晰简洁，方便用户使用，支持手动添加数据和文件添加数据。  * 缺点  1. 由于对于Qt编译器的控件还不够熟悉，界面设计比较简单不够美观。 2. 本程序仅支持以秒为单位的时间设置。 3. 平均周转时间不能通过已排序的形式显示仅能人工比较。  参考资料  1. Qt学习笔记 TableWidget使用说明和增删改操作的实现   https://blog.csdn.net/weixin\_34130389/article/details/90087146   1. 操作系统进程调度模拟算法实现（C语言版）   https://www.it610.com/article/1385240534318141440.htm |
| 附录：源代码  * 头文件  1. pcb.h   #ifndef PCB\_H  #define PCB\_H  class PCB  {  public:  PCB();  int pid; //进程号  int arriveTime;//到达时间  int serviceTime;//服务时间  int priority;//优先级  int startTime;//开始运行时间  int finishTime;//完成运行时间  int waitTime;//等待时间  float responseRatio;//响应比  float turnaroundTime;//周转时间  int state[20];//状态 1:就绪 2:执行 3:结束  };  #endif // PCB\_H   1. mainwindow.h   #ifndef MAINWINDOW\_H  #define MAINWINDOW\_H  #include <QMainWindow>  #include <pcb.h>  #include "QTimer"  namespace Ui {  class MainWindow;  }  class MainWindow : public QMainWindow  {  Q\_OBJECT  public:  explicit MainWindow(QWidget \*parent = 0);  ~MainWindow();  private slots:  void on\_addButton\_clicked();  void on\_startButton\_clicked();  void updateTime();  void on\_clearButton\_clicked();  void on\_comboBox\_currentIndexChanged(int index);  void on\_fileButton\_clicked();  private:  QTimer \*timer;  int second;  Ui::MainWindow \*ui;  PCB pcbArr[20];  PCB temp[20];  float fcfsAverageTurnaroundTime;  float sjfAverageTurnaroundTime;  float rrAverageTurnaroundTime;  float pfAverageTurnaroundTime;  float hrrnAverageTurnaroundTime;  void init();  void initPcb();  void clearPcb(int rows);  void sort();  void copyPcb();  void fcfs();  void sjf();  void rr();  void pf();  void hrrn();  };  #endif // MAINWINDOW\_H   * 源文件  1. main.cpp   #include "mainwindow.h"  #include <QApplication>  int main(int argc, char \*argv[])  {  QApplication a(argc, argv);  MainWindow w;  w.show();  return a.exec();  }   1. pcb.cpp   #include "pcb.h"  PCB::PCB()  {  }   1. mainwindow.cpp   #include "mainwindow.h"  #include "ui\_mainwindow.h"  #include "QTimer"  #include <QFileDialog>  #include <QTextStream>  #include <typeinfo>  #include <iostream>  #include <QDebug>  #include <QMessageBox>  using namespace std;  MainWindow::MainWindow(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  ui->setupUi(this);  init();  }  MainWindow::~MainWindow()  {  delete ui;  }  void MainWindow::init()  {  //下拉框  ui->comboBox->addItem("先来先服务");  ui->comboBox->addItem("时间片轮转");  ui->comboBox->addItem("短作业优先");  ui->comboBox->addItem("静态优先权优先调度");  ui->comboBox->addItem("高响应比调度");  //表格  ui->tableWidget->setColumnCount(7);  ui->tableWidget->setHorizontalHeaderLabels(QStringList()<<"进程号"<<"进入时间"<<"服务时间"<<"优先级"<<"响应时间"<<"结束时间"<<"状态");  ui->tableWidget->verticalHeader()->setVisible(false);  ui->label->setVisible(false);  ui->timeEdit->setVisible(false);  // 测试数据  // ui->tableWidget->setRowCount(2);  // QStringList pidList;  // pidList<<"124"<<"534"<<"267";  // QStringList enterTime;  // enterTime<<"0"<<"2"<<"3";  // QStringList serveTime;  // serveTime<<"3"<<"5"<<"1";  // QStringList priority;  // priority<<"3"<<"2"<<"1";  // for(int i=0;i<2;i++){  // int col = 0;  // ui->tableWidget->setItem(i,col++,new QTableWidgetItem(pidList[i]));  // ui->tableWidget->setItem(i,col++,new QTableWidgetItem(enterTime[i]));  // ui->tableWidget->setItem(i,col++,new QTableWidgetItem(serveTime[i]));  // ui->tableWidget->setItem(i,col++,new QTableWidgetItem(priority[i]));  // ui->tableWidget->setItem(i,6,new QTableWidgetItem("就绪"));  // }  // //让tableWidget内容中的每个元素居中  // for (int i=0;i<2;i++){  // for (int j=0;j<4;j++){  // ui->tableWidget->item(i,j)->setTextAlignment(Qt::AlignHCenter|Qt::AlignVCenter);  // }  // }  }  //初始化PCB  void MainWindow::initPcb()  {  int rows = ui->tableWidget->rowCount();  for(int i = 0;i<rows;i++)  {  pcbArr[i].pid = (ui->tableWidget->item(i,0)->text()).toInt();  pcbArr[i].arriveTime = (ui->tableWidget->item(i,1)->text()).toInt();  pcbArr[i].serviceTime = (ui->tableWidget->item(i,2)->text()).toInt();  pcbArr[i].priority = (ui->tableWidget->item(i,3)->text()).toInt();  memset(pcbArr[i].state,0,sizeof(pcbArr[i].state));  }  }  //清除PCB  void MainWindow::clearPcb(int rows)  {  for(int i = 0;i<rows;i++)  {  pcbArr[i].pid = -1;  pcbArr[i].arriveTime = -1;  pcbArr[i].serviceTime = -1;  pcbArr[i].priority = -1;  memset(pcbArr[i].state,0,sizeof(pcbArr[i].state));  }  }  //对到达时间从小到大排序  void MainWindow::sort()  {  int rows = ui->tableWidget->rowCount();  for(int i=0;i<rows;i++){  temp[i] = pcbArr[i];  temp[i].startTime = -1;  }  int i,j;  for(i = 1; i < rows; i++)  {  PCB t = temp[i];  for(j = i - 1; j >= 0 && t.arriveTime < temp[j].arriveTime; j--)  {  temp[j+1] = temp[j];  }  temp[j+1] = t;  }  }  //将pcb信息复制  void MainWindow::copyPcb()  {  int rows = ui->tableWidget->rowCount();  for(int a = 0;a<rows;a++)  {  for(int b = 0;b<rows;b++)  {  if(pcbArr[a].pid == temp[b].pid)  {  pcbArr[a].startTime = temp[b].startTime;  pcbArr[a].finishTime = temp[b].finishTime;  for(int i = 0;i<20;i++)  {  pcbArr[a].state[i] = temp[b].state[i];  }  a++;  }  }  }  }  //先来先服务  void MainWindow::fcfs()  {  sort();  int rows = ui->tableWidget->rowCount();  int sum = 0;  for(int k = 0;k<rows;k++)  {  if(k == 0)  {  temp[k].startTime = temp[k].arriveTime;  temp[k].finishTime = temp[k].startTime + temp[k].serviceTime;  temp[k].turnaroundTime = temp[k].finishTime - temp[k].arriveTime;  }  else  {  if(temp[k].arriveTime <= temp[k-1].finishTime)  {  temp[k].startTime = temp[k-1].finishTime;  }  else//第一个进程完成而第二个进程还未到达  {  temp[k].startTime = temp[k].arriveTime;  }  temp[k].finishTime = temp[k].startTime + temp[k].serviceTime;  temp[k].turnaroundTime = temp[k].finishTime - temp[k].arriveTime;  }  sum += temp[k].turnaroundTime;  }  fcfsAverageTurnaroundTime = sum / rows;  copyPcb();  }  //短作业优先  void MainWindow::sjf()  {  sort();  int rows = ui->tableWidget->rowCount();  int current = temp[0].arriveTime;//当前时间等于第一个进程的到达时间  int min\_p,lastmin\_p;  int flag[rows];//标记进程是否执行结束  int f = 0;//标记是否有上一次的下标  for(int i = 0;i<rows;i++)  {  flag[i] = 0;  }  while(1)  {  int compare[rows];//用于存放当前已到达但并未结束进程的剩余服务时间  for(int i = 0;i<rows;i++)  {  compare[i] = 0;  }  for(int i = 0;i<rows;i++)  {  if(temp[i].arriveTime <= current && flag[i] == 0)  {  compare[i] = temp[i].serviceTime;  }  }  int min = 100;  for(int i = 0;i<rows;i++)//找出当前进程最小剩余服务时间  {  if(compare[i] < min && compare[i] > 0)  {  min\_p = i;//标记该进程  min = compare[i];  }  }  if(f == 1)  {  if((min\_p != lastmin\_p) && (temp[lastmin\_p].state[current] != 3))//如果上一个进程被抢占  {  temp[lastmin\_p].state[current] = 1;//就绪状态  }  }  if(temp[min\_p].startTime == -1)  {  temp[min\_p].startTime = current;  }  temp[min\_p].serviceTime--;  temp[min\_p].state[current] = 2;//执行状态  current++;  if(temp[min\_p].serviceTime == 0)//判断当前进程是否结束  {  flag[min\_p] = 1;  temp[min\_p].finishTime = current;  temp[min\_p].state[current] = 3;//结束状态  }  if(temp[min\_p].serviceTime < 0)//判断所有进程是否结束  {  break;  }  lastmin\_p = min\_p;  f = 1;  }  int sum = 0;  for(int i = 0;i<rows;i++)  {  temp[i].turnaroundTime = temp[i].finishTime - temp[i].arriveTime;  sum += temp[i].turnaroundTime;  }  sjfAverageTurnaroundTime = sum / rows;  copyPcb();  }  //时间片轮转  void MainWindow::rr()  {  sort();  int time;  time = (ui->timeEdit->text()).toInt();  int rows = ui->tableWidget->rowCount();  int curNum;  int f;  int current = temp[0].arriveTime;//当前时间等于第一个进程的到达时间  int flag[rows];//标记进程是否执行结束  for(int j = 0;j<rows;j++)  {  flag[j] = 0;  }  while(1)  {  f = 0;  for(int i = 0; i<rows; i++)  {  if(temp[i].startTime == -1)  {  temp[i].startTime = current;  }  if(temp[i].serviceTime <= time && flag[i] == 0)//时间片内可以执行完毕  {  temp[i].state[current] = 2;//执行状态  current += temp[i].serviceTime;  temp[i].finishTime = current;  temp[i].state[current] = 3;//结束状态  flag[i] = 1;  }  curNum = 0;  if(temp[i].serviceTime > time && flag[i] == 0)//时间片内不能执行完毕  { cout<<i<<endl;  temp[i].state[current] = 2;//执行状态  current += time;  temp[i].serviceTime -= time;  for(int j = 0 ;j<rows;j++)  {  if(temp[j].arriveTime <= current)  {  curNum++;  }  }  if(i+1 == curNum && curNum <rows)//如果还有未到达的进程  {  i = -1;//重回第一个进程，因为for循环后要++因此i=-1即为ℹ=0  }  else  {  temp[i].state[current] = 1;//被抢占后变为就绪状态  cout << i<<current<<endl;  }  }  }  for(int i = 0;i<rows;i++)  {  if(flag[i] == 0 && f == 0)//所有进程未完成置f为1  {  f = 1;  }  }  if(f == 0)  {  break;  }  }  int sum = 0;  for(int i = 0;i<rows;i++)  {  temp[i].turnaroundTime = temp[i].finishTime - temp[i].arriveTime;  sum += temp[i].turnaroundTime;  }  rrAverageTurnaroundTime = sum / rows;  copyPcb();  }  //静态优先权优先调度  void MainWindow::pf()  {  sort();  int rows = ui->tableWidget->rowCount();  int current = temp[0].arriveTime;//当前时间等于第一个进程的到达时间  int min\_p,lastmin\_p;  int flag[rows];//标记进程是否执行结束  for(int i = 0;i<rows;i++)  {  flag[i] = 0;  }  int f = 0;//标记是否有上一次的下标  while(1)  {  int compare[rows];  for(int i = 0;i<rows;i++)  {  compare[i] = 0;  }  for(int i = 0;i<rows;i++)  {  if(temp[i].arriveTime <= current && flag[i] == 0)//将当前时间已到达且未完成的进程的优先级存入数组  {  compare[i] = temp[i].priority;  }  }  int min = 100;  for(int i = 0;i<rows;i++)//默认数值越小优先级越高  {  if(compare[i] < min && compare[i] > 0)  {  min\_p = i;  min = compare[i];//找出当前优先级最高的进程的下标和优先级  }  }  if(f == 1)  {  if((min\_p != lastmin\_p) && (temp[lastmin\_p].state[current] != 3))//如果上一个进程被抢占  {  temp[lastmin\_p].state[current] = 1;//就绪状态  }  }  if(temp[min\_p].startTime == -1)  {  temp[min\_p].startTime = current;  }  temp[min\_p].state[current] = 2;//执行状态  temp[min\_p].serviceTime--;  current++;  if(temp[min\_p].serviceTime == 0)  {  flag[min\_p] = 1;  temp[min\_p].finishTime = current;  temp[min\_p].state[current] = 3;//结束状态  }  if(temp[min\_p].serviceTime < 0)  {  break;  }  lastmin\_p = min\_p;//保留当前下标  f = 1;  }  int sum = 0;  for(int i = 0;i<rows;i++)  {  temp[i].turnaroundTime = temp[i].finishTime - temp[i].arriveTime;  sum += temp[i].turnaroundTime;  }  pfAverageTurnaroundTime = sum / rows;  copyPcb();  }  //高响应比调度  //等待时间=上一个的完成时间-该作业到达的时刻  //响应比=（等待时间+服务时间）/服务时间=等待时间/服务时间+1  void MainWindow::hrrn()  {  sort();  int rows = ui->tableWidget->rowCount();  int run\_p = 0;  int last\_p = 0;  int flag[rows];//标记进程是否执行结束  for(int i = 0;i<rows;i++)  {  flag[i] = 0;  }  while(1)  {  if(run\_p == 0)  {  temp[0].startTime = temp[0].arriveTime;  temp[0].finishTime = temp[0].startTime + temp[0].serviceTime;  flag[0] = 1;  }  int f = 0;  int compare[rows];  for(int i = 0;i<rows;i++)  {  compare[i] = 0;  }  for(int i = 1;i<rows;i++)  {  if(flag[i] == 0)  {  temp[i].waitTime = temp[last\_p].finishTime - temp[i].arriveTime;  temp[i].responseRatio = (temp[i].waitTime/temp[i].serviceTime)+1;  compare[i] = temp[i].responseRatio;  }  }  float maxRatio = 0;  for(int i = 1;i<rows;i++)  {  if(compare[i] > maxRatio)  {  run\_p = i;  maxRatio = compare[i];  }  }  temp[run\_p].startTime = temp[last\_p].finishTime;  temp[run\_p].finishTime = temp[run\_p].startTime + temp[run\_p].serviceTime;  flag[run\_p] = 1;  last\_p = run\_p;  for(int i = 0;i<rows;i++)  {  if(flag[i] == 0 && f == 0)  {  f = 1;  }  }  if(f == 0)  {  break;  }  }  int sum = 0;  for(int i = 0;i<rows;i++)  {  temp[i].turnaroundTime = temp[i].finishTime - temp[i].arriveTime;  sum += temp[i].turnaroundTime;  }  hrrnAverageTurnaroundTime = sum / rows;  copyPcb();  }  //手动添加进程  void MainWindow::on\_addButton\_clicked()  {  int rows = ui->tableWidget->rowCount();  ui->tableWidget->insertRow(rows);  for(int i=0;i<4;i++){  ui->tableWidget->setItem(rows,i,new QTableWidgetItem("0"));  }  ui->tableWidget->setItem(rows,6,new QTableWidgetItem(" "));  ui->tableWidget->selectRow(rows);  // for (int j=0;j<4;j++){  // ui->tableWidget->item(rows,j)->setTextAlignment(Qt::AlignHCenter|Qt::AlignVCenter);  // }  }  //开始  void MainWindow::on\_startButton\_clicked()  {  initPcb();  if(ui->tableWidget->rowCount() == 0)//如果表格无数据  {  QMessageBox msg;  msg.setText("请添加数据！");  msg.exec();  return;  }  second = 0;  if(ui->comboBox->currentText() == "先来先服务")  {  fcfs();  }  if(ui->comboBox->currentText() == "短作业优先")  {  sjf();  }  if(ui->comboBox->currentText() == "时间片轮转")  {  rr();  }  if(ui->comboBox->currentText() == "静态优先权优先调度")  {  pf();  }  if(ui->comboBox->currentText() == "高响应比调度")  {  hrrn();  }  timer = new QTimer();  connect(timer,SIGNAL(timeout()),this,SLOT(updateTime()));  timer->start(1000);  }  //动态更新表格数据  void MainWindow::updateTime()  {  int rows = ui->tableWidget->rowCount();  ui->timer->setText(QString::number(second));  for(int i = 0;i<rows;i++)  {  if(second < pcbArr[i].startTime && second >= pcbArr[i].arriveTime)  {  ui->tableWidget->setItem(i,6,new QTableWidgetItem("就绪"));  }  if(pcbArr[i].state[second] == 1)  {  ui->tableWidget->setItem(i,6,new QTableWidgetItem("等待"));  }  if(pcbArr[i].state[second] == 2)  {  ui->tableWidget->setItem(i,6,new QTableWidgetItem("执行"));  }  if(second == pcbArr[i].startTime)  {  ui->tableWidget->setItem(i,4,new QTableWidgetItem(QString::number(pcbArr[i].startTime)));  ui->tableWidget->setItem(i,6,new QTableWidgetItem("执行"));  //ui->tableWidget->item(i,4)->setTextAlignment(Qt::AlignHCenter|Qt::AlignVCenter);  }  if(second == pcbArr[i].finishTime)  {  ui->tableWidget->setItem(i,5,new QTableWidgetItem(QString::number(pcbArr[i].finishTime)));  ui->tableWidget->setItem(i,6,new QTableWidgetItem("结束"));  //ui->tableWidget->item(i,5)->setTextAlignment(Qt::AlignHCenter|Qt::AlignVCenter);  }  }  //当所有进程结束时，定时器关闭  int flag = rows;  for(int j = 0;j<rows;j++)  {  if((ui->tableWidget->item(j,6)->text()) == "结束")  {  flag--;  }  }  if(flag == 0)  {  if(ui->comboBox->currentText() == "先来先服务")  {  ui->fcsfEdit->setText(QString::number(fcfsAverageTurnaroundTime));  }  if(ui->comboBox->currentText() == "短作业优先")  {  ui->sjfEdit->setText(QString::number(sjfAverageTurnaroundTime));  }  if(ui->comboBox->currentText() == "时间片轮转")  {  ui->rrEdit->setText(QString::number(rrAverageTurnaroundTime));  }  if(ui->comboBox->currentText() == "静态优先权优先调度")  {  ui->pfEdit->setText(QString::number(pfAverageTurnaroundTime));  }  if(ui->comboBox->currentText() == "高响应比调度")  {  ui->hrrnEdit->setText(QString::number(hrrnAverageTurnaroundTime));  }  timer->stop();  delete timer;  }  else  {  second++;  }  }  //删除表格所有行  void MainWindow::on\_clearButton\_clicked()  {  int rows = ui->tableWidget->rowCount();  clearPcb(rows);  for(int i = 0;i<rows;i++)  {  ui->tableWidget->removeRow(0);  }  ui->timer->setText(QString::number(0));  ui->fcsfEdit->setText("");  ui->sjfEdit->setText("");  ui->rrEdit->setText("");  ui->pfEdit->setText("");  ui->hrrnEdit->setText("");  }  //更换选项清空上一次结果  void MainWindow::on\_comboBox\_currentIndexChanged(int index)  {  int rows = ui->tableWidget->rowCount();  switch(index)  {  case 1:  ui->label->setVisible(true);  ui->timeEdit->setVisible(true);  for(int i = 0;i<rows;i++)  {  ui->tableWidget->setItem(i,4,new QTableWidgetItem(""));  ui->tableWidget->setItem(i,5,new QTableWidgetItem(""));  ui->tableWidget->setItem(i,6,new QTableWidgetItem(""));  }  break;  default:  ui->label->setVisible(false);  ui->timeEdit->setVisible(false);  for(int i = 0;i<rows;i++)  {  ui->tableWidget->setItem(i,4,new QTableWidgetItem(""));  ui->tableWidget->setItem(i,5,new QTableWidgetItem(""));  ui->tableWidget->setItem(i,6,new QTableWidgetItem(""));  }  break;  }  }  void MainWindow::on\_fileButton\_clicked()  {  QString fileName;  fileName = QFileDialog::getOpenFileName(this,tr("open file"),"",tr("text(\*.txt)"));  if(!fileName.isEmpty())  {  QFile file;  file.setFileName(fileName);  //打开文件  if(!file.open(QIODevice::ReadOnly | QIODevice::Text))  {  QMessageBox msg;  msg.setText("打开文件失败!");  msg.exec();  }  else  {  if(!file.size())  {  QMessageBox::warning(0,tr("警告"),tr("文件大小为空!"),QMessageBox::Ok);  }  else  {  //将文件数据导入表格  int r\_count = 0; //统计文件的行数  QStringList textList; //记录文件中每一行的数据  QTextStream in(&file);  while (!in.atEnd())  {  QString line = in.readLine();  textList.append(line); //保存文件的数据  r\_count++; //记录文件的行数  }  file.close(); //关闭文件  if(!textList.isEmpty())  {  ui->tableWidget->setRowCount(r\_count);  for(int row = 0; row < r\_count; row++)  {  QStringList tmpList;  tmpList = textList.at(row).split(" ");  for(int col = 0; col < 4; col++)  {  QTableWidgetItem \*item = new QTableWidgetItem(tmpList.at(col));  ui->tableWidget->setItem(row, col, item);  }  ui->tableWidget->setItem(row,6,new QTableWidgetItem(" "));  }  }  }  }  }  } |