项目文档

谈瑞

项目名称：电梯调度系统

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 专业班级：软件工程4班  电话：18936361545 | 学号：1452775  电子邮件：tanrui106@163.com | 课目：操作系统小项目  个人网站：http://guitoubing.top |
|  |  |  |

目录

[项目简介 3](#_Toc513997000)

[项目需求 3](#_Toc513997001)

[开发环境及语言 3](#_Toc513997002)

[项目基本思想 3](#_Toc513997003)

[主要功能代码分析 4](#_Toc513997004)

[1. 电梯线程 4](#_Toc513997005)

[成员变量 4](#_Toc513997006)

[成员函数 5](#_Toc513997007)

[2. 添加用户线程 7](#_Toc513997008)

[成员变量 7](#_Toc513997009)

[成员函数 8](#_Toc513997010)

[各线程调度类 9](#_Toc513997011)

[运行说明 13](#_Toc513997012)

# 项目简介

## 项目需求

假设每个页面可存放10条指令，分配给一个作业的内存块为4。模拟一个作业的执行过程，该作业有320条指令，即它的地址空间为32页，目前所有页还没有调入内存。

在模拟过程中，如果所访问指令在内存中，则显示其物理地址，并转到下一条指令；如果没有在内存中，则发生缺页，此时需要记录缺页次数，并将其调入内存。如果4个内存块中已装入作业，则需进行页面置换。

所有320条指令执行完成后，计算并显示作业执行过程中发生的缺页率。置换算法选用FIFO算法。

作业中指令访问次序可以按照下面原则形成：50%的指令是顺序执行的，25%是均匀分布在前地址部分，25％是均匀分布在后地址部分。

## 开发环境及语言

本项目采用JavaFX（Java图形界面库）编程技术，开发工具采用IntelliJ IDEA Community Edition，在win10下开发，Java版本号为9.0.4

## 项目基本思想

本项目使用Java语言编程，创建Work类和Memory类，Work类表示作业，一个作业包含320条指令及32页，Memory类表示内存，包含4个内存块。

实施方法如下：

在0－319条指令之间，随机选取一个起始执行指令，如序号为m，顺序执行下一条指令，即序号为m+1的指令，通过随机数，跳转到前地址部分0－m-1中的某个指令处，其序号为m1，顺序执行下一条指令，即序号为m1+1的指令，通过随机数，跳转到后地址部分m1+2~319中的某条指令处，其序号为m2，顺序执行下一条指令，即m2+1处的指令。重复跳转到前地址部分、顺序执行、跳转到后地址部分、顺序执行的过程，直到执行完320条指令。

# 主要功能代码分析

## Work类

### 成员变量

// 电梯编号  
private int id;  
// 电梯当前楼层  
private int presentFloor;  
// 电梯目标楼层  
private int aimFloor;  
// 电梯运行方向（true为上行，false为下行）  
private boolean direction;  
// 当前正在处理的用户请求  
private UserRequest runningRequest;  
// 最后一个用户请求  
private UserRequest lastRequest;  
// 等待的用户请求列表（采用优先级队列实现）  
private PriorityBlockingQueue<UserRequest> waitingRequest;  
// 当前所需等待时间  
private double waitingTime;  
// 绑定到界面上的楼层label  
private Label label;  
// 绑定到界面上的电梯状态label  
private Label state;  
// 绑定到界面上的电梯日志TextArea  
private TextArea log;  
// 是否正在运行  
public boolean isRun;  
// 是否已停止运行  
public boolean exit;  
private static final String *RUN\_UP* = "上行";  
private static final String *RUN\_DOWN* = "下行";  
private static final String *STOP* = "停止";  
private static final String *OPEN* = "开门";  
private static final String *CLOSE* = "关门";  
// 每层楼所需等待时间  
private static final long *TIME\_ON\_EACH\_FLOOR* = 1000;  
// 开关门所需时间  
private static final long *WAITING\_TIME* = 2000;  
// 线程停止  
private static final int *THREADSTOP* = -1;  
// 线程运行  
private static final int *THREADRUNNING* = 1;  
// 线程等待  
private static final int *THREADWAITING* = 0;  
// 线程复苏  
private static final int *THREADRESUME* = 2;  
// 是否被报警  
public boolean WARNING = false;  
// 电梯状态  
private int status;

### 成员函数

/\*  
 处理一个用户请求，从等待的用户请求列表中取出优先级最高的用户，并放入电梯中，而后处理该用户请求  
\*/  
public synchronized void runARequest() throws InterruptedException {  
 // 从用户列表中取出优先级最高的用户  
 if (waitingRequest.isEmpty()){  
 runningRequest = null;  
 isRun = false;  
 }else {  
 runningRequest = waitingRequest.poll();  
 }  
 // 处理用户请求  
 if (runningRequest != null){  
 isRun = true;  
 aimFloor = runningRequest.getPresentFloor();  
 direction = aimFloor > presentFloor;  
 if (direction)  
 Platform.*runLater*(()-> state.setText(*RUN\_UP*));  
 else  
 Platform.*runLater*(()-> state.setText(*RUN\_DOWN*));  
 // 从当前楼层前往当前用户所在楼层  
 while (presentFloor != aimFloor && !exit){  
 if (exit)  
 return;  
 if (status != *THREADWAITING*) {  
 Thread.*sleep*(*TIME\_ON\_EACH\_FLOOR*);  
 if (direction)  
 presentFloor++;  
 else  
 presentFloor--;  
 Platform.*runLater*(() -> {  
 if (label != null)  
 label.setText(String.*valueOf*(presentFloor));  
 });  
 waitingTime -= *TIME\_ON\_EACH\_FLOOR*;  
 }else  
 return;  
 }  
 // 等待用户进入电梯  
 isRun = false;  
 Platform.*runLater*(()->state.setText(*OPEN*));  
 Thread.*sleep*(3000);  
 Platform.*runLater*(()->state.setText(*CLOSE*));  
 Thread.*sleep*(1000);  
 // 运载用户前往所需要去的楼层  
 isRun = true;  
 waitingTime -= *WAITING\_TIME*;  
 aimFloor = runningRequest.getAimFloor();  
 String logStr = "电梯日志——进电梯 : " + id + "号电梯有用户进入，需前往第" + aimFloor + "层\n";  
 final String finalLogStr = logStr;  
 Platform.*runLater*(()-> log.appendText(finalLogStr));  
 direction = aimFloor > presentFloor;  
 if (direction)  
 Platform.*runLater*(()-> state.setText(*RUN\_UP*));  
 else  
 Platform.*runLater*(()-> state.setText(*RUN\_DOWN*));  
 while (presentFloor != aimFloor && !exit){  
 if (exit)  
 return;  
 if (status != *THREADWAITING*) {  
 Thread.*sleep*(*TIME\_ON\_EACH\_FLOOR*);  
 if (direction)  
 presentFloor++;  
 else  
 presentFloor--;  
 Platform.*runLater*(() -> {  
 if (label != null)  
 label.setText(String.*valueOf*(presentFloor));  
 });  
 System.*out*.println("电梯"+id+"当前楼层"+presentFloor);  
 }else {  
 return;  
 }  
 }  
 waitingTime -= runningRequest.getTime();  
 logStr = "电梯日志——出电梯 : " + id + "号电梯有用户出电梯，停在第" + presentFloor + "层\n";  
 final String finalLogStr1 = logStr;  
 Platform.*runLater*(()-> log.appendText(finalLogStr1));  
 // 等待用户出电梯，并处理下一个用户请求  
 Platform.*runLater*(()-> state.setText(*OPEN*));  
 Thread.*sleep*(3000);  
 Platform.*runLater*(()-> state.setText(*CLOSE*));  
 Thread.*sleep*(1000);  
 Platform.*runLater*(()-> state.setText(*STOP*));  
 }  
}  
  
/\*  
 重写线程run方法  
\*/  
@Override  
public synchronized void run(){  
 while (!exit){  
 try {  
 runARequest();  
 } catch (InterruptedException e) {  
 System.*out*.println("无法接受一个用户请求 ");  
 }  
 }  
}  
  
/\*  
 添加一个用户请求  
 \*/  
public void addWaitingRequest(UserRequest userRequest){  
 waitingRequest.put(userRequest);  
 if (lastRequest == null)  
 waitingTime += (userRequest.getTime() + *abs*(presentFloor-userRequest.getPresentFloor())\**TIME\_ON\_EACH\_FLOOR* + *WAITING\_TIME*);  
 else  
 waitingTime += (userRequest.getTime() + *abs*(lastRequest.getAimFloor() - userRequest.getPresentFloor())\**TIME\_ON\_EACH\_FLOOR* + *WAITING\_TIME*);  
 isRun = true;  
}  
  
/\*  
 电梯优先级比较  
\*/  
@Override  
public int compareTo(Object o) {  
 Elevator elevator = (Elevator) o;  
 if (this.waitingTime >= elevator.waitingTime){  
 return 1;  
 }  
 else {  
 return -1;  
 }  
}

## 添加用户线程

### 成员变量

private static final long *TIME\_ON\_EACH\_FLOOR* = 1000;  
private static final long *WAITING\_TIME* = 2000;  
private static final int *THREADSTOP* = -1;  
private static final int *THREADRUNNING* = 1;  
private static final int *THREADWAITING* = 0;  
private static final int *THREADRESUME* = 2;  
private static final int *ELEVATORS* = 5;  
// 用户请求列表  
PriorityBlockingQueue<UserRequest> userRequests;  
// 链接到所有电梯线程  
ArrayList<Elevator> elevators;  
// 线程状态  
private int status;  
// 绑定到界面上的电梯日志  
private TextArea log;  
// 当前线程是否退出  
public boolean exit;

### 成员函数

/\*  
 重写run方法  
 \*/  
@Override  
public synchronized void run() {  
 while (!exit){  
 if (status != *THREADWAITING*){  
 UserRequest userRequest = userRequests.poll();  
 if (userRequest != null){  
 int choose = ChooseElevator(userRequest);  
 if (choose == -1){  
 Platform.*runLater*(()->log.appendText("电梯日志——提示 ： 当前所有电梯均处于不可用状态\n"));  
 }else {  
 elevators.get(choose).addWaitingRequest(userRequest);  
 System.*out*.println("用户" + userRequest.getId() + "进入电梯" + elevators.get(choose).getId() + "的排队序列");  
 }  
 }  
 }  
 }  
}  
  
/\*  
 对每个提交用户请求进行电梯调度，选择用时最少的电梯  
 \*/  
public int ChooseElevator(UserRequest userRequest){  
 ArrayList<Double> chooses = new ArrayList<>();  
 for (Elevator e : elevators){  
 chooses.add(CalcWaitingTime(e, userRequest));  
 }  
 double minTime = -1;  
 int i = 0;  
 for (; i < chooses.size(); i++){  
 if (elevators.get(i).WARNING){  
 continue;  
 }else {  
 minTime = chooses.get(i);  
 break;  
 }  
 }  
 if (minTime == -1){  
 return -1;  
 }else {  
 int index = i;  
 for (; i < chooses.size(); i++){  
 if (chooses.get(i) < minTime){  
 minTime = chooses.get(i);  
 index = i;  
 }  
 }  
 return index;  
 }  
}  
  
/\*  
 计算某一电梯对于当前用户请求所需要的等待时间  
 \*/  
private double CalcWaitingTime(Elevator e, UserRequest userRequest){  
 if (!e.isRun){  
 return *abs*(e.getPresentFloor() - userRequest.getPresentFloor())\**TIME\_ON\_EACH\_FLOOR*;  
 }  
 else {  
 if ((e.getDirection() && e.getPresentFloor() >= userRequest.getPresentFloor()) || (!e.getDirection() && e.getPresentFloor() < userRequest.getPresentFloor())){  
 double time = *abs*(e.getAimFloor() - e.getPresentFloor())\*2\**TIME\_ON\_EACH\_FLOOR* + *WAITING\_TIME* + *abs*(e.getPresentFloor() - userRequest.getPresentFloor())\**TIME\_ON\_EACH\_FLOOR*;  
 for (UserRequest request : e.getWaitingRequest()){  
 time += request.getTime();  
 }  
 return time;  
 }  
 else {  
 double time = *abs*(e.getPresentFloor() - userRequest.getPresentFloor())\**TIME\_ON\_EACH\_FLOOR*;  
 for (UserRequest request : e.getWaitingRequest()){  
 time += request.getTime();  
 }  
 return time;  
 }  
 }  
}

## 各线程调度类

public class Controller {  
 static boolean *RUNNED* = false;  
 static int *RequestID* = 1;  
 static int *ElevatorID* = 1;  
 static final int *ELEVATORS* = 5;  
 static AddRequest *addRequest*;  
 static ArrayList<Elevator> *elevators*;  
 @FXML  
 private TextField presentFloor;  
 @FXML  
 private TextField aimFloor;  
 @FXML  
 private Label elevator1;  
 @FXML  
 private Label elevator2;  
 @FXML  
 private Label elevator3;  
 @FXML  
 private Label elevator4;  
 @FXML  
 private Label elevator5;  
 ArrayList<Label> labels;  
 @FXML  
 private Label state1;  
 @FXML  
 private Label state2;  
 @FXML  
 private Label state3;  
 @FXML  
 private Label state4;  
 @FXML  
 private Label state5;  
 ArrayList<Label> states;  
 @FXML  
 private TextArea log;  
 public static boolean *SystemRunning* = false;  
 @FXML  
 protected void onClicked() {  
 if (*SystemRunning*)  
 *addRequest*.userRequests.put(new UserRequest(*RequestID*++, Integer.*parseInt*(presentFloor.getText()), Integer.*parseInt*(aimFloor.getText())));  
 else  
 log.appendText("电梯日志——提示 ： 当前所有电梯均处于不可用状态\n");  
 }  
 @FXML  
 protected synchronized void run(){  
 if (!*SystemRunning*){  
 *ElevatorID* = 1;  
 *RUNNED* = true;  
 log.appendText("电梯恢复运行！\n");  
 *SystemRunning* = true;  
 System.*out*.println("系统状态："+*SystemRunning*);  
 log.appendText("");  
 if (labels == null) {  
 labels = new ArrayList<>();  
 labels.add(elevator1);  
 labels.add(elevator2);  
 labels.add(elevator3);  
 labels.add(elevator4);  
 labels.add(elevator5);  
 }  
 if (states == null) {  
 states = new ArrayList<>();  
 states.add(state1);  
 states.add(state2);  
 states.add(state3);  
 states.add(state4);  
 states.add(state5);  
 }  
 // 创建电梯进程  
 *elevators* = new ArrayList<>();  
 for (int i = 0; i < *ELEVATORS*; i++){  
 Elevator elevator = new Elevator(*ElevatorID*++, labels.get(i), states.get(i), log);  
 *elevators*.add(elevator);  
 elevator.start();  
 }  
 // 创建处理用户进程  
 *addRequest* = new AddRequest(*elevators*, log);  
 *addRequest*.start();  
 }  
 }  
 @FXML  
 protected synchronized void stop() throws InterruptedException {  
 if (*SystemRunning*){  
 *SystemRunning* = false;  
 log.appendText("电梯日志——报警 ： 所有电梯停止运行（当前用户出电梯后停止）！\n");  
 for (Elevator e : *elevators*){  
 e.WARNING = true;  
 }  
 }  
 }  
 @FXML  
 protected synchronized void stop1(){  
 if (!*elevators*.get(0).WARNING){  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(0).getId() + "号电梯有用户报警！\n");  
 *elevators*.get(0).WARNING = true;  
 *elevators*.get(0).setWaitingRequest(new PriorityBlockingQueue<>(200, new UserRequestComparator()));  
 }  
 }  
 @FXML  
 protected synchronized void stop2(){  
 if (!*elevators*.get(1).WARNING) {  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(1).getId() + "号电梯有用户报警！\n");  
 *elevators*.get(1).WARNING = true;  
 *elevators*.get(1).setWaitingRequest(new PriorityBlockingQueue<>(200, new UserRequestComparator()));  
 }  
 }  
 @FXML  
 protected synchronized void stop3(){  
 if (!*elevators*.get(2).WARNING) {  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(2).getId() + "号电梯有用户报警！\n");  
 *elevators*.get(2).WARNING = true;  
 *elevators*.get(2).setWaitingRequest(new PriorityBlockingQueue<>(200, new UserRequestComparator()));  
 }  
 }  
 @FXML  
 protected synchronized void stop4(){  
 if (!*elevators*.get(3).WARNING) {  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(3).getId() + "号电梯有用户报警！\n");  
 *elevators*.get(3).WARNING = true;  
 *elevators*.get(3).setWaitingRequest(new PriorityBlockingQueue<>(200, new UserRequestComparator()));  
 }  
 }  
 @FXML  
 protected synchronized void stop5(){  
 if (!*elevators*.get(4).WARNING) {  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(4).getId() + "号电梯有用户报警！\n");  
 *elevators*.get(4).WARNING = true;  
 *elevators*.get(4).setWaitingRequest(new PriorityBlockingQueue<>(200, new UserRequestComparator()));  
 }  
 }  
  
 @FXML  
 protected synchronized void resume1(){  
 if (*elevators*.get(0).WARNING){  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(0).getId() + "号电梯报警解除！\n");  
 *elevators*.get(0).WARNING = false;  
 }  
 }  
 @FXML  
 protected synchronized void resume2(){  
 if (*elevators*.get(1).WARNING){  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(1).getId() + "号电梯报警解除！\n");  
 *elevators*.get(1).WARNING = false;  
 }  
 }  
 @FXML  
 protected synchronized void resume3(){  
 if (*elevators*.get(2).WARNING){  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(2).getId() + "号电梯报警解除！\n");  
 *elevators*.get(2).WARNING = false;  
 }  
 }  
 @FXML  
 protected synchronized void resume4(){  
 if (*elevators*.get(3).WARNING){  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(3).getId() + "号电梯报警解除！\n");  
 *elevators*.get(3).WARNING = false;  
 }  
 }  
 @FXML  
 protected synchronized void resume5(){  
 if (*elevators*.get(4).WARNING){  
 log.appendText("电梯日志——报警 ： " + *elevators*.get(4).getId() + "号电梯报警解除！\n");  
 *elevators*.get(4).WARNING = false;  
 }  
 }  
}

# 运行说明

1.首先双击jar包打开程序

2. 点击“开始运行电梯”按钮运行电梯调度系统

3. 点击“我要上电梯”以提交请求