项目文档

谈瑞

项目名称：页面调度程序

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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](#_Toc515267549)

[项目需求 3](#_Toc515267550)

[开发环境及语言 3](#_Toc515267551)

[项目基本思想 3](#_Toc515267552)

[主要功能代码分析 4](#_Toc515267553)

[Work类 4](#_Toc515267554)

[Memory类 5](#_Toc515267555)

[Controller类 7](#_Toc515267556)

[运行说明 11](#_Toc515267557)

# 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

import java.util.ArrayList;  
import java.util.Random;  
public class Work {  
 //页面类  
 public class Page{  
 //页面类  
 Integer pageID;  
 ArrayList<Instruct> instructs;  
 Integer memoryNum;  
 boolean inMemory;  
  
 public Integer getPageID() {  
 return pageID;  
 }  
  
 public void setPageID(Integer pageID) {  
 this.pageID = pageID;  
 }  
 public Page(Integer pageID, ArrayList<Instruct> instructs) {  
 this.pageID = pageID;  
 this.instructs = instructs;  
 }  
  
 public ArrayList<Instruct> getInstructs() {  
 return instructs;  
 }  
  
 public void setInstructs(ArrayList<Instruct> instructs) {  
 this.instructs = instructs;  
 }  
  
 public Integer getMemoryNum() {  
 return memoryNum;  
 }  
  
 public void setMemoryNum(Integer memoryNum) {  
 this.memoryNum = memoryNum;  
 }  
  
 public boolean isInMemory() {  
 return inMemory;  
 }  
  
 public void setInMemory(boolean inMemory) {  
 this.inMemory = inMemory;  
 }  
  
 public Page(Integer pageID) {  
 this.pageID = pageID;  
 this.instructs = new ArrayList<>();  
 this.memoryNum = 0;  
 this.inMemory = false;  
 }  
 }  
 public ArrayList<Page> getPages() {  
 return pages;  
 }  
 public void setPages(ArrayList<Page> pages) {  
 this.pages = pages;  
 }  
 //指令类  
 public class Instruct{  
 Integer instructNum;  
 boolean isExecute;  
 public Instruct(Integer instructNum, boolean isExecute) {  
 this.instructNum = instructNum;  
 this.isExecute = isExecute;  
 }  
 }  
 //共320条指令  
 public final Integer instructNum = 320;  
 public final Integer minInstruct = 0;  
 public final Integer maxInstruct = 319;  
 //每页存储10条指令  
 public final Integer instructsInEachPage = 10;  
 //未执行的指令数  
 public Integer remainedInstruct = 320;  
 //执行次数  
 public Integer executeCount = 0;  
 //首条指令  
 public Integer firstInstruct = new Random().nextInt(maxInstruct);  
 //存储页面  
 private ArrayList<Page> pages;  
 //构造函数  
 public Work(ArrayList<Page> pages){  
 this.pages = pages;  
 }  
 //默认构造函数，初始化页面  
 public Work() {  
 pages = new ArrayList<>();  
 for (int i = 0; i < instructNum/instructsInEachPage; i++){  
 Page page = new Page(i);  
 for (int j = 0; j < instructsInEachPage; j++){  
 page.instructs.add(new Instruct(i\*instructsInEachPage+j, false));  
 }  
 pages.add(page);  
 }  
 }  
 //完成一个指令，当该指令未被执行过，remainedInstruct减一  
 public void finishAnInstruct(){  
 remainedInstruct--;  
 }  
 //判断作业是否完成  
 public boolean finishWork(){  
 return remainedInstruct == 0;  
 }  
}

## Memory类

import java.util.ArrayList;  
import java.util.concurrent.LinkedBlockingQueue;  
public class Memory {  
 // 四个内存块及其编号  
 public final Integer minMemoryBlock = 1;  
 public final Integer maxMemoryBlock = 4;  
 public final Integer MemoryBlockNum = 4;  
 // 缺页数  
 public int lostPageCount = 0;  
 // 内存块类  
 public class Block{  
 Integer ID;  
 boolean isNull;  
 Integer pageID;  
  
 public Integer getID() {  
 return ID;  
 }  
  
 public void setID(Integer ID) {  
 this.ID = ID;  
 }  
  
 public boolean isNull() {  
 return isNull;  
 }  
  
 public void setNull(boolean aNull) {  
 isNull = aNull;  
 }  
  
 public Integer getPageID() {  
 return pageID;  
 }  
  
 public void setPageID(Integer pageID) {  
 this.pageID = pageID;  
 }  
  
 Block(Integer ID){  
 this.ID = ID;  
 this.isNull = true;  
 this.pageID = -1;  
 }  
 }  
 // 页面队列，放在四个内存块中  
 private LinkedBlockingQueue<Work.Page> pageQueue;  
 // 存放四个内存块  
 private ArrayList<Block> memoryBlocks;  
 public ArrayList<Block> getMemoryBlocks() {  
 return memoryBlocks;  
 }  
 // 构造函数  
 public Memory(){  
 this.pageQueue = new LinkedBlockingQueue<>();  
 this.memoryBlocks = new ArrayList<>();  
 for (int i = minMemoryBlock; i <= maxMemoryBlock; i++){  
 memoryBlocks.add(new Block(i));  
 }  
 }  
 // 执行一条指令  
 public String executeAnInstruct(Work work, Integer instruct) throws InterruptedException {  
 // 执行次数自增  
 work.executeCount++;  
 System.*out*.println("当前剩余指令数："+work.remainedInstruct);  
 int pageNum = instruct/work.instructsInEachPage;  
 int instructNum = instruct - pageNum\*work.instructsInEachPage;  
 StringBuilder stringBuilder = new StringBuilder().append("执行指令").append(instruct).append("，指令位于页面").append(pageNum).append("，页面偏移为").append(instructNum).append("\n");  
 Work.Page page = work.getPages().get(pageNum);  
 //若该指令未被执行过，则将其标记为已被执行，且剩余未执行指令减少1  
 if (!page.getInstructs().get(instructNum).isExecute){  
 page.getInstructs().get(instructNum).isExecute = true;  
 work.finishAnInstruct();  
 stringBuilder.append("该指令未被执行过，");  
 }else {  
 stringBuilder.append("该指令已被执行过，");  
 }  
 // 若页面不在内存块中，则显示缺页，并进行页面调度  
 if (!page.inMemory){  
 lostPageCount++;  
 if (pageQueue.size() == MemoryBlockNum){  
 Work.Page lostPage = pageQueue.poll();  
 assert lostPage != null;  
 stringBuilder.append("四个内存块已被占满\n将页面").append(lostPage.pageID).append("置换出来，\n将页面").append(pageNum).append("放入").append(lostPage.memoryNum).append("号内存块中，并执行该指令。");  
 page.memoryNum = lostPage.memoryNum;  
 page.inMemory = true;  
 lostPage.inMemory = false;  
 lostPage.memoryNum = 0;  
 memoryBlocks.get(page.memoryNum - 1).isNull = false;  
 memoryBlocks.get(page.memoryNum - 1).pageID = page.pageID;  
 try {  
 pageQueue.put(page);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
// System.out.println("页面" + lostPage.pageID + "被被置换，" + "页面" + page.pageID + "被加入内存块" + memoryBlocks.get(lostPage.memoryNum - 1).ID);  
 }else {  
 int memoryNum = pageQueue.size() + 1;  
 stringBuilder.append("内存块仍未占满\n将页面").append(pageNum).append("放入").append(memoryNum).append("号内存块中，并执行该指令。");  
 page.memoryNum = memoryNum;  
 page.inMemory = true;  
 memoryBlocks.get(memoryNum - 1).isNull = false;  
 memoryBlocks.get(memoryNum - 1).pageID = page.pageID;  
 pageQueue.put(page);  
// System.out.println("页面" + page.pageID + "被加入内存块" + memoryBlocks.get(page.memoryNum - 1).ID);  
 }  
 }else {  
 //否则直接执行指令即可  
 stringBuilder.append("页面").append(pageNum).append("已在内存块中，直接执行指令。");  
 }  
 return stringBuilder.toString();  
 }  
  
}

## Controller类

负责对界面元素控制，以及对点击事件响应

import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextArea;  
import java.util.ArrayList;  
import java.util.Random;  
  
public class Controller {  
 @FXML  
 private TextArea instructDetail;  
 @FXML  
 private TextArea workDetail;  
 @FXML  
 private Label page1WithoutExecute;  
 @FXML  
 private Label page2WithoutExecute;  
 @FXML  
 private Label page3WithoutExecute;  
 @FXML  
 private Label page4WithoutExecute;  
 @FXML  
 private Label page1WithExecute;  
 @FXML  
 private Label page2WithExecute;  
 @FXML  
 private Label page3WithExecute;  
 @FXML  
 private Label page4WithExecute;  
  
 private Work work;  
 private Memory memory;  
 private int step = 0;  
  
 @FXML// 单指令运行  
 void executeOne() throws InterruptedException {  
 instructDetail.setText("");  
 if (work == null && memory == null){  
 instructDetail.setEditable(false);  
 work = new Work();  
 memory = new Memory();  
 }  
 ArrayList<Integer> ps = new ArrayList<>();  
 for (int i = 0; i < memory.MemoryBlockNum; i++){  
 ps.add(memory.getMemoryBlocks().get(i).pageID);  
 }  
 assert work != null;  
 if (!work.finishWork()){  
 int m = work.firstInstruct;  
 String detail = "";  
 if (step == 0){  
 //0到m-1指令中随机执行  
 work.firstInstruct = new Random().nextInt(m);  
 m = work.firstInstruct;  
 detail = memory.executeAnInstruct(work, m);  
 step++;  
 }  
 else if (!work.finishWork() && step == 1){  
 detail = memory.executeAnInstruct(work, m+1);  
 step++;  
 }  
 else if (!work.finishWork() && step == 2){  
 //m-1到319指令中随机执行  
 work.firstInstruct = (new Random().nextInt(work.maxInstruct - m)) + m;  
 m = work.firstInstruct;  
 detail = memory.executeAnInstruct(work, m);  
 step++;  
 }  
 else if (!work.finishWork() && step == 3){  
 detail = memory.executeAnInstruct(work, m+1);  
 step=0;  
 }  
 instructDetail.setText(detail);  
 for (int i = 0; i < memory.MemoryBlockNum; i++){  
 ps.add(memory.getMemoryBlocks().get(i).pageID);  
 }  
 updatePage(ps.get(0), ps.get(1), ps.get(2), ps.get(3), ps.get(4), ps.get(5), ps.get(6), ps.get(7));  
 }  
 }  
  
 // 更新内存块信息  
 private void updatePage(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8){  
 page1WithoutExecute.setText(p1 == -1 ? "空" : String.*valueOf*(p1));  
 page2WithoutExecute.setText(p2 == -1 ? "空" : String.*valueOf*(p2));  
 page3WithoutExecute.setText(p3 == -1 ? "空" : String.*valueOf*(p3));  
 page4WithoutExecute.setText(p4 == -1 ? "空" : String.*valueOf*(p4));  
 page1WithExecute.setText(p5 == -1 ? "空" : String.*valueOf*(p5));  
 page2WithExecute.setText(p6 == -1 ? "空" : String.*valueOf*(p6));  
 page3WithExecute.setText(p7 == -1 ? "空" : String.*valueOf*(p7));  
 page4WithExecute.setText(p8 == -1 ? "空" : String.*valueOf*(p8));  
 }  
  
 @FXML// 单作业运行，执行所有指令  
 void executeAll() throws InterruptedException {  
 Work work = new Work();  
 Memory memory = new Memory();  
 while (!work.finishWork()){  
 int m = work.firstInstruct;  
 //0到m-1指令中随机执行  
 work.firstInstruct = new Random().nextInt(m);  
 m = work.firstInstruct;  
 appendToWorkDetail(memory.executeAnInstruct(work, m), work);  
 if (!work.finishWork()){  
 appendToWorkDetail(memory.executeAnInstruct(work, m+1), work);  
 }  
 if (!work.finishWork()){  
 //m-1到319指令中随机执行  
 work.firstInstruct = (new Random().nextInt(work.maxInstruct - m)) + m;  
 m = work.firstInstruct;  
 appendToWorkDetail(memory.executeAnInstruct(work, m),work);  
 }  
 if (!work.finishWork()){  
 appendToWorkDetail(memory.executeAnInstruct(work, m+1), work);  
 }  
 }  
 Double lostPageRate;  
 lostPageRate = ((double)memory.lostPageCount / work.executeCount.longValue());  
 workDetail.appendText("本次作业共执行了" + work.executeCount + "次，缺页" + memory.lostPageCount + "次，缺页率为" + lostPageRate\*100 + "%");  
 memory.lostPageCount = 0;  
 work.executeCount = 0;  
 }  
  
 // 添加作业指令执行详情  
 private void appendToWorkDetail(String detail, Work work){  
 workDetail.appendText("Step" + String.*valueOf*(work.executeCount) + "：");  
 workDetail.appendText(detail.replace("\n", ","));  
 workDetail.appendText("\n");  
 }  
  
 @FXML// 单作业运行，执行320次指令  
 private void executeAllLimited() throws InterruptedException {  
 Work work = new Work();  
 Memory memory = new Memory();  
 while (!work.finishWork() && !(work.executeCount.equals(work.instructNum))){  
 int m = work.firstInstruct;  
 //0到m-1指令中随机执行  
 work.firstInstruct = new Random().nextInt(m);  
 m = work.firstInstruct;  
 appendToWorkDetail(memory.executeAnInstruct(work, m), work);  
 if (!work.finishWork()){  
 appendToWorkDetail(memory.executeAnInstruct(work, m+1), work);  
 }  
 if (!work.finishWork()){  
 //m-1到319指令中随机执行  
 work.firstInstruct = (new Random().nextInt(work.maxInstruct - m)) + m;  
 m = work.firstInstruct;  
 appendToWorkDetail(memory.executeAnInstruct(work, m),work);  
 }  
 if (!work.finishWork()){  
 appendToWorkDetail(memory.executeAnInstruct(work, m+1), work);  
 }  
 }  
 Double lostPageRate;  
 lostPageRate = ((double)memory.lostPageCount / work.executeCount.longValue());  
 workDetail.appendText("本次作业共执行了" + work.executeCount + "次，缺页" + memory.lostPageCount + "次，缺页率为" + lostPageRate\*100 + "%");  
 memory.lostPageCount = 0;  
 work.executeCount = 0;  
 }  
}

# 运行说明

* 双击“页面调度项目.jar”即可打开程序，如图所示：



* 点击“单指令运行”，可显示每一条指令执行时页面交换、内存块分配情况：



* 单作业运行分为：

1. 单作业运行320条指令，此时程序会一直执行指令，直到所有指令都被执行了至少一遍。此时程序执行次数由于随机数指令存在必然会超过320次，因此执行起来会有点慢。文本框显示每一步中指令执行的详细情况以及最后会显示缺页数及缺页率。



1. 单作业运行320次，此时程序会一直执行指令，知道执行次数为320次，此时可能存在有的指令未被执行，有的指令执行了多次。文本框显示每一步中指令执行的详细情况以及最后会显示缺页数及缺页率，同时会显示未执行指令数和作业完成数（完成的指令数占所有指令的比例）。

