项目说明文档

操作系统课程设计

——内存管理

作 者 姓 名： 谢宇翔

学 号： 1951708

指 导 教 师： 张惠娟

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 要求 1](#_Toc9531)

[2 设计 1](#_Toc21345)

[2.1 语言设计 1](#_Toc15726)

[2.2 主要类结构设计 1](#_Toc1275)

[2.2.1 Frame类 1](#_Toc13228)

[2.2.2 Mainstore 类，Block类 2](#_Toc5186)

[2.3 系统设计 2](#_Toc30323)

[2.3.1Frame类 2](#_Toc9783)

[3 实现 3](#_Toc11603)

[3.1 外部循环实现 3](#_Toc18547)

[3.1.1主要代码 3](#_Toc19110)

[3.1.2主要思路 3](#_Toc23173)

[3.2 内部循环实现 4](#_Toc29680)

[3.2.1主要代码 4](#_Toc26444)

[3.2.2主要思路 5](#_Toc15957)

[3.3动画实现 6](#_Toc5646)

# 1 要求

假设初始态下，可用内存空间为640K，并有下列请求序列，请分别用首次适应算法和最佳适应算法进程内存块的分配和回收，并显示出每次分配和回收后的空闲分区链的情况来。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。

置换算法可以选用FIFO或者LRU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。

# 2 设计

## 2.1 语言设计

本次作业使用了java语言，使用java自带的swing库可以很好地生成界面以及设计图形，实现了算法可视化，动态化。

## 2.2 主要类结构设计

### 2.2.1 Frame类

2.2.1.1内置数据

继承了Jframe类，实现了runnable接口，设计整个页面的布局，以此为基础加入其他部件。

int timegap=1000;//Variable for speed  
public int count=0;//count break times  
public int pos=-1;//position of block which will be changed  
public Font font;//font family  
public JButton Begin,Clear,Stop;//3 buttons  
public boolean isclear=false;  
public boolean isrun=false;  
public Mainstore MyMainstore;//Memory  
public Choose MyChoose;//choose algorithm  
public Wait WaitingOrder;//waiting queue  
public Operations Myoperation;//operation state  
JButton SpeedUP,SpeedDown;//speed controlling button

2.2.1.2内置函数

public boolean Find(int order)//look for whether this instruction in my memory,if in,record its position in pos

public void getFI()//find the page which should be changed by FIFO algorithm,record position in pos

public void getLRU()//find the page which should be changed by LRU algorithm,record position in pos

public void run()//run function

public void Herwego()

### 2.2.2 Mainstore 类，Block类

继承了Jpanel类，主存控件，加入Block类，实现了4个Block协同实现。

2.2.2.1 Mainstore内置数据

public Block blocks[]=new Block[5];//5 blocks

2.2.2.2 Block内置数据

public int Order[]={-10,-10,-10,-10,-10,  
 -10,-10,-10,-10,-10};  
public JLabel Memorys[]=new JLabel[10];//10 instructions  
public int VirtualMemory=-1;  
public int times=0;//Variables for LRU  
JLabel Physicsmenory;  
JLabel Virtualmemory;

## 2.3 系统设计

### 2.3.1Frame类

1.首先建立Frame类为整个界面，设置窗口初始化。

2.加入各种部件：内存部件，选择算法部件，指令等待队列部件，操作显示部件

3.加入速度控制按钮，程序控制开始，停止，清零按钮

4.执行run函数

# 3 实现

## 3.1 外部循环实现

### 3.1.1主要代码

@Override  
public void run()//run function  
{  
 while(true)//loop  
 {  
 count=0;  
 if(isrun)  
 {  
 Herwego();  
 Myoperation.Opertion.setText("程序运行完成，共有"+count+"个中断");  
 }  
 if(isclear)  
 {  
 WaitingOrder.CreateOrder();  
 count=0;  
 Myoperation.Opertion.setText("清零成功！");  
 for (int i = 0; i < 4; i++) {  
 WaitingOrder.Memorys[i].setText(""+(WaitingOrder.Myorder[i]));  
 }  
 isclear=false;  
 }  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

### 3.1.2主要思路

通过run函数实现

1. 判断isrun是否为true，如果是，执行内部循环。
2. 判断isclear是否为true,如果是，则清零指令队列，并重新开始

## 3.2 内部循环实现

### 3.2.1主要代码

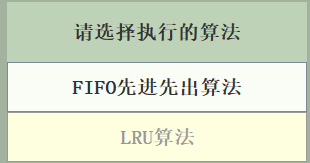
public void Herwego()  
{  
 int i=0;  
 while(isrun&&i<320)  
 {  
 boolean isin=Find(WaitingOrder.Myorder[i]);//look for whether this instruction in my memory,if in,record its position in pos  
 if(!isin)//not in  
 {  
 count++;  
 if(MyChoose.getNowchoose()==1)  
 getFI();//find the page which should be changed by FIFO algorithm,record position in pos  
 else  
 getLRU();//find the page which should be changed by LRU algorithm,record position in pos  
 int OUTpage=MyMainstore.blocks[pos].Order[0]/10+1;  
 int INpage=WaitingOrder.Myorder[i]/10+1;  
 Myoperation.change(OUTpage,INpage);  
 //change Myoperation's show  
 MyMainstore.blocks[pos].change(INpage);  
 //change data of memory  
 }  
 else  
 {  
 MyMainstore.blocks[pos].times=0;  
 Myoperation.change(0,0);  
 }  
 for (int j = 0; j < 10; j++) {  
 if(MyMainstore.blocks[pos].Order[j]==WaitingOrder.Myorder[i])  
 {//show the specific instruction  
 MyMainstore.blocks[pos].Memorys[j].setBackground(Color.*pink*);  
 MyMainstore.blocks[pos].Memorys[j].setOpaque(true);  
 try {// wait for a second  
 Thread.*sleep*(timegap);  
 } catch (InterruptedException e1) {  
 // *TODO Auto-generated catch block* e1.printStackTrace();  
 }  
 MyMainstore.blocks[pos].Memorys[j].setBackground(new Color(250, 253, 246));  
 }  
 }  
 for (int j = 0; j < 4; j++) {  
 MyMainstore.blocks[j].times++;  
 if(i+j+2<320)  
 WaitingOrder.Memorys[j].setText(""+WaitingOrder.Myorder[i+j+2]);  
 else  
 WaitingOrder.Memorys[j].setText("NULL");  
 }  
 i++;  
 }  
 isrun=false;//end of one run  
}

### 3.2.2主要思路

通过HereWogo函数实现

1. 使用i来确定当前要处理的指令
2. 检查isrun是否为true和i是否小于320，都满足则进入循环
3. 从内存中寻找当前指令是否在内存中
4. 如果不在内存里，通过FIFO或LRU算法得出要换出的内存块，更改内存部件的显示和操作显示
5. 如果在内存里，只需更新针对于LRU的变量times，更新操作显示模块
6. 更新内存部件，将具体的指令高亮；更新等待指令部件
7. i++
8. 完成所有的指令调度，isrun设置为false并退出
9. 在外层函数显示中断次数

## 3.3动画实现

1.整体实现：

选择算法模块：

2.运行过程：

1. 运行结束：