**内存管理**

**解决问题：**利用java编写模拟一个320条指令程序的执行过程，内部存储的换页情况的小程序。具体如下：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。

**设计思想和算法：**以操作系统中虚拟内存也面置换算法中的FIFO和LRU为测试样例，通过每次运行计数来模拟页面进入内存的时间和未使用的时间。进入内存的时间会依次递增，每次需要换页时计算出进入内存时间最大的页面换出，未使用的时间在未使用当前页面时递增，使用时清零，每次需要换页时计算出未使用时间最长的页面幻出。在程序中设计5个模块，一个模块物理内存页面，一个模块模拟内存中的页面，一个是打印记录运行过程中是否换页和换页情况的text模块，还有一个显示当前算法和执行的模块,最后一个模块显示缺页率。内存中的页面会记录其装载信息在物理存储中的页面位置。执行时显示物理存储到内存的映射情况。

**数据结构：**封装在相应的图形化类中，该类即可显示，又符合解决问题所要的功能。为简化代码，出去图形化部分函数实现，下面是相关数据结构。

|  |
| --- |
| //外部等待页面  **public** **class** OutPage **extends** JPanel{    **private** **boolean** isCallIn= **false**;  **private** **int** OutPageNum =0;    **public** OutPage(**int** num);  **public** **void** setUnCalledOutPage();    **public** **void** setCalledOutPage();  } |

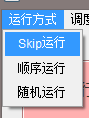
|  |
| --- |
| //内部页面  **public** **class** Page **extends** JPanel// implements Runnable  {    **public** **static** **int** *MAXINSTRUCTIONS*=10;  **private** **int** PagNum=0;  **public** Instruction Instructions[]=**new** Instruction[*MAXINSTRUCTIONS*];//每个页面10条指令  **private** JPanel ShowPage = **new** JPanel();  **private** JLabel PageLabel ;  **public** **int** CurrentPage=0; //当前所驻的外部页面  **public** **int** ExistCount=0; //记录进入内存时间  //记录未访问的次数  **public** **void** ResetCount();    } |

**需要改进的地方：**外存模拟的内存页是用Panel实现，可以考虑复写Label显示背景和数字，优化内存。

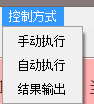
**界面如下**



**操作说明**

可选运行方式：顺序执行，SKIP执行（百分之25前地址和后地址，百分之50顺序执行），随即运行

可选调度算法：FIFO，LRU

可选控制方式： 手动执行，自动执行，结果输出。

手动可单指令运行，点击“开始演示”一次执行一条，自动执行点击“开始演示”后自动运行到320条指令停止，速度中速。结果输出点击“开始演示”后自动快速运行，结束后输出结果“缺页数”和“缺页率”。

算法API：

|  |
| --- |
| //FIFO的调度算法,找到要换出的页面  **public** **int** getOutPageByFIFO()  {  **int** temp=**this**.Pages[0].ExistCount;  **int** page=1;  **for**(**int** i=0;i<**this**.Pages.length;i++)  {  **if**(**this**.Pages[i].ExistCount>temp)  {  temp=**this**.Pages[i].ExistCount;  page=i+1; //页面从1开始计数  }  }  **return** page;  }  //LRU的调度算法，找到要换出的页面  **public** **int** getOutPageByLRU()  {  **int** temp=**this**.Pages[0].UnAllocCount;  **int** page=1;  **for**(**int** i=0;i<**this**.Pages.length;i++)  {  **if**(**this**.Pages[i].UnAllocCount>temp) //UnAllocCout计数和  exitcount不同  {  temp=**this**.Pages[i].UnAllocCount;  page=i+1;  }  }  **return** page;  } |

执行指令产生API

|  |
| --- |
| //随机数的产生函数产生指令  **public** **int** getRandomInstruction()  {  **return** (**int**)((Math.*random*())\*320)+1;    }  //skip指令产生指令  **public** **int** getSkipInstruction()  {  **int** a= (**int**)(Math.*random*()\*100);  //百分之50的几率顺序执行  **if**(a>=0&&a<50&&CurrentInstruction<*MAXINSTRUCTION*)**return**(**this**.CurrentInstruction+1);  **else**{  //百分之25的几率前地址执行  **if**(a>=50&&a<75)  {  **int** b;    b= (**int**)(Math.*random*()\*CurrentInstruction)+1;  **return** b;    }  **else**  {  //百分之25的几率后地址执行  **if**(a>=75&&a<100)  {      **return** (**int**)(((1-(Math.*random*()))\*(*MAXINSTRUCTION*-CurrentInstruction))+CurrentInstruction);  }  **else** **return** 1;  }  }      }    //顺序指令产生  **public** **int** getSequenceInstruction()  {  **if**(CurrentInstruction<*MAXINSTRUCTION*)  **return** CurrentInstruction+1;  **else** **return** 0;  } |

执行的API

|  |
| --- |
| **public** **void** ExecuteInstruction(**int** step)  {  **int** Page = 0;  //所有页面调入时间+1  **for**(**int** i=0;i<**this**.Pages.length;i++)  {  **this**.Pages[i].ExistCount++;  **this**.Pages[i].UnAllocCount++;  }    //指令在当前页  **if**((Page=**this**.isInMemory(CurrentInstruction))!=-1)  {  **this**.Pages[Page-1].UnAllocCount=0; //未访问时间清零    **this**.calledAction(**this**.getPositionOfInstruction(CurrentInstruction),Page);  **this**.ShowStepExecuteText.append(" "+step+"\t" +CurrentInstruction+"\t"+**this**.getPageOfInstruction(CurrentInstruction)  +"\t"+"--"+"\t"+"否"+'\n');  scroll.getVerticalScrollBar().setValue(scroll.getVerticalScrollBar().getMaximum());  }  //不在当前页，则换页  **else**        {    **this**.LackPageNum++;  **int** PageByFIFONum;  **int** PageByLRUNum;  **if**(AllocMethod==FIFO)  {  //得到置换出的页面  PageByFIFONum=getOutPageByFIFO();  **int** LastPage=**this**.Pages[PageByFIFONum-1].CurrentPage;  **if**(**this**.Pages[PageByFIFONum-1].CurrentPage!=0)  **this**.WaitPage[**this**.Pages[PageByFIFONum-1].CurrentPage-1].setUnCalledOutPage();  //改变相关状态    **this**.showPageChanged(CurrentInstruction,PageByFIFONum);      **this**.ShowStepExecuteText.append(" "+step+"\t" +CurrentInstruction+"\t"+**this**.getPageOfInstruction(CurrentInstruction)  +"\t"+LastPage+"\t"+"是"+'\n');  //scroll.getVerticalScrollBar().setValue(scroll.getVerticalScrollBar().getMaximum());  **this**.WaitPage[**this**.Pages[ PageByFIFONum-1].CurrentPage-1].setCalledOutPage();  //换入页刚刚被访问，未访问为0  **this**.Pages[PageByFIFONum-1].UnAllocCount=0;  //换入页的存在次数为0  **this**.Pages[PageByFIFONum-1].ExistCount=0;    **this**.calledAction(**this**.getPositionOfInstruction(CurrentInstruction),PageByFIFONum);    }  **if**(AllocMethod==LRU)  {  PageByLRUNum=getOutPageByLRU();  **int** LastPage=**this**.Pages[PageByLRUNum-1].CurrentPage;  **if**(**this**.Pages[PageByLRUNum-1].CurrentPage!=0)  **this**.WaitPage[**this**.Pages[PageByLRUNum-1].CurrentPage-1].setUnCalledOutPage();    **this**.showPageChanged(CurrentInstruction, PageByLRUNum);  **this**.ShowStepExecuteText.append(" "+step+"\t" +CurrentInstruction+"\t"+**this**.getPageOfInstruction(CurrentInstruction)  +"\t"+LastPage+"\t"+"是"+'\n');  // scroll.getVerticalScrollBar().setValue(scroll.getVerticalScrollBar().getMaximum());    **this**.WaitPage[**this**.Pages[PageByLRUNum-1].CurrentPage-1].setCalledOutPage();  //换入页刚刚被访问，未访问为0  **this**.Pages[PageByLRUNum-1].UnAllocCount=0;  //换入页的存在次数为0  **this**.Pages[PageByLRUNum-1].ExistCount=0;  **this**.calledAction(**this**.getPositionOfInstruction(CurrentInstruction),PageByLRUNum);  }    }      } |