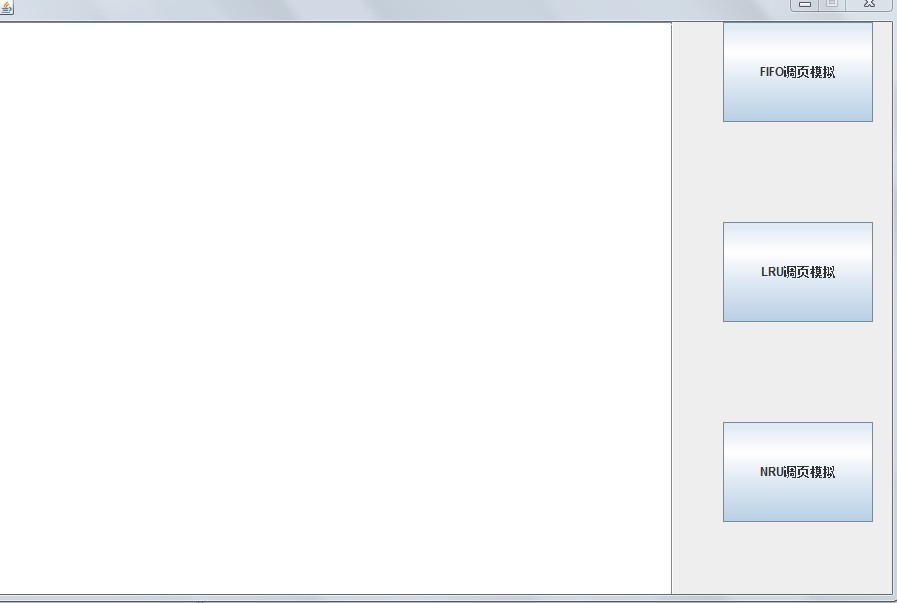
**请求调页设计报告**

1152745 邱峰

1. **程序使用说明**

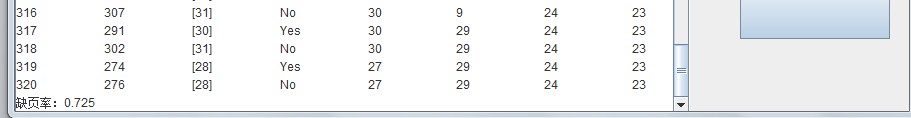
打开程序，得到如下画面



点击FIFO调页得到FIFO（先进先出）调页模拟，点击LRU调页得到（最近最少使用）调页模拟，点击NRU调页得到NRU（轮转）调页模拟。

下图为FIFO调页结果





页面最后有缺页率的计算。

1. **程序设计图**

程序共有2个大类

1. Memory

Memory中包括成员变量

lackCount： 计算却页数

orderNumber： 执行的命令编号

totMemroyPage： 内存的总页数（题目要求为4）

runModel： 运行模式，FIFO || LRU || NRU

point: 在运行NRU方法时的指针

pageMemory[totMemroyPage] 每个内存中的页的情况

PageMemory类

pageAge: 指令进入的时间

runPage： 该内存页中存储的指令中的哪个页

use： NRU调页中的use标记

1. Page

该类为外部命令产生随机指令的类。

主要变量有

totOrder： 指令总数

hashPage[]： 判断随机数是否产生过

1. **实现思想**
2. Page产生随机数

开辟一个hashPage[totOrder/10]大小的int 数组，利用位运算方法判重。每一个hashPage[i]判断10条指令，计算机存储数据是以二进制方法存储，因此，一个int有32位，取它的后面10位即可完成次操作。如，需要判断第15个页中的第8条（从0开始计算）指令是否产生过，只需要判断hashPage[14] & (1<<8)是否为0；如果产生一个17页中的第9条指令，需要将它在以后的指令中删除，只需要hashPage[16]=hashPage[16] | (1<<9)即可。初始时，将每一个hashPage[i]制为0。

在利用Random得到一个随机数randomNumber以后，用这个位运算进行判重。开始的时候，我严格按照课件上的要求去产生随机数，每次产生随机数之前先判断这个数之前（之后）是否有随机数。如果有才产生，没有转换一个方法去取随机数。

但是，后来实践证明这个方法效率不够高，于是，我就修改成了随机10次产生随机数，如果都失败了，则换一个方法取随机数。

1. Memory置换方法

FIFO、LRU、NRU，都需要先检查内存的页中是否已有目前需要的

页号。因此，我先设定一个int check(int page)的函数，函数返回哪一个内存页中的runPage=page，如果都没有，返回-1；

FIFO：

在得到一个指令order以后，order/10为相应的page编号，如果在check(order/10)中，返回的不是-1，则直接退出，代表内存已有次页号。

如果返回的不是-1，在所有的pageMemory[i]中，找到pageAge(也就是最早进入的)最小的一个，替换掉新加入的page，并且把pageMemory[i].pageAge重新更新。并且lackCount++；

LRU：

LRU其实和FIFO差不多，唯一的区别就是在如果check(order/10)在返回不是-1时，我重新更新pageMemory[which].pageAge的值即可。

NRU：

NRU轮转算法是FIFO和LRU的折中，首先，建立一个point指针和pageMemory[i].use分别表示指针轮转到哪个地方，以及该内存中的页是否在最近被使用过。

与上两个方法类似，首先，返回一个which= check(order/10)，如果which!=-1,则pageMemory[which].use=true。

在置换出一个内存时，将pageMemory[i].use=true。

在寻找置换出来的内存时，需要将point移动，移动过程中，如果pageMemory[point].use=true，将其置为false，如果pageMemory[point].use=false，则停止point的移动，并且置换出此时的point。

1. **心得体会**

这个项目比起操作系统的第一次项目简单了不少，算法本身也相对简单，

目的只是为了让我们了解页面调度的各种算法。但是，从我个人的角度来说，我一直觉得操作系统的算法特别简单。这次写项目，却发现FIFO，NRU,LRU三个几乎是相同的，尤其是NRU，后来纠结了好久，问了好多同学，居然得出了3个不同的NRU版本。。。最后是通过问老师才解决的。