操作系统 课程设计任务书

学生姓名		**专业班级**		**学号**	
题　目	请求调页存储管理方式的模拟-1				
指导教师	**林穗**		**题目编号**		2012 秋-09
主要内容	通过对页面、页表、地址转换和页面置换过程的模拟，加深对请求调页系统的原理和实现过程的理解。				
任务要求	1）假设每个页面中可存放 10 条指令，分配给作业的内存块数为 4。 2）用 c 语言模拟一个作业的执行过程，该作业共有 320 条指令，即它的地址空间为 32 页，目前它的所有页都还未调入内存。在模拟过程中，如果所访问的指令已在内存，则显示其物理地址，并转下一条指令。如果所访问的指令还未装入内存，则发生缺页，此时需记录缺页的次数，并将相应页调入内存。如果 4 个内存块均已装入该作业，则需进行页面置换，最后显示其物理地址，并转下一条指令。 3）在所有 320 指令执行完毕后，请计算并显示作业运行过程中发生的缺页率。 4）置换算法：采用先进先出（FIFO）置换算法和最近最久未使用（LRU）算法。				
参考文献	[1] 计算机操作系统， 汤小丹等 ，西安电子科技大学出版社 [2] 操作系统实验指导书，傅秀芬，广东工业大学（自编） [3] 计算机操作系统教程 (第二版)， 张尧学、 史美林，清华大学出版社 [4] 现代操作系统，A.S.Tanenbaum 著，陈向群等译机械工业出版社				
审查意见	**指导教师签字：** **系主任签字：**　　**年　　月　　日**				

说明：本表由指导教师填写，由系主任审核后下达给选题学生，装订在设计（论文）首页

目录

正文

1 设计思想说明

1.1 设计环境

VS 2010

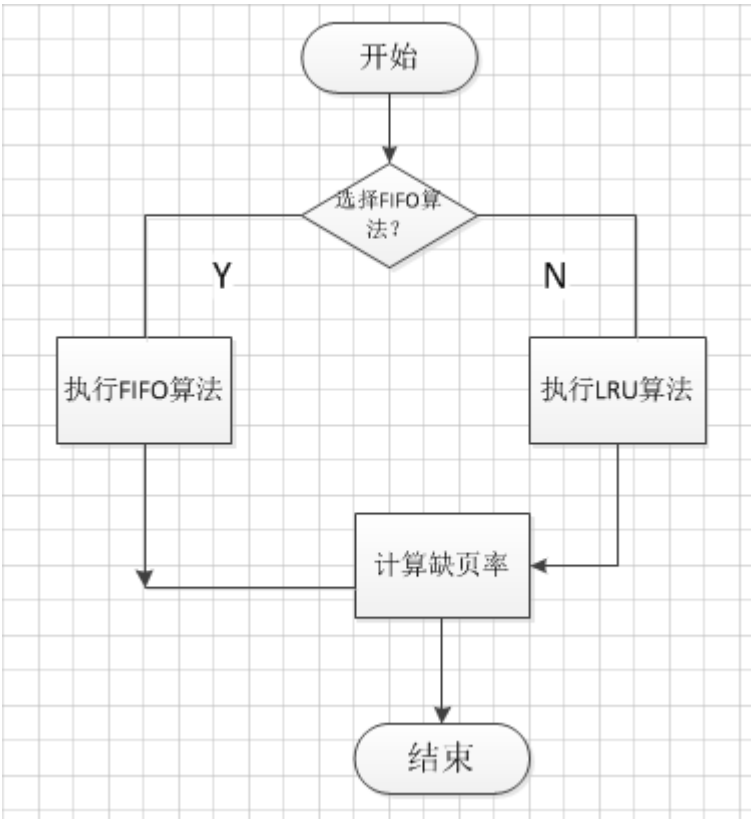
1.2 设计思想

先随机生成 320 条（1-320）的指令串，随机数表示指令的地址，根据指令地址串得到指令所在的页面串，遇到指令不在内存中时，利用 FIFO 算法，LRU 算法，替换掉内存中的页面。

FIFO 页面置换算法，是根据页面调入内存的顺序情况进行决策。先进入内存的页面将会被置换掉，

LRU 页面置换算法，是根据页面调入内存后的使用情况进行决策。由于无法预测各页面将来的使用情况，只能利用“最近的过去”作为“最近的将来”的近似，因此 **LRU** 算法是选择最近最久未使用的页面予以淘汰。

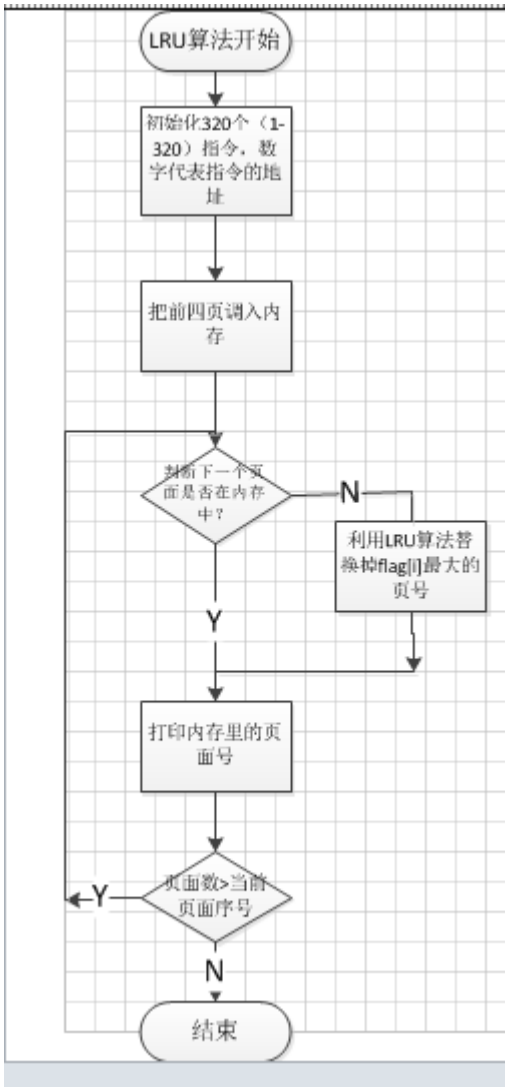
2 系统结构

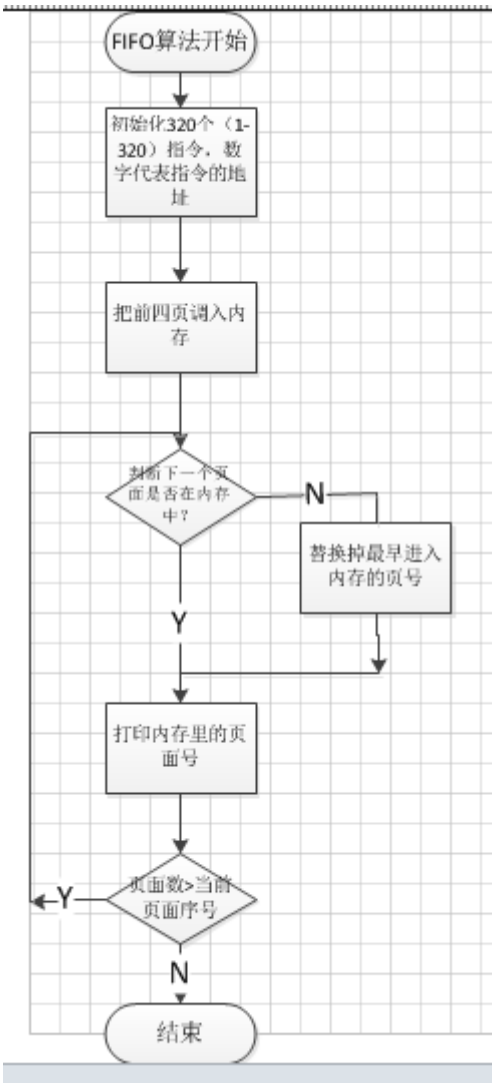


3 数据结构的说明

```
static int memery[mSIZE] = {0};   //初始化物理块数组
static int process[pSIZE] = {0};  //初始化页面串
static int order[pSIZE] = {0};    //初始化指令串
static int address[pSIZE] ={0};   //指令的页内地址
```

4 算法流程图


LRU算法开始
初始化320个（1-320）指令，数字代表指令的地址
把前四页调入内存
判断下一个页面是否在内存中？
N
利用LRU算法替换掉flag[i]最大的页号
Y
打印内存里的页面号
页面数>当前页面序号
Y
N
结束

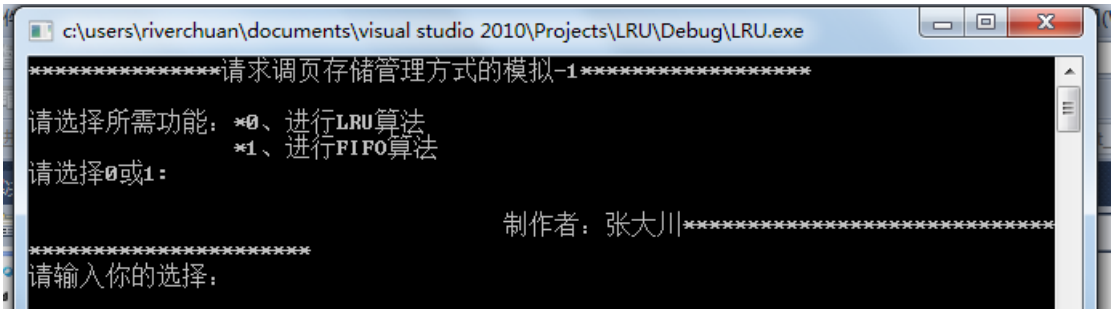


5 主要函数列表

函数原型	功能	入口、出口参数说明
Void bulid();	初始化 320 个（1-320）的指令	
Void LRU();	进行 LRU 算法	
Void FIFO();	进行 FIFO 算法	
Void init();	启动	

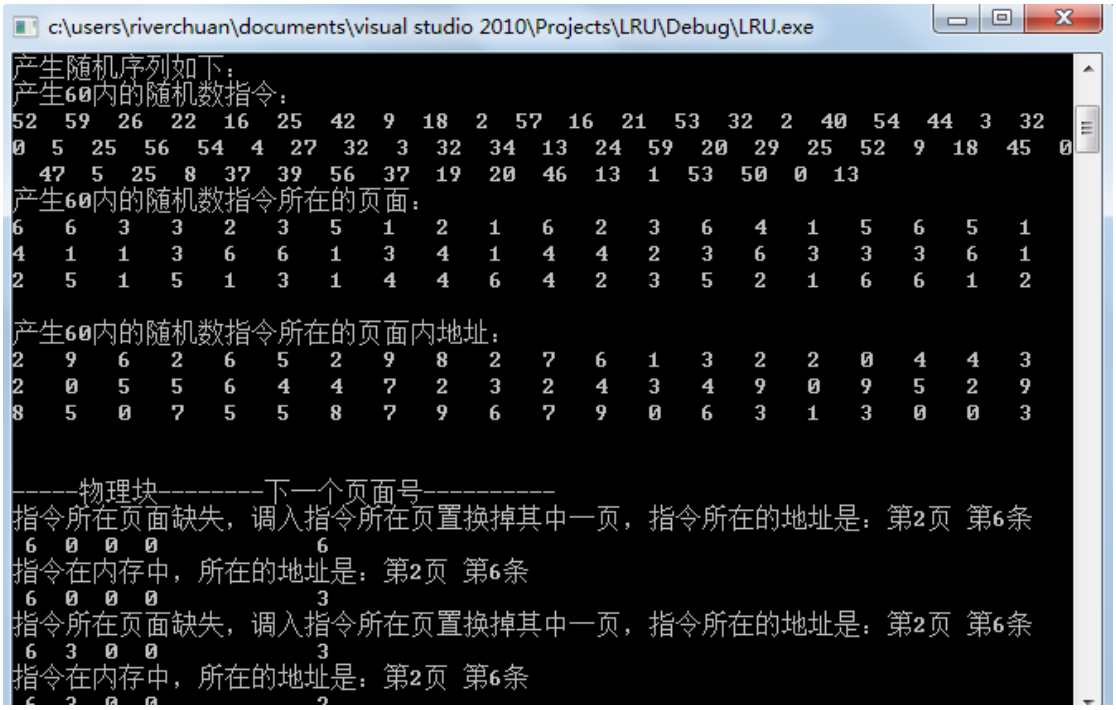
6 测试与分析

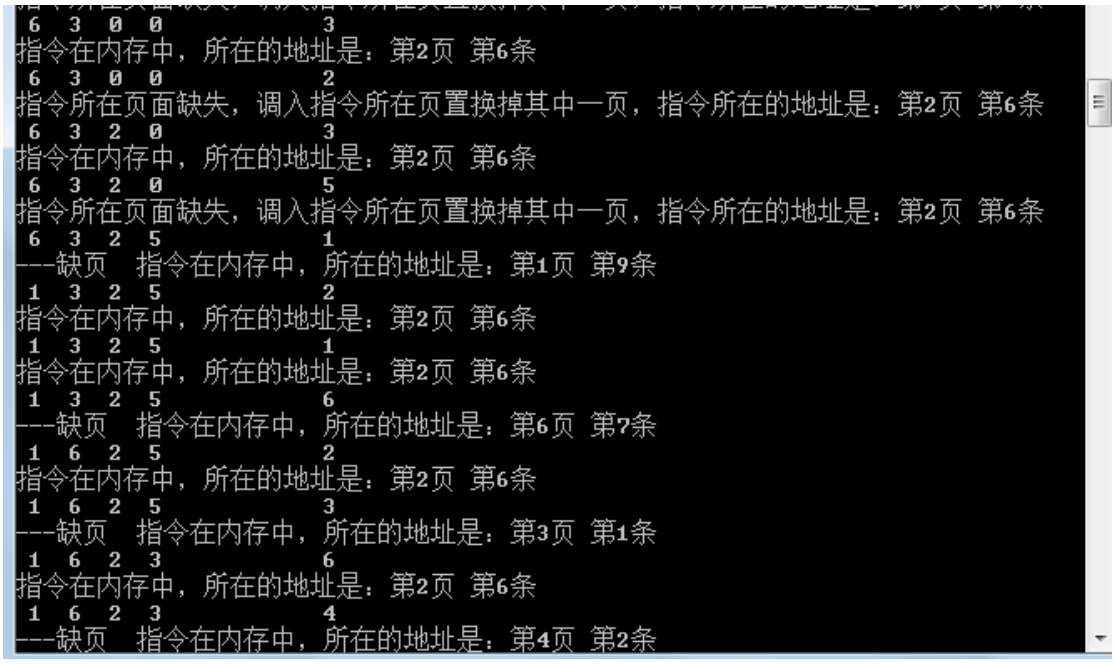
6.1 用户菜单：

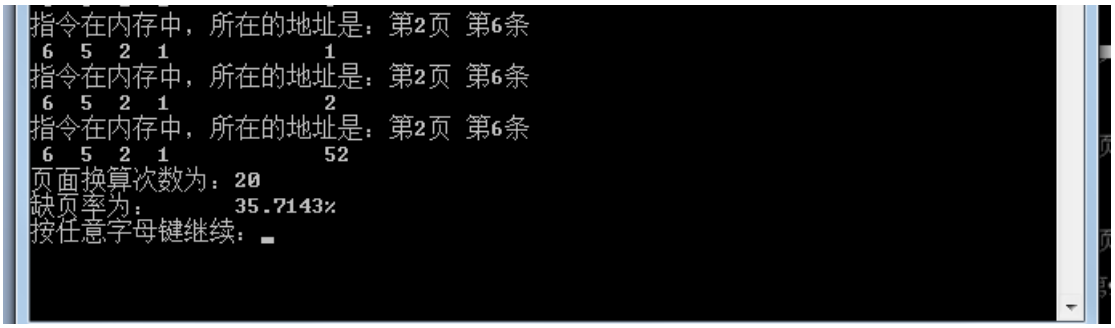


6.2 测试 LRU 算法

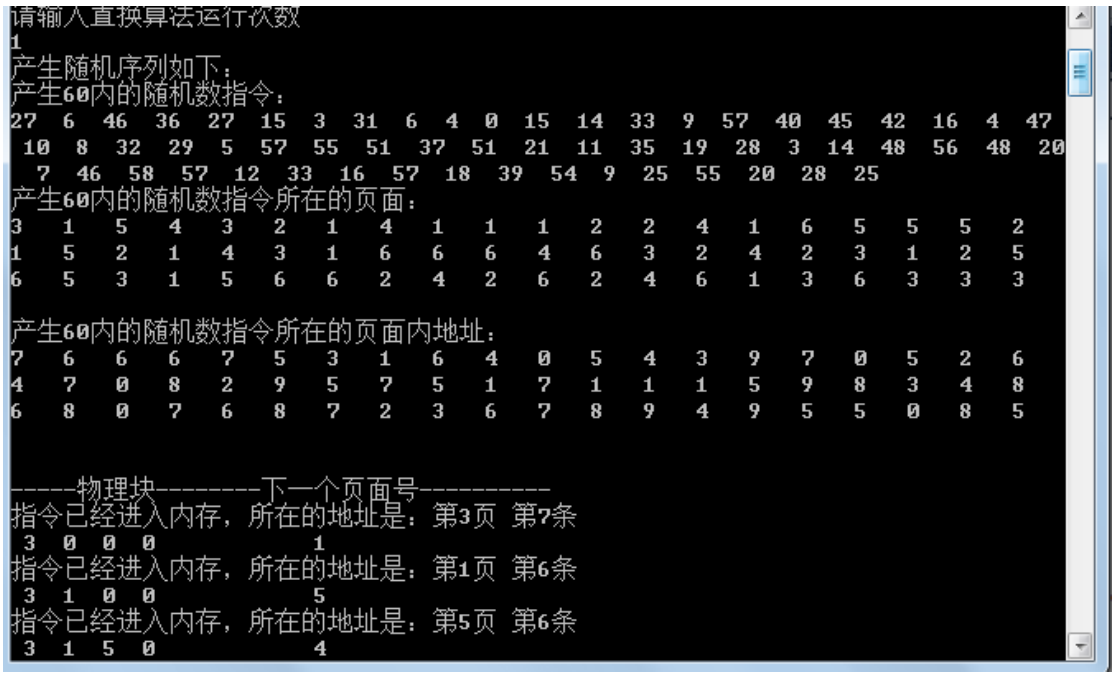
由于采用 320 条指令，数据太长了，不好显示，便用用 60 条指令进行测试。

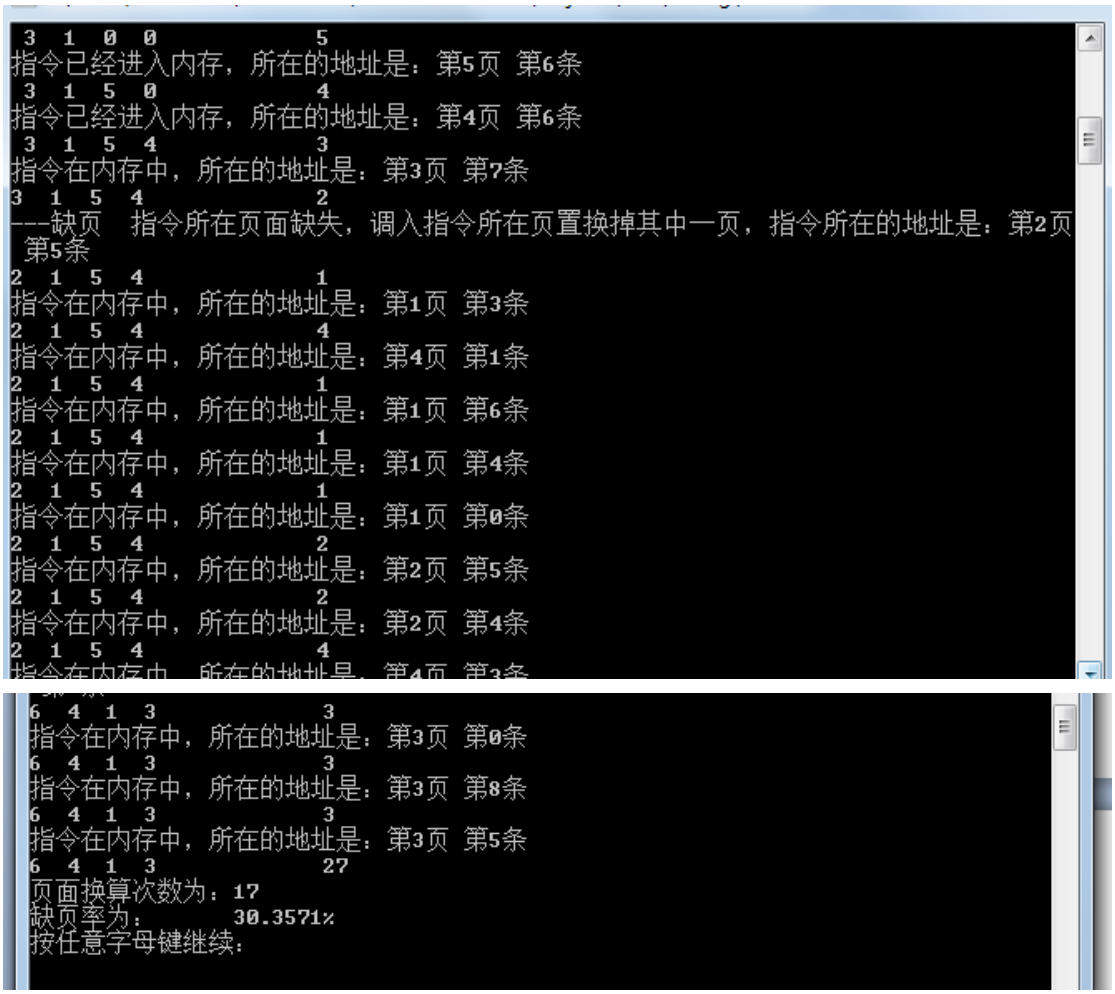



6 3 0 0 3
指令在内存中，所在的地址是：第2页 第6条
6 3 0 0 2
指令所在页面缺失，调入指令所在页置换掉其中一页，指令所在的地址是：第2页 第6条
6 3 2 0 3
指令在内存中，所在的地址是：第2页 第6条
6 3 2 0 5
指令所在页面缺失，调入指令所在页置换掉其中一页，指令所在的地址是：第2页 第6条
6 3 2 5 1
---缺页 指令在内存中，所在的地址是：第1页 第9条
1 3 2 5 2
指令在内存中，所在的地址是：第2页 第6条
1 3 2 5 1
指令在内存中，所在的地址是：第2页 第6条
1 3 2 5 6
---缺页 指令在内存中，所在的地址是：第6页 第7条
1 6 2 5 2
指令在内存中，所在的地址是：第2页 第6条
1 6 2 5 3
---缺页 指令在内存中，所在的地址是：第3页 第1条
1 6 2 3 6
指令在内存中，所在的地址是：第2页 第6条
1 6 2 3 4
---缺页 指令在内存中，所在的地址是：第4页 第2条


指令在内存中，所在的地址是：第2页 第6条
6 5 2 1 1
指令在内存中，所在的地址是：第2页 第6条
6 5 2 1 2
指令在内存中，所在的地址是：第2页 第6条
6 5 2 1 52
页面换算次数为：20
缺页率为： 35.7143%
按任意字母键继续：

6.3、FIFO 算法测试


请输入置换算法运行次数
1
产生随机序列如下：
产生60内的随机数指令：
27 6 46 36 27 15 3 31 6 4 0 15 14 33 9 57 40 45 42 16 4 47
10 8 32 29 5 57 55 51 37 51 21 11 35 19 28 3 14 48 56 48 20
7 46 58 57 12 33 16 57 18 39 54 9 25 55 20 28 25
产生60内的随机数指令所在的页面：
3 1 5 4 3 2 1 4 1 1 1 2 2 4 1 6 5 5 5 2
1 5 2 1 4 3 1 6 6 6 4 6 3 2 4 2 3 1 2 5
6 5 3 1 5 6 6 2 4 2 6 2 4 6 1 3 6 3 3 3
产生60内的随机数指令所在的页面内地址：
7 6 6 6 7 5 3 1 6 4 0 5 4 3 9 7 0 5 2 6
4 7 0 8 2 9 5 7 5 1 7 1 1 1 5 9 8 3 4 8
6 8 0 7 6 8 7 2 3 6 7 8 9 4 9 5 5 0 8 5
-----物理块--------下一个页面号----------
指令已经进入内存，所在的地址是：第3页 第7条
3 0 0 0 1
指令已经进入内存，所在的地址是：第1页 第6条
3 1 0 0 5
指令已经进入内存，所在的地址是：第5页 第6条
3 1 5 0 4



7 用户使用说明

按照程序提示使用。

附录 ：源代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define random(x) (rand()%x)

#define mSIZE 4    //物理块数目
```

```
#define pSIZE 60   //进程数目（页面）

static int memery[mSIZE] = {0};   //初始化物理块数组
static int process[pSIZE] = {0};  //初始化页面串
static int order[pSIZE] = {0};    //初始化指令串
static int address[pSIZE] ={0};   //指令的页内地址

void FIFO();
void build();             //生成一个随机数序列
void LRU();               //最近最久未使用(LRU)置换算法
void frame();
void init();
//用户界面
void frame(){

        printf("***************请求调页存储管理方式的模拟-1*****************\n\n");
        printf("请选择所需功能：*0、进行LRU算法\n");
        printf("                *1、进行FIFO算法\n");
        printf("请选择0或1: \n\n");
        printf("                                    制作者：张大川");
        printf("*****************************************************\n");

}
//初始化指令串
void build(){
     int i;
     srand((int)time(0));
     printf("产生%d内的随机数指令：\n",pSIZE);
     for(i = 0; i < pSIZE; i++){      //产生320条（1-320）指令串,用random函数，存入order
数组
         order[i] = random(pSIZE);
         printf("%d  ",order[i]);
     }
     printf("\n");
     printf("产生%d内的随机数指令所在的页面：\n",pSIZE);
    for(i=0; i<pSIZE; i++){
            process[i] = order[i]/10+1;//将产生的320条指令串的页号存入process数组中
             address[i] = order[i]%10; //将产生的320条指令串的页内号存入address数组中
            printf("%d   ",process[i]);
    }
     printf("\n");
     printf("产生%d内的随机数指令所在的页面内地址：\n",pSIZE); //打印随机数的页内地址
     for(i=0; i<pSIZE; i++){
        printf("%d   ",address[i]);
     }
    printf("\n");
```

```
     printf("\n");
}
//FIFO算法
void FIFO(){

    int i,j;
    int k;
//  int flag = 0;
    float count = 0; //记下缺页次数
    int c[pSIZE] = {0}; //是否存在内存的标志
    printf("-----物理块--------下一个页面号----------\n");
    for( i = 0; i<mSIZE; i++){   //产生320条（0-320）之间的条指令随机数
        memery[i] = 0;
    }
    for( i = 0; i<mSIZE; i++){   //把前四条指令串的页号调入内存的空闲空间里

        printf("指令已经进入内存，所在的地址是：第%d页 第%d条\n",process[i],address[i]);
        memery[i] = process[i] ;
        for ( j= 0;j<mSIZE; j++){ //每次把内存中的指令串打印出来

            printf(" %d ",memery[j]);
        }
        printf("          %d ",process[i+1]);
        printf("\n");
    }

    k = mSIZE;
    for(j=mSIZE;j<pSIZE;j++){

        for(i=0;i<mSIZE;i++)
            if(process[j]==memery[i]){//如果内存中存在下一个被调入指令的页号，则把c[j]
= 1;
             c[j] = 1;
             break;
            }
        if(c[j] == 1){ // 如果内存中存在下一个被调入的指令的页号，则打印内存中的页号。
               printf("指令在内存中，所在的地址是：第%d页  第%d条
\n",process[j],address[j]);
               for(i=0;i<mSIZE;i++)
                     printf("%d  ",memery[i]);

        printf("          %d ",process[j+1]);
//      printf("\n");

//      printf("\n");
```

```
        }
        if(c[j] == 0){
            printf("---缺页  ");//如果内存中缺页，则根据fifo算法，把最先进来的页面替换。
            printf("指令所在页面缺失，调入指令所在页置换掉其中一页，指令所在的地址是：
第%d页 第%d条\n",process[j],address[j]);
            memery[k%mSIZE]=process[j];
            k++;
            count++;
            for(i=0;i<mSIZE;i++)
                printf("%d  ",memery[i]);
            printf("           %d ",process[j+1]);
        }
        printf("\n");
    }
    printf("页面换算次数为：%.0f\n",count);
    float w;
    w=count/(pSIZE-mSIZE);
    printf("缺页率为：       %.4f%%\n",w*100);

}


//LRU算法
void LRU(){                    //LRU过程

    int flag[mSIZE] = {0};    //在物理块中逗留时间标志，初始化为0，
    int i = 0, j = 0;         //辅助变量
    int m = -1, n = -1;       //m空块号，n相同进程块号（如果存在）
    int max = -1,maxflag = 0; //标志最大逗留时间值
    float count = 0;            //计数器，计算缺页次数
    int x=0;
    for(i = 0; i<mSIZE ; i++){      //先把内存空间设为0，表示内存空闲
        memery[i] = 0;
    }
    printf("-----物理块--------下一个页面号----------\n");
    for(i = 0; i<pSIZE; i++){

        for(j=0; j<mSIZE; j++){      //找第一个空闲的物理块（若存在）

            if(memery[j] == 0){
                m = j;      //第一个空物理块序号存入m
                break;      //找到则跳出，每次只需要找到第一个
            }
        }

        for(j = 0; j < mSIZE; j++){   //找与新进页面号相同的物理块（若存在）
```

```
            if(memery[j] == process[i]){
                n = j;              //找到则把该物理块号存入n

            }
        }

        for(j = 0; j < mSIZE;j++){          //找时间标志flag最大的物理块号

            if(flag[j]>maxflag){
                maxflag = flag[j];
                max = j;                    //此时max为最大值
            }
        }

        if(n == -1){                //物理块中不存在与新页面相同页面号

            if(m != -1){            //存在一个空闲物理块

                memery[m] = process[i];  //装入空物理块
                  printf("指令所在页面缺失，调入指令所在页置换掉其中一页，指令所在的地址是：第%d页 第%d条\n",process[j],address[j]);
                flag[m] = 0;                //时间标志0
                for(j = 0;j <= m; j++){

                    flag[j]++;              //时间标志全部加一
                }
                m = -1;     //该空闲物理块已经使用，不再为空
            }
            else{             //不存在空闲物理块
                  printf("---缺页  ");
                 printf("指令在内存中，所在的地址是：第%d页  第%d条\n",process[i],address[i]);
                memery[max] = process[i];     //新页面替换内存中 flag值最大（即最近最久未使用）的物理块
                flag[max] = 0;                //新替换的页面flag值标志为0
                for(j = 0;j < mSIZE; j++){

                    flag[j]++;              //时间标志全部加一
                }
                max = -1;                   //最大时间标志恢复初始值，下一次重新查找

                maxflag = 0;
                count++;                            //计数器标志替换次数
                x=1;                    //缺页标志，方便显示
            }
```

```
            }
            else{                          //物理块中存在与新页面相同的页面号

                memery[n] = process[i];
                  printf(" 指 令 在 内 存 中 ， 所 在 的 地 址 是 ： 第 %d 页   第 %d 条
\n",process[j],address[j]);
                flag[n] = 0;
                if(m != -1){        //若存在空闲物理块

                     flag[m] = 0;  //时间标志0
                }
                for(j = 0;j < mSIZE; j++){

                      flag[j]++;   //时间标志全部加一
                }
                max = -1;            //所有标志恢复初始值，下一次重新查找
                maxflag = 0;
                n = -1;
            }

            for(j = 0 ;j < mSIZE; j++){      //显示该次过程用内存物理块内的内容

                printf(" %d ",memery[j]);
            }
             printf("           %d ",process[i+1]);
             if(x==1){
                   x=0;
             }
            printf("\n");
      }
      printf("页面换算次数为：%.0f\n",count);
       float w;
       w=count/(pSIZE-mSIZE);
       printf("缺页率为：       %.4f%%\n",w*100);
}
//启动项
void init(){
     frame();
     int a=0;
     int m;
     printf("请输入你的选择：");
     scanf("%d",&m);
     printf("请输入置换算法运行次数\n");//运行次数
     scanf("%d",&a);
     switch(m){
```

```
    case 0:
        for(int i = 0; i<a; i++){

            printf("产生随机序列如下：\n");
            build();    //产生320条（1-320）之间的条指令随机数
            LRU();      //LRU算法
            printf("按任意字母键继续：");
            getchar();
            getchar();
        }
       getchar();
        break;
    case 1:
        for(int i = 0; i<a; i++){

            printf("产生随机序列如下：\n");
            build();   //产生320条（0-320）之间的条指令随机数
            FIFO();    //FIFO算法
            printf("按任意字母键继续：");
            getchar();
            getchar();
        }
        getchar();
        break;
    }
}

void main(){

    init(); //初始化
}
```