**一． 设计目的：**

**加深对请求页式存储管理实现原理的理解，掌握页面置换算法。**

**二．设计内容:**

**设计一个程序，有一个虚拟存储区和内存工作区，实现下述三种算法中的任意两种，计算访问命中率**

**（命中率=1-页面失效次数/页地址流长度）。附加要求：能够显示页面置换过程。算法包括：先进先出的**

**算法（FIFO）、最少使用算法（LRU）、最近未使用算法（NUR）该系统页地址流长度为320，页面失效**

**次数为每次访问相应指令时，该指令对应的页不在内存的次数。   程序首先用srand()和rand()函数分别进**

**行初始化、随机数定义和产生指令序列，然后将指令序列变换成相应的页地址流，并针对不同的算法计算**

**出相应的命中率。通过随机数产生一个指令序列。共320条指令，指令的地址按下述原则生成：**

**（1）50%的指令是顺序执行的。**

**（2）25%的指令是均匀分布在前地址部分。**

**（3）25%的指令是均匀分布在后地址部分。**

**具体的实施方法如下：**

**在【0，319】的指令地址之间随机选取一起点m。**

**顺序执行一条指令，即执行地址为m+1的指令。**

**在前地址【0,m+1】中随机选取一条指令并执行，该指令的地址为m’。**

**顺序执行一条指令，其地址为m’+1。**

**在后地址【m’+2,319】中随机选取一条指令并执行。**

**重复步骤（1）-（5），直到320次指令。**

**将指令序列变换为页地址流。**

**设：**

**页面大小为1KB。**

**用户内存容量4页到32页。（动态变化）**

**用户虚存容量为32KB。（32页）**

**在用户虚存中，按每K存放10条指令虚存地址，即320条指令在虚存中的存放方式为：**

**第0条～9条指令为第0页（对应虚存地址为【0，9】）。**

**第10条～19条指令为第1页（对应虚存地址为【10，19】）。**

**……**

**第310条～319条指令为第31页（对应虚拟地址为【310，319】）。**

**按以上方式，用户指令可组成32页。**

**计算每种算法在不同内存容量下的命中率。**

**分页管理是这样的，将内存和作业分成大小相等的页块，作业中的每个页块在不需要被使用时存放在外存中（虚拟存储区），**

**当需要使用时将其从外存调入内存页块中；根据题意，在外存中的页面顺序存储着指令，需要执行哪一条指令就找到其对应的**

**页面，若页面已在内存则无需再操作，否则此页面缺页，需要将其调入内存，当内存块未满时，只需要直接将其插入内存块中**

**若内存块已满，则需要调用先进先出算法淘汰出一个页面（将其调回外存），再将此页面调入。**

**首先通过 rand 函数和 srand 函数产生320条指令，计算每条指令对应的页面很简单，只需要将指令/10即可；得到页地址流后**

**（页地址流存放在数组中），从头到尾访问一遍页地址流，每访问一个页面就判断其是否已经在内存中，在无需操作，不在则**

**将其（使用FIFO）调入内存。**

**FIFO：在页面缺页并且内存块不足时，只需要将内存块中原先的页面依次淘汰即可。**

**假设页地址流为：1　　2　　3　　5　　4　　3　　8　　11　　12　　6（内存块大小为 3 ）**

**1　　2　　3　　进入内存**

**5　　2　　3　　1被调出**

**5　　4　　3　　2被调出**

**5　　4　　3　　命中**

**5　　4　　8　　3被调出（形成一个循环）**

**11　  4　　8　　5被调出**

**11　 12　  8　　4被调出**

**11　 12　  6　　8被调出（形成一个循环）**

**只要为内存块编号（不是为页面编号），用一个变量（初值为1）作为指针，此变量指向的内存块，就是被FIFO选中需要调出的**

**内存块，调出后变量+1，当变量大于内存块数时，再将其置为1（循环）**

**三．代码：**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

#define max\_page 10

int Page[320]={0};

int Page\_flu[320]={0};

int count=0;

double lack\_page=0;

int count\_page=max\_page;

int circle=1;

struct Memo{

int num;

int a;

struct Memo \*next;

};

int Judge\_Page(int value){

return value/10;

}

int scan\_queen(struct Memo \*hear,int value){

struct Memo \*move;

move=hear->next;

while(move!=NULL){

if(move->a==value){

return 1;

}

move=move->next;

}

return 0;

}

void print(struct Memo \*hear){

struct Memo \*move;

move=hear->next;

while(move!=NULL){

printf("%d ",move->a);

move=move->next;

}

printf("\n");

}

void insert(struct Memo \*hear,int value,int ZL){

if(count\_page>=1){

struct Memo \*move;

move=hear->next;

while(move->a!=-1){

move=move->next;

}

move->a=value;

count\_page--;

printf("页面 %d 被调入————对应指令为: %d \n",value,ZL);

}

else{

struct Memo \*move;

move=hear->next;

while(move->num!=circle){

move=move->next;

}

printf("页面 %d 被调出,页面 %d 被调入————指令为: %d \n",move->a,value,ZL);

move->a=value;

circle++;

if(circle==max\_page+1){

circle=1;

}

}

print(hear);

}

void FIFO(struct Memo \*hear){

int i=0;

for(i=0;i<=319;i++){

if( scan\_queen(hear,Page\_flu[i])==0){

lack\_page++;

insert(hear,Page\_flu[i],Page[i]); }

else{

printf("指令 %d 对应页面 %d 已在内存\n",Page[i],Page\_flu[i]);

}

}

}

void Pro\_Page(){

int m=0;

m=rand()%320;

Page[count]=m;

count++;

if(count==320){

return;

}

int m\_=0;

m\_=rand()%(m+1);

Page[count]=m\_;

count++;

if(count==320){

return;

}

Page[count]=m\_+1;

count++;

if(count==320){

return;

}

int m\_\_=0;

m\_\_=(m\_+2)+rand()%( 319-(m\_+2)+1 );

Page[count]=m\_\_;

count++;

if(count==320){

return;

}

Pro\_Page();

}

void Flu(){

int i=0;

for(i=0;i<=319;i++){

Page\_flu[i]=Judge\_Page( Page[i] );

}

}

int main(){

struct Memo Stu[max\_page+1];

struct Memo \*hear;

hear=&Stu[0];

int i=0;

for(i=0;i<=max\_page;i++){

if(i==max\_page){

Stu[i].a=-1;

Stu[i].next=NULL;

Stu[i].num=i;

break;

}

Stu[i].next=&Stu[i+1];

Stu[i].a=-1;

Stu[i].num=i;

}

srand(time(0));

Pro\_Page();

Flu();

/\*

printf("页地址流：\n");

for(i=0;i<=319;i++){ //输出页地址流

printf("%d ",Page[i]);

if(i%3==0 && i!=0){

printf("\n");

}

}

printf("\n");

\*/

FIFO(hear);

printf("缺页次数为： %0.0lf\n",lack\_page);

printf("命中率为：%lf\n",1-lack\_page/320);

return 0;

}

**四．运行结果：**

