实验七、页面置换算法

**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。

**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。

**算法（FIFO）：**

代码：

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

#define max\_page 10 //内存页面数

int Page[320]={0}; //虚拟存储区,存储320条指令,32个页面

int Page\_flu[320]={0}; //存储320个页地址流

int count=0; //计算随机产生的指令条数

double lack\_page=0; //记录缺页数

int count\_page=max\_page; //计算队列空页面个数

int circle=1; //在队列中循环指向被调出的进程

struct Memo{ //用结构体存储内存页面块

int num; //给每个页面编号,方便将其从队列中找到并调出

int a;

struct Memo \*next;

};

int Judge\_Page(int value){ //输入指令,返回指令对面的页面号

return value/10;

}

int scan\_queen(struct Memo \*hear,int value){ //value代表页面号,扫描队列,缺页返回0,否则返回1

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){ //输出内存页面

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){ //将页面value调入内存,ZL为对应指令

if(count\_page>=1){ //内存页面空间充足

struct Memo \*move;

move=hear->next;

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

move=move->next;

}

move->a=value; //将页面调入

count\_page--;

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

}

else{ //内存空间不足，调出最先进入的页面后，将页面value后调入

struct Memo \*move;

move=hear->next;

while(move->num!=circle){ //circle存储的是需要调出的页面编号

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; //将页面调入

circle++;

if(circle==max\_page+1){ //当circle>max\_page+1时,最先进入的页面为队列首页面

circle=1;

}

}

print(hear); //调入后输出内存队列

}

void FIFO(struct Memo \*hear){

int i=0;

for(i=0;i<=319;i++){ //循环扫描页面

if( scan\_queen(hear,Page\_flu[i])==0){ //判断是否缺页

lack\_page++;

insert(hear,Page\_flu[i],Page[i]); //缺页将页面调入内存

}

else{ //不缺页

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

}

//不缺页无需操作

}

}

void Pro\_Page(){ //形成页地址流函数

int m=0; //在[0,319]的指令地址之间随机选取一起点m

m=rand()%320;

Page[count]=m;

count++;

if(count==320){

return;

}

int m\_=0; //在前地址[0,m+1]中随机选取一条指令并执行

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 ); //在后地址[m\_+2,319]的指令地址之间随机选取一条指令并执行

Page[count]=m\_\_;

count++;

if(count==320){

return;

}

Pro\_Page();

}

void Flu(){ //将指令转换为页地址流

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++){ //形成内存页面队列

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)); //放在Pro\_Page函数外面

Pro\_Page(); //形成页地址流

Flu(); //形成页地址流

/\*

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);

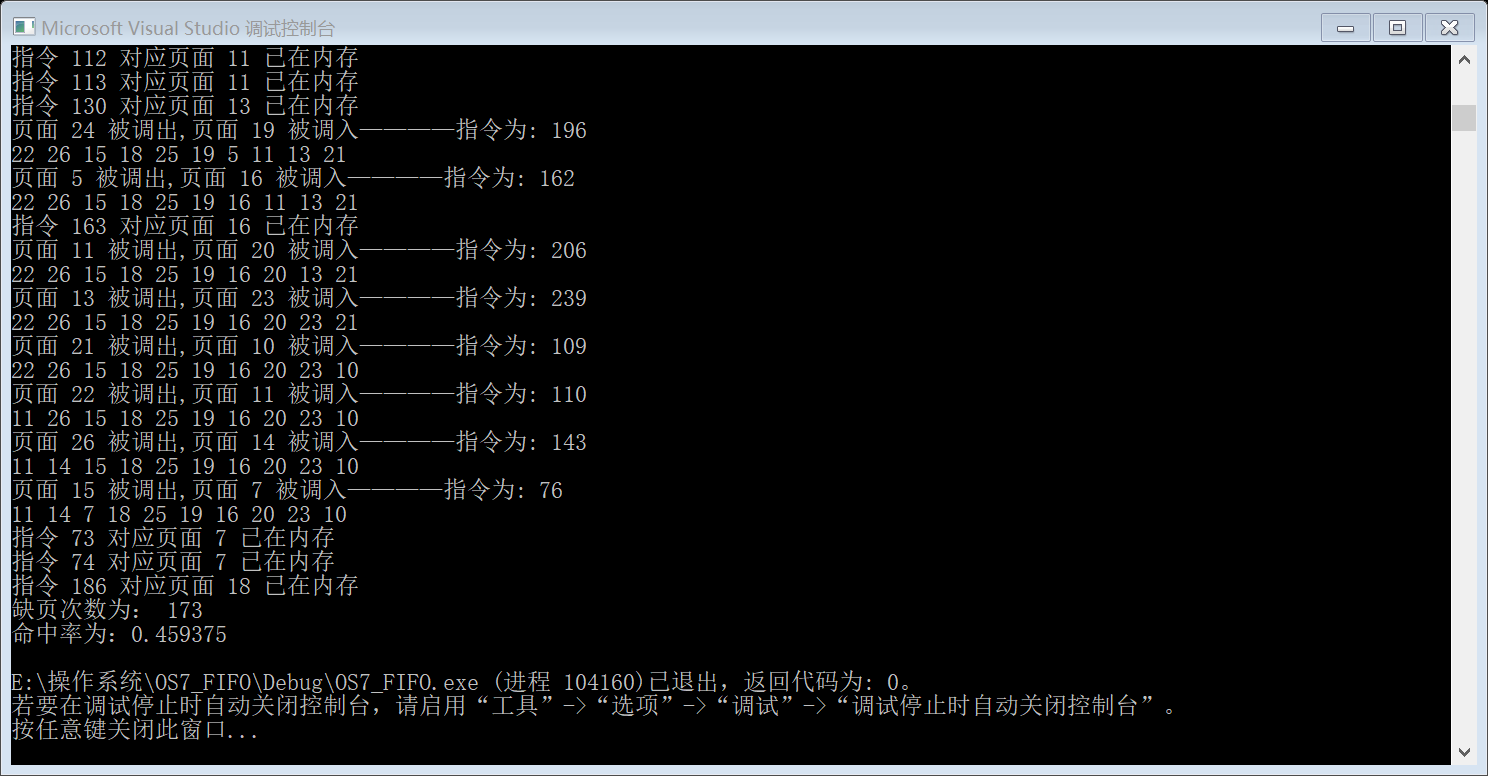
printf("命中率为：%lf\n",1-lack\_page/320);

return 0;

}

实验结果：实验报告上给的代码，可以运行并得出正确结果。

运行结果：



在上述代码的基础上设计另一种算法。

设计LRU算法

将结构体改为如下：

struct Memo { //用结构体存储内存页面块

int num; //给每个页面编号,方便将其从队列中找到并调出

int a;

int time;

struct Memo \*next;

};

将插入函数改为如下：

void insert(struct Memo \*hear, int value, int ZL) { //将页面value调入内存,ZL为对应指令

if (count\_page >= 1) { //内存页面空间充足

struct Memo \*move;

move = hear->next;

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

move = move->next;

}

move->a = value; //将页面调入

move->time = 0;

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;

int timemax = 0;

int flag;

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

if (move->time >= timemax)

{

timemax = move->time;

flag = move->num;

}

move = hear->next;

while (move->num != flag)

{

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; //将页面调入

move->time = 0;

move = hear->next;

while (move->a != -1)

{

move->time++;

move = move->next;

}

}

print(hear); //调入后输出内存队列

}

运行结果：

