## 实验7 页面置换算法的模拟实现

### 一、实验目的

1、熟悉基本分页存储管理。

2、掌握FIFO页面置换算法。

3、掌握LRU页面置换算法。

### 二、实验理论基础及教材对应关系

1、操作系统中内存管理。

2、基本分页内存、分段内存管理。

3、常用页面置换算法的实现。

### 三、实验内容与步骤

1、定义相关数据

#define InitPysiBlocks 4

#define MaxPages 16：

unsigned int PysicalBlocks[InitPysiBlocks] = { 0 };

unsigned int PageSequence[30] = { 1,2,3,6,4,7,3,2,1,4,7,5,6,5,2,1};

2、按照教材中FIFO、LRU算法描述进行算法设计

unsigned FIFO(unsigned \*py,unsigned \*pg)

unsigned LRU(unsigned \*py,unsigned \*pg)

3、查看运行结果是否与手工计算一致。

### 四、实验材料的提交与成绩评定

本实验的实验报告一份（电子版一份，格式参考学校实验报告）

代码：

#include<iostream>

#define InitPysiBlocks 4

#define MaxPages 16

using namespace std;

unsigned int PysicalBlocks[InitPysiBlocks] = { 0 };

unsigned int PageSequence[30] = { 1,2,3,6,4,7,3,2,1,4,7,5,6,5,2,1 };

//FIFO算法

unsigned FIFO(unsigned \*py, unsigned \*pg)

{

int i,j;

//初始化填满数据

for (i = 0; i < InitPysiBlocks; i++)

{

py[i] = pg[i];

}

cout << "FIFO置换过程如下：" << endl;

cout << py[0] << " " << py[1] << " " << py[2] << " " << py[3] << endl;

//判断新获取数据进来的时间

int time[4] = { 3,2,1,0 };

//开始检测

for (i = 4; i < MaxPages; i++)

{

for (j = 0; j < InitPysiBlocks; j++)

{

//如果新获取的已存在，直接输出

if (py[0] == pg[i] || py[1] == pg[i] || py[2] == pg[i] || py[3] == pg[i])

{

cout << py[0] << " " << py[1] << " " << py[2] << " " << py[3] << endl;

break;

}

else

{

for (int k = 0; k < 4; k++)

{

if (time[k] == 3)

{

time[k] = 0;

py[k] = pg[i];

}

else

{

time[k]++;

}

}

cout << py[0] << " " << py[1] << " " << py[2] << " " << py[3] << endl;

break;

}

}

}

cout << "置换结束" << endl;

return 0;

}

//LRU算法

//LRU辅助函数:返回除某一排下标后数组中最大元素下标

int Max(unsigned \*py, int te)

{

int i,max = 0;

int index = -1;

for (i = 0; i < InitPysiBlocks; i++)

{

if (i == te)

{

continue;

}

else

{

if (py[i] > max)

{

max = py[i];

index = i;

}

else

{

continue;

}

}

}

return index;

}

unsigned LRU(unsigned \*py, unsigned \*pg)

{

int i,j,k;

//初始化填满数据

for (i = 0; i < InitPysiBlocks; i++)

{

py[i] = pg[i];

}

cout << "LRU置换过程如下：" << endl;

cout << py[0] << " " << py[1] << " " << py[2] << " " << py[3] << endl;

//判断新获取数据进来的时间

unsigned time[4] = { 3,2,1,0 };

//统计命中次数

int count[4] = { 0 };

//开始检测

for (i = InitPysiBlocks; i < 16; i++)

{

for (j = 0; j < InitPysiBlocks; j++)

{

//如果新获取的已存在，直接输出,并将对应的命中变为1

if (py[0] == pg[i] || py[1] == pg[i] || py[2] == pg[i] || py[3] == pg[i])

{

cout << py[0] << " " << py[1] << " " << py[2] << " " << py[3] << endl;

if (py[0] == pg[i])

{

count[0] = 1;

}

else if (py[1] == pg[i])

{

count[1] = 1;

}

else if (py[2] == pg[i])

{

count[2] = 1;

}

else if (py[3] == pg[i])

{

count[3] = 1;

}

else

{

count[4] = 0;

}

break;

}

else

{

if (count[0] == 1)

{

//返回数组中最大值

int temp = Max(time, 0);

py[temp] = pg[i];

for (k = 0; k < InitPysiBlocks; k++)

{

time[k]++;

}

time[temp] = 0;

count[0] = 0;

}

else if (count[1] == 1)

{

//返回数组中最大值

int temp = Max(time, 1);

py[temp] = pg[i];

for (k = 0; k < InitPysiBlocks; k++)

{

time[k]++;

}

time[temp] = 0;

count[1] = 0;

}

else if (count[2] == 1)

{

//返回数组中最大值

int temp = Max(time, 2);

py[temp] = pg[i];

for (k = 0; k < InitPysiBlocks; k++)

{

time[k]++;

}

time[temp] = 0;

count[2] = 0;

}

else if (count[3] == 1)

{

//返回数组中最大值

int temp = Max(time, 3);

py[temp] = pg[i];

for (k = 0; k < InitPysiBlocks; k++)

{

time[k]++;

}

time[temp] = 0;

count[3] = 0;

}

else

{

//返回数组中最大值

int temp = Max(time, -1);

py[temp] = pg[i];

for (k = 0; k < InitPysiBlocks; k++)

{

time[k]++;

}

time[temp] = 0;

}

cout << py[0] << " " << py[1] << " " << py[2] << " " << py[3] << endl;

break;

}

}

}

cout << "置换结束" << endl;

return 0;

}

int main()

{

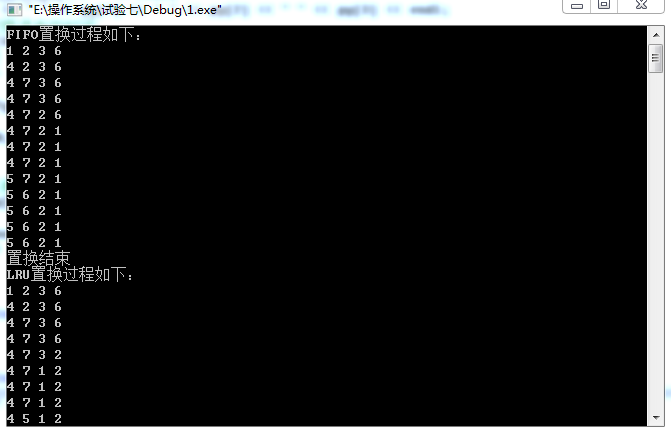
FIFO(PysicalBlocks, PageSequence);

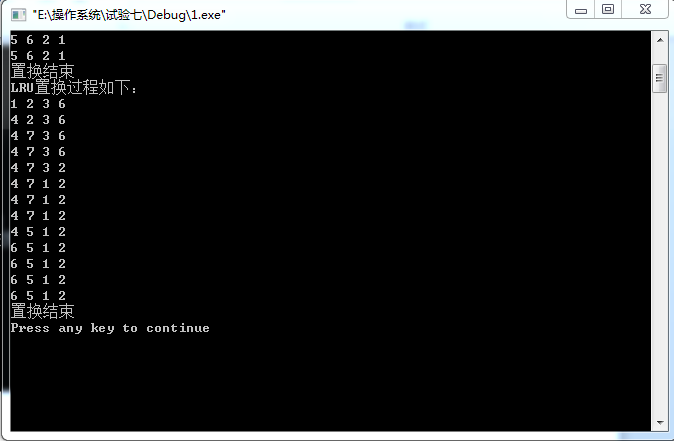
LRU(PysicalBlocks, PageSequence);

return 0;

}

结果：





结论：运行结果正确，与人工计算的结果一样。