

操作系统

实 验 2 报 告

学院： 计算机科学学院

专业： 网络工程1701

姓名： 胡玉琛41709040102

姓名： 何一鸣41709040107

1. 1. **实验题目**

虚拟内存页面置换算法

1. **实验目的**

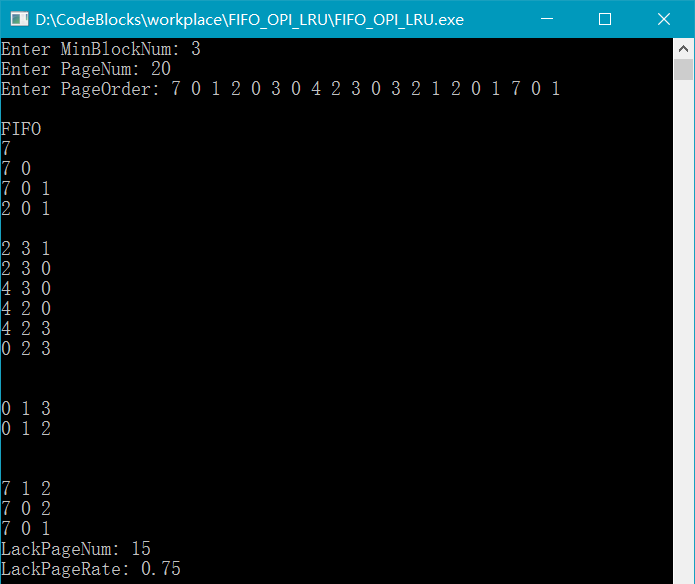
通过这次实验，加深对虚拟内存页面置换概念的理解，进一步掌握先进先出FIFO，最佳置换OPI和最近最久未使用LRU页面置换算法的实现方法。

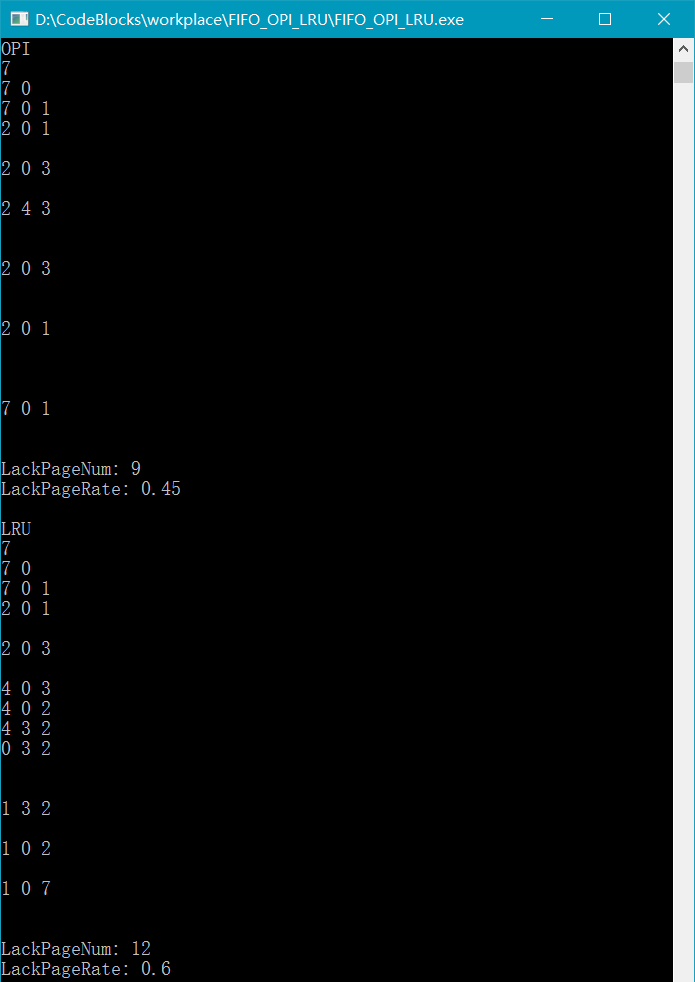
1. **实验环境**

Windows10，Microsoft Visual Studio 2013

1. **实验内容**
2. 利用先进先出FIFO，最佳置换OPI和最近最久未使用LRU三种页面置换算法模拟页面访问过程。
3. 模拟三种算法的页面置换过程，给出每个页面访问时的内存分配情况。
4. 输入：最小物理块数m，页面个数n,页面访问序列P1，…,Pn算法选择1-FIFO。2-OPI,3-LRU。

**五、实验结果**

****

****

1. **附录**

源程序：

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

#define MaxNumber 100

int MinBlockNum,PageNum,LackNum,LackPageNum; //物理块数，页面个数，缺页次数，缺页数

double LackPageRate; //缺页率

int PageOrder[MaxNumber]; //页面序列

int PageDisCount[MaxNumber]; //当前内存距离下一次出现的距离

int LRUtime[MaxNumber]; //存储队列中各个页面最近使用情况

int VirtualQueue[MaxNumber]; //虚拟队列

void Enter(); //输入物理块数、页面号顺序

void initial(); //模拟物理块顺序被填满，初始化物理块

void FIFO(); //先进先出FIFO

void OPI(); //最佳置换OPI

void LRU(); //最近最久未使用LRU

void display(); //显示执行结果

int main(){

Enter();

// initial();

FIFO();

OPI();

LRU();

return 0;

}

void Enter(){

int i;

cout<<"Enter MinBlockNum: ";

cin>>MinBlockNum;//输入最小物理块数

cout<<"Enter PageNum: ";

cin>>PageNum;//输入页面个数

//输入页面序列

cout<<"Enter PageOrder: ";

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

cin>>PageOrder[i];

}

}

void initial(){

int i,j,k;

bool isInQueue;

LackPageNum = MinBlockNum;//缺页数=物理块数+缺页次数

LackPageRate = 0.0;

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

PageDisCount[i] = 0; //初始化距离

VirtualQueue[i] = -1; //初始化队列

}

for (i = 0;i<MinBlockNum;i++){//初始化物理块

isInQueue = false;

LRUtime[i] = 0;

for (j = 0;j<MinBlockNum;j++){

if (VirtualQueue[j] == PageOrder[i])//如果页面在队列中

isInQueue = true;

}

if (!isInQueue){ //当有新的进程进入到队列时，便计算其对应的距离

VirtualQueue[i] = PageOrder[i];//小于物理块数时，页面顺序进入队列

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

LRUtime[k]++; //之前的页面对应的时间+1

}

display();

}

else

LRUtime[i] = 0; //重新更新为0，表示最近刚刚使用

}

}

void display(){

int i;

for (i = 0;i<MinBlockNum && VirtualQueue[i]>=0;i++)

cout<<VirtualQueue[i]<<" ";

cout<<endl;

}

void FIFO(){

cout<<endl;

int i,k;

cout<<"FIFO"<<endl;

initial();

bool isInQueue;

int point = 0; //指向队列中最老的

//从物理块数+1的地方开始继续分配内存

for (i = MinBlockNum;i<PageNum;i++){

isInQueue = false;

for (k = 0;k<MinBlockNum;k++){

if (VirtualQueue[k] == PageOrder[i]){ //页面在队列中

isInQueue = true;

}

}

if (!isInQueue){ //如果页面不在队列中，则进行相应的处理

LackPageNum++; //缺页数加1

VirtualQueue[point] = PageOrder[i];//页面替换队列中最老的

display();//输出物理块状态

point++;//后移

if (point == MinBlockNum)//当指向队尾时后，重新指向队首

point = 0;

}

else

cout<<endl;

}

LackPageRate = (LackPageNum \* 1.0)/PageNum;

cout<<"LackPageNum: "<<LackPageNum<<endl;

cout<<"LackPageRate: "<<LackPageRate<<endl;

}

void OPI(){

cout<<endl;

int i,k,m,s,t;

cout<<"OPI"<<endl;

initial();

bool isInQueue;

int distance; //表示队列每个值距离下一次访问的距离

int point; //指向最长时间未被访问的下标

for(i = MinBlockNum;i<PageNum;i++){

isInQueue = false;

for (k = 0;k<MinBlockNum;k++){

if (VirtualQueue[k] == PageOrder[i]){ //页面在队列中

isInQueue = true;

}

}

if (!isInQueue){

LackPageNum++;

//计算当前队列每一页对应的下一次出现的距离

for (s = 0;s < MinBlockNum;s++){

distance = 1;

for (t = i;t<PageNum;t++){ //向后找离他最远的

if (VirtualQueue[s] != PageOrder[t])

distance++;

else

break;

}

PageDisCount[s] = distance;//当前内存距离下一次出现的距离

}

//向后比较队列内最长时间不被访问的并淘汰，置换页面

point = 0;

for (m = 1;m < MinBlockNum;m++){

if (PageDisCount[point] < PageDisCount[m])

point = m;

}

VirtualQueue[point] = PageOrder[i];

display();

}

else

cout<<endl;

}

LackPageRate = (LackPageNum\*1.0)/PageNum;

cout<<"LackPageNum: "<<LackPageNum<<endl;

cout<<"LackPageRate: "<<LackPageRate<<endl;

}

void LRU(){

cout<<endl;

int i,j,s,k;

cout<<"LRU"<<endl;

initial();

int point;//指向最长时间未被访问的下标

bool isInQueue;

for(i = MinBlockNum;i<PageNum;i++){

isInQueue = false;

for (k = 0;k<MinBlockNum;k++){

if (VirtualQueue[k] == PageOrder[i]) //页面在队列中

isInQueue = true;

}

if (!isInQueue){

LackPageNum++;

//向前比较队列内最长时间不被访问的并淘汰，置换页面

point = 0;

for (j = 1;j<MinBlockNum;j++){

if (LRUtime[point]<LRUtime[j])

point = j;

}

for (s = 0;s<MinBlockNum;s++){//其余页面对应的时间要+1{

if (VirtualQueue[s] != VirtualQueue[point])

LRUtime[s]++;

}

VirtualQueue[point] = PageOrder[i];

LRUtime[point] = 0;

display();

}

else{ //负责更新当前对应页面的时间

for (s = 0;s<MinBlockNum;s++){//其余页面对应的时间要+1

if (VirtualQueue[s] != PageOrder[i])

LRUtime[s]++;

else

LRUtime[s] = 0;

}

cout<<endl;

}

}

LackPageRate = (LackPageNum\*1.0)/PageNum;

cout<<"LackPageNum: "<<LackPageNum<<endl;

cout<<"LackPageRate: "<<LackPageRate<<endl;

}