

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 操作系统实验 | | | | |
| 实验编号： | 实验六 | | | | |
| 实验名称： | 页面置换算法 | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 18级计算机科学与技术创新班 | | | |
| 实验日期： | 2020.6.4 | | | | |
| 实验室： |  | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 一、实验目的

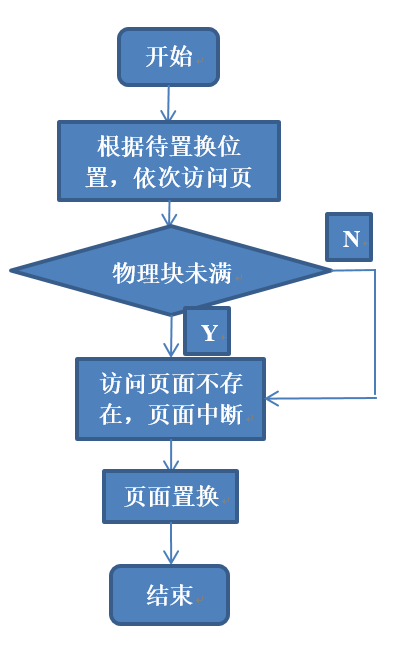
# 理解页面置换算法,实现先进先出（FIFO）置换算法和最久未使用（LRU）置换算法。给出任意的输入流，计算置换次数。

# 二、实验要求

# 给出任意的输入流，计算置换次数。物理块大小可定制，0－9数字可任意排序的数字流。

# 三、设计

【定义所有抽象数据类型，自定义函数的功能详细描述（伪代码表示），以及主程序的流程图。】



#include<iostream>

#include<stdlib.h>

#include<string.h>

#include<iomanip>

using namespace std;

/\*

测试数据

-------------------------------------------

3

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 #

-------------------------------------------

\*/

int GetDistance(int currentPageID,int page);//按照页面编号获取在物理块内的页面最久未使用的时间（哪个距离当前pageID最远）

class BLOCK{

public:

int blockNum; //物理块总数

int pageNum; //物理块中的页面数量

int \*pageID; //页面号（大小为blockNum）

int \*stayTime; //页面在物理块中的停留时间（与物理块ID对应）

BLOCK(int num)

{

int i;

pageNum=0;

blockNum=num;

pageID=new int[num];

stayTime=new int[num];

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

{

pageID[i]=-1; //初始化每个物理块中没有放置，页面号表示为-1

stayTime[i]=0; //初始化停留时间为0

}

}

void Init()

{

int i;

int num=blockNum;

pageNum=0;

pageID=new int[num];

stayTime=new int[num];

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

{

pageID[i]=-1; //初始化每个物理块中没有放置，页面号表示为-1

stayTime[i]=0; //初始化停留时间为0

}

}

void ShowPage()

{

int i;

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

cout<<"Page["<<i<<"]: "<<pageID[i]<<endl;

}

void ShowStayTime()

{

int i;

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

cout<<"Stay["<<i<<"]: "<<stayTime[i]<<endl;

}

int GetLongestStay() //获取在物理块中停留时间最长的页面所在物理块号

{

int i;

int max\_pos=0;

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

if(stayTime[max\_pos]<stayTime[i])

max\_pos=i;

return max\_pos;

}

int GetRencentNotUse(int currentPageID) //获取在物理块中最近最久未使用的页面编号

{

//默认currentPageID一定大于等于BLOCKNUM

int i;

int DestID=0;

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

{

if(GetDistance(currentPageID,pageID[i])>GetDistance(currentPageID,pageID[DestID]))

DestID=i;

}

return DestID;

}

}; //物理块数据结构定义

//-----------------------全局变量-------------------------

int BLOCKNUM; //物理块数

int \*PVS; //PageVisitSequence页面访问序列

int PVS\_NUM; //页面访问序列长度

int \*\*replaceTable; //页面置换表格（维度：BLOCKNUM\*PVS\_NUM）

int \*replaceArray; //页面置换标志数组（大小为访问页面的次数，存储每次访问是否进行页面置换）

int \*lackArray; //缺页中断标志数组（大小为访问页面的次数，存储每次访问是否存在缺页中断）

//-----------------------函数声明-------------------------

void showMenu(); //菜单显示

void InputAndInit(); //数据输入和变量初始化

void ReplaceFIFO(BLOCK block); //FIFO页面置换算法

int FindPage(int pageID,BLOCK block); //页面查找（按照页面编号在物理块中查找页面是否存在）

void ShowReplaceTable(); //置换表格输出

void ReplaceLRU(BLOCK block); //LRU页面置换算法

void InfoDisplay(); //初始化信息显示

int GetReplaceTimes(); //获取页面置换总次数

int GetLackTimes(); //获取缺页中断总次数

//-----------------------函数定义-------------------------

int main()

{

int select;

int i;

InputAndInit();

BLOCK block(BLOCKNUM); //定义物理块（注意该语句必须在变量和数据初始化后，否则BLOCKNUM未知）

cout<<"信息初始化完成！"<<endl<<endl;

showMenu();

cout<<"请输入要进行的操作："<<endl;

cin>>select;

while(1)

{

switch(select)

{

case 1:

InfoDisplay();

cout<<endl;

break;

case 2:

ReplaceFIFO(block);

cout<<"|=================> FIFO页面调度算法正在执行......"<<endl;

ShowReplaceTable();

cout<<"页面置换次数为："<<GetReplaceTimes()<<endl;

cout<<"缺页中断次数为："<<GetLackTimes()<<endl;

cout<<endl;

cout<<endl;

break;

case 3:

ReplaceLRU(block);

cout<<"|=================> LRU页面调度算法正在执行......"<<endl;

ShowReplaceTable();

cout<<"页面置换次数为："<<GetReplaceTimes()<<endl;

cout<<"缺页中断次数为："<<GetLackTimes()<<endl;

cout<<endl;

cout<<endl;

break;

case 0:

return 0;

default:

cout<<"输入有误，请重新输入..."<<endl;

cout<<endl;

break;

}

//------防止页面置换和缺页次数计算错误-------------

for(i=0;i<PVS\_NUM;i++)

{

replaceArray[i]=0; //页面置换标志数组初始化为0

lackArray[i]=0; //缺页中断标志数组初始化为0

}

showMenu();

cout<<"请输入要进行的操作（退出请输入0）..."<<endl;

cin>>select;

}

delete[] PVS;

delete[] replaceArray;

delete[] lackArray;

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

delete[] replaceTable[i];

delete[] replaceTable;

return 0;

}

//----------------------FIFO页面置换算法--------------------------

void ReplaceFIFO(BLOCK block) //FIFO页面置换算法

{

int i,j;

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

for(j=0;j<PVS\_NUM;j++)

replaceTable[i][j]=0;

block.Init();

int replacePosition; //待置换位置

for(i=0;i<PVS\_NUM;i++) //依次对页面访问序列的每一个页面PVS[i]进行操作

{

for(j=0;j<block.pageNum;j++)

block.stayTime[j]++; //每循环一次，物理块（0~pageNum）停留时间自增

if(block.pageNum<block.blockNum)

{

if(!FindPage(PVS[i],block)) //若页面PVS[i]不存在

{

lackArray[i]=1; //由于访问页面不存在造成页面中断

block.pageID[block.pageNum]=PVS[i];

block.pageNum++;

}

}

else //FIFO算法（置换停留时间最长的页面所在物理块位置）

{

// 若页面已存在的情况（上述三条语句应该是页面不存在的情况，应加上if(页面在物理块中不存在)的判断）

if(!FindPage(PVS[i],block)) //若页面PVS[i]不存在

{

replaceArray[i]=1; //由于访问页面不存在且无空闲物理块造成页面置换

lackArray[i]=1; //由于访问页面不存在造成页面中断

replacePosition=block.GetLongestStay();

block.pageID[replacePosition]=PVS[i]; //选择停留时间最长的页面置换

block.stayTime[replacePosition]=0; //置换后，该页面所在物理位置停留时间清零

}

}

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

replaceTable[j][i]=block.pageID[j]; //将访问一次页面后的结果存入数组中（replaceTable）

}

}

int FindPage(int pageID,BLOCK block) //页面查找（按照页面编号在以存放页面的物理块（长度为pageNum）中查找页面是否存在）

{

int i=0;

for(i=0;i<block.pageNum;i++)

if(block.pageID[i]==pageID)

break;

return !(i==block.pageNum); //若页面存在，则返回1，否则返回0

}

//----------------------LRU页面置换算法--------------------------

void ReplaceLRU(BLOCK block) //LRU页面置换算法

{

int i,j;

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

for(j=0;j<PVS\_NUM;j++)

replaceTable[i][j]=0;

block.Init();

int replacePosition; //待置换位置

for(i=0;i<PVS\_NUM;i++) //依次对页面访问序列的每一个页面PVS[i]进行操作

{

for(j=0;j<block.pageNum;j++)

block.stayTime[j]++; //每循环一次，物理块（0~pageNum）停留时间自增

if(block.pageNum<block.blockNum)

{

if(!FindPage(PVS[i],block)) //若页面PVS[i]不存在

{

lackArray[i]=1; //由于访问页面不存在造成页面中断

block.pageID[block.pageNum]=PVS[i];

block.pageNum++;

}

}

else //FIFO算法（置换停留时间最长的页面所在物理块位置）

{

// 若页面已存在的情况（上述三条语句应该是页面不存在的情况，应加上if(页面在物理块中不存在)的判断）

if(!FindPage(PVS[i],block)) //若页面PVS[i]不存在

{

replaceArray[i]=1; //由于访问页面不存在且无空闲物理块造成页面置换

lackArray[i]=1; //由于访问页面不存在造成页面中断

replacePosition=block.GetRencentNotUse(i);

block.pageID[replacePosition]=PVS[i]; //选择停留时间最长的页面置换

block.stayTime[replacePosition]=0; //置换后，该页面所在物理位置停留时间清零

}

}

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

replaceTable[j][i]=block.pageID[j]; //将访问一次页面后的结果存入数组中（replaceTable）

}

}

//----------------------OTHRES--------------------------

void showMenu() //菜单显示

{

cout<<"\t\t|----------------------------MENU-------------------------------|"<<endl;

cout<<"\t\t|\t1. 初始化信息显示 |"<<endl;

cout<<"\t\t|\t2. FIFO页面置换算法 |"<<endl;

cout<<"\t\t|\t3. LRU页面置换算法 |"<<endl;

cout<<"\t\t|\t0. 退出程序 |"<<endl;

cout<<"\t\t|---------------------------------------------------------------|"<<endl;

}

void InputAndInit() //数据输入和变量初始化

{

int i=0;

int j=0;

char PVS\_char[100];

cout<<"请输入物理块数..."<<endl;

cin>>BLOCKNUM;

cout<<"请依次输入页面访问序列（数字0~9，每个数字之间有空格，以#结束）..."<<endl;

cin>>PVS\_char[i];

getchar();

while(PVS\_char[i]!='#')

{

i++;

cin>>PVS\_char[i];

getchar();

}

PVS\_char[i]='\0'; //设置字符串终止符

PVS\_NUM=i;

PVS=new int[PVS\_NUM];

for(i=0;i<PVS\_NUM;i++)

PVS[i]=PVS\_char[i]-'0'; //字符转数字（页面号）

replaceArray=new int[PVS\_NUM];

lackArray=new int[PVS\_NUM];

for(i=0;i<PVS\_NUM;i++)

{

replaceArray[i]=0; //页面置换标志数组初始化为0

lackArray[i]=0; //缺页中断标志数组初始化为0

}

replaceTable=new int\*[BLOCKNUM]; //页面置换表初始化

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

replaceTable[i]=new int[PVS\_NUM];

}

void ShowReplaceTable() //置换表格输出

{

int i,j;

cout<<"页面置换过程如下图所示"<<endl<<endl;

cout<<setw(PVS\_NUM\*3+14)<<setfill('-')<<"-"<<endl;

cout.fill(' ');

cout<<"页面访问序列 ";

for(i=0;i<PVS\_NUM;i++)

cout<<" "<<setw(2)<<PVS[i];

cout<<endl;

cout<<setw(PVS\_NUM\*3+14)<<setfill('-')<<"-"<<endl;

cout.fill(' ');

cout<<"页面置换过程 ";

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

{

if(i>0)

cout<<"\t ";

for(j=0;j<PVS\_NUM;j++)

{

if(replaceTable[i][j]!=-1)

cout<<"|"<<setw(2)<<replaceTable[i][j];

else cout<<"|"<<setw(2)<<" "; //-1时代表该物理块无页面，不输出

}

cout<<"|"<<endl;

}

cout<<setw(PVS\_NUM\*3+14)<<setfill('-')<<"-"<<endl;

cout<<"页面置换标志 ";

cout.fill(' ');

for(i=0;i<PVS\_NUM;i++)

cout<<" "<<setw(2)<<replaceArray[i];

cout<<endl;

cout<<setw(PVS\_NUM\*3+14)<<setfill('-')<<"-"<<endl;

cout<<"页面中断标志 ";

cout.fill(' ');

for(i=0;i<PVS\_NUM;i++)

cout<<" "<<setw(2)<<lackArray[i];

cout<<endl;

cout<<setw(PVS\_NUM\*3+14)<<setfill('-')<<"-"<<endl<<endl;

}

int GetDistance(int currentPageID,int page) //按照页面编号获取在物理块内的页面最久未使用的时间（哪个距离当前pageID最远）

{

int distance=0;

int i;

for(i=currentPageID-1;i>=0;i--)

if(PVS[i]!=page)

distance++;

else break;

return distance;

}

void InfoDisplay() //初始化信息显示

{

int i;

cout<<"本页面置换模拟算法中: "<<endl;

cout<<"物理块数为： "<<BLOCKNUM<<endl;

cout<<"页面访问序列为：";

for(i=0;i<PVS\_NUM;i++)

cout<<PVS[i]<<" ";

cout<<endl;

}

int GetReplaceTimes() //获取页面置换总次数

{

int sum=0;

int i;

for(i=0;i<PVS\_NUM;i++)

sum+=replaceArray[i];

return sum;

}

int GetLackTimes() //获取页面中断总次数

{

int sum=0;

int i;

for(i=0;i<PVS\_NUM;i++)

sum+=lackArray[i];

return sum;

}

# 四、调试分析和思考

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

FIFO算法和LRU算法的比较：FIFO置换算法的性能之所以较差，是因为它所依据的条件是各个页面调入内存的时间，而页面调入的先后并不能反映页面的使用情况。LRU的页面置换算法是根据页面调入内存后的使用情况作出决策的。

FIFO算法中，第一组数据有3个物理块，当进程第一次访问页面2时，将把第7页换出，因为它是最先被调入内存的；在第一次访问页面3时，又将把第0页换出，因为它在现有的2、0、1三个页面中是最老的页。

LRU算法中，当进程第一次对页面2进行访问时，由于页面7是最近最久未被访问的，故将它置换出去。当进程第一次对页面3进行访问时，第1页成为最近最久未使用的页，将它换出。

# 五、测试数据与结果

【列出你的测试结果，明确输入和输出数据。测试数据应该完整和严格，可以直接贴结果图。】

实验结果如下：

# C:\Users\wuyu\Documents\Tencent Files\1124274751\FileRecv\MobileFile\Image\YXOT14`]CGJ0XGSU)`6ZB@R.png

