学号 P21714001 专业 计算机英才班 姓名 刘峰

实验日期 **2019.12.20**  教师签字 成绩

实验报告

【实验名称】 **先进先出和最近最久未(LRU)使用**

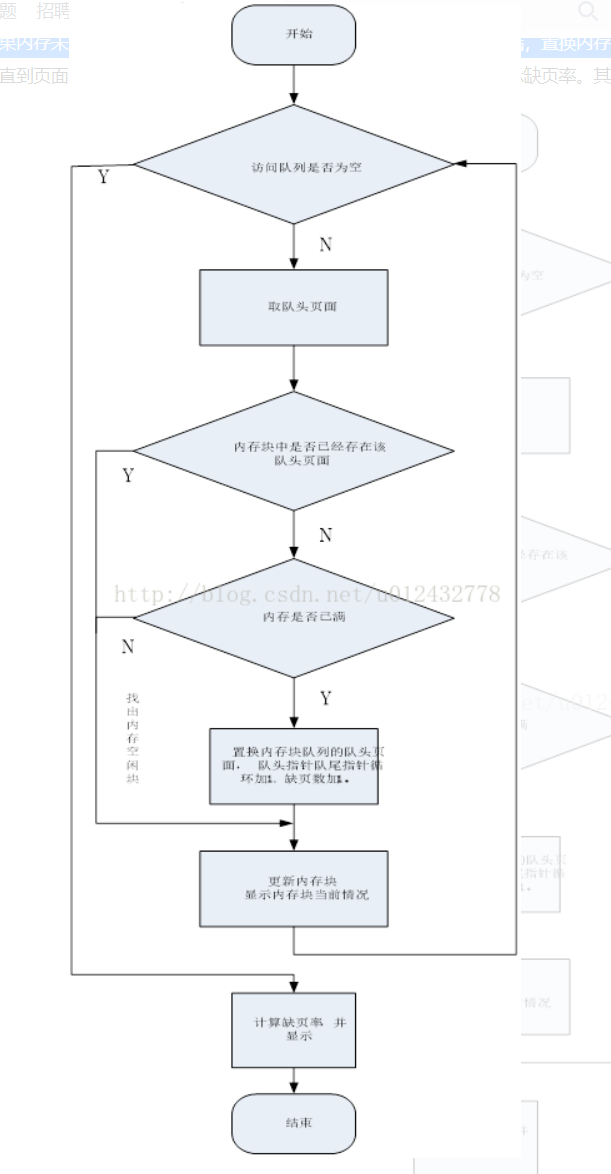
【实验目的】

为了使大的作业（其地址空间超过主存可用空间）或多个作业的地址空间之和超过实际主存空间时仍能运行，引入了虚拟内存的概念。使运行作业的一部分地址空间在主存，另一部分地址空间在辅存，由操作系统实现多级存储器的自动管理，实现主存空间的自动覆盖。要求学生通过本实验，对请求分页管理有一个清楚的概念。

【实验原理】

在内存块初始化后，取出页面访问序列队列的队头。首先判断内存块中是否已经存在该队头页面，如果存在则直接显示内存块当前情况；否则，判断此时内存是否已满。如果内存未满，循环遍历找出空闲内存块，进行页面置换；若内存已满，置换内存块队列的队头页面，缺页数加1，队头指针队尾指针循环加1.

FIFO算法的性能较差，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状况。最近最久未使用（LRU）的页面置换算法是根据页面调入内存后的使用情况做出决策的。由于无法预测各页面将来的使用情况，只能利用“最近的过去”作为“最近的将来”的近似，因此，LRU置换算法是选择最近最久未使用的页面予以淘汰。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来所经历的时间t。当需要淘汰一个页面时，选择现有也面中t值最大的，即最近最久未使用的页面予以淘汰



【实验内容】

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

const int MaxNumber=100;

int PageOrder[MaxNumber]; //页面序列

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

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

double LackPageRate; //缺页率

int LackPageNum; //缺页数

int VirtualQueue[MaxNumber]; //虚拟队列

void input();

void initial();

void FIFO(); //先进先出

void LRU(); //最近最久未使用LRU页面置换算法

void display();

void input()

{

ifstream readData;

readData.open("data.txt");

readData>>MinBlockNum;

readData>>PageNum;

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

{

readData>>PageOrder[i];

}

cout<<"读取数据结果如下："<<endl;

cout<<"最小物理块数 = "<<MinBlockNum<<endl;

cout<<"页面个数 = "<<PageNum<<endl;

cout<<"页面序列如下:"<<endl;

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

{

cout<<PageOrder[i]<<" ";

}

cout<<endl;

}

void initial()

{

LackPageNum = MinBlockNum;

LackPageRate = 0.0;

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

{

PageDisCount[i] = 0; //初始化距离都为0

VirtualQueue[i] = -1; //初始化队列的值都为负数

}

for (int i = 0;i<MinBlockNum;i++)

{

bool isInQueue2 = false;

int dis = 0;

LRUtime[i] = 0;

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

{

if (VirtualQueue[j] == PageOrder[i])

{

isInQueue2 = true;

}

}

if (!isInQueue2) //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];

for (int 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 FIFO()

{

cout<<"\*\*\*\*\*\*\*\*\* 你选择了FIFO算法：\*\*\*\*\*\*\*\*\* "<<endl;

cout<<"页面置换情况如下:"<<endl;

initial();

bool isInQueue;

int point = 0; //指向最老的页面

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

{

isInQueue = false;

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

{

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

{

isInQueue = true;

}

}

if (!isInQueue) //如果当前页面不在队列中，则进行相应的处理

{

LackPageNum++; //缺页数加1

VirtualQueue[point] = PageOrder[i];

display();

point++;

if (point == MinBlockNum)

{

point = 0; //当point指向队尾后一位的时候，将point重新指向队首

}

}

}

LackPageRate = (LackPageNum \* 1.0)/PageNum;

cout<<"缺页数LackPageNum = "<<LackPageNum<<endl;

cout<<"缺页率LackPageRate = "<<LackPageRate<<endl;

}

void OPI()

{

cout<<"\*\*\*\*\*\*\*\*\* 你选择了OPI算法：\*\*\*\*\*\*\*\*\* "<<endl;

cout<<"页面置换情况如下:"<<endl;

initial();

bool isInQueue;

int dis; //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(int i = MinBlockNum;i<PageNum;i++)

{

isInQueue = false;

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

{

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

{

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 (int s = 0;s < MinBlockNum;s++)

{

dis = 1;

for (int t = i;t<PageNum;t++) //从页面序列的第i个位置开始找起

{

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

{

dis++;

}

else

{

break;

}

}

PageDisCount[s] = dis;

}

point = 0;

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

{

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

{

point = m;

}

}

VirtualQueue[point] = PageOrder[i];

display();

}//if

}//for

LackPageRate = (LackPageNum\*1.0)/PageNum;

cout<<"缺页数LackPageNum = "<<LackPageNum<<endl;

cout<<"缺页率LackPageRate = "<<LackPageRate<<endl;

}

void LRU()

{

cout<<"\*\*\*\*\*\*\*\*\* 你选择了LRU算法：\*\*\*\*\*\*\*\*\* "<<endl;

cout<<"页面置换情况如下:"<<endl;

initial();

bool isInQueue;

int point,k; //指向最长时间未被访问的下标

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

{

isInQueue = false;

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

{

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

{

isInQueue = true;

}

}

if (!isInQueue)

{

LackPageNum++;

point = 0;

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

{

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

{

point = j;

}

}

for (int 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();

}//if

else //负责更新当前对应页面的时间

{

for (int 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;

}

}

}//for

LackPageRate = (LackPageNum\*1.0)/PageNum;

cout<<"缺页数LackPageNum = "<<LackPageNum<<endl;

cout<<"缺页率LackPageRate = "<<LackPageRate<<endl;

}

void display()

{

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

{

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

}

cout<<endl;

}

int main()

{

input();

int isContinue = 1;

int chooseAlgorithm;

while(isContinue)

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\* 请选择算法 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\* 1代表FIFO算法 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\* 2代表LRU算法 \*\*\*\*\*\*\*\*\*\*"<<endl;

cin>>chooseAlgorithm;

switch(chooseAlgorithm)

{

case 1:

FIFO();

break;

case 2:

LRU();

break;

default:

cout<<"请输入正确的序号进行选择："<<endl;break;

}

cout<<"\*\*\*\*\*\*\*\*\*\* 是否继续选择算法? \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 输入1代表继续，输入0代表退出！ \*\*\*\*\*\*\*\*\*\*"<<endl;

cin>>isContinue;

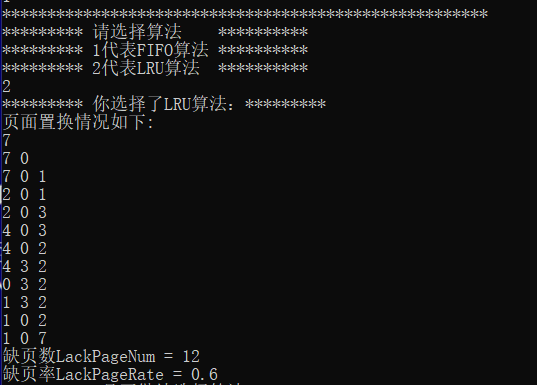
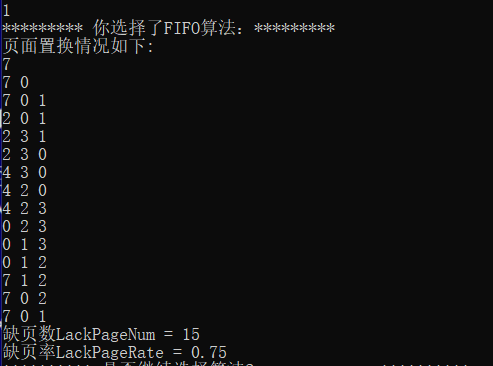
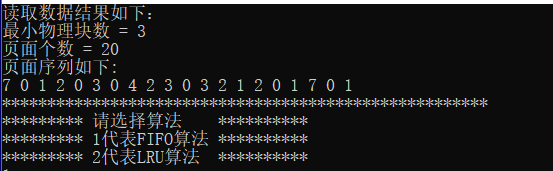
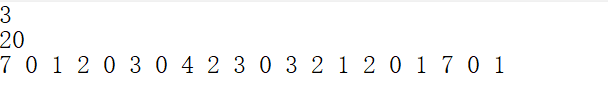
}

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

return 0;

}

输入序列：



【实验总结】

通过本次实验更加深入了解了FIFO和LRU算法工作以及具体实现过程，增加了我对虚拟内存管理的认识。实验实现过程也加深对所学数据结构以及相关算法的理解与回顾。