**页式存储管理**

**——最近最久未使用页面置换算法(LRU)**

在这一次的页面存储管理的实验课上，老师给我们提供了先进先出算法(FIFO),在理论课的学习当中老师已经讲解了它的优点和缺点，其中一个缺点就是它额性能比较差，因为它所依据的是各个页面调入内存的时间，而页面调入的先后不能反应页面的使用情况，所以我就结合老师给的代码，在模仿的基础上自己写了一个简易版的的最近最久未使用的页面置换算法(LRU)。

这个算法的原理就是看虚存中的哪一个页面是最久未使用的，就把哪一个页面调出去，再去把当前需要的页面调入虚存当中。

**我再下面就直接贴上我自己写的代码，和它的运行截图。**

**代码：**

//最近最久未使用的页面置换算法

#include<iostream>

#include<algorithm>

#include<cstdio>

#include<ctime>

#include<cstdlib>

using namespace std;

const int max\_page=10;

const int max\_number\_code=1000+5;

int codeIndexInMemry[max\_page];

int codeToIndex[32];

int codeIndex[max\_number\_code];

int numberOfcode;

bool isInMemry[32];

int useFre[10];

int lack\_page;

void Pro\_page()

{

srand(time(0));

numberOfcode=rand()%200;//假设有n条指令需要调用

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

codeIndex[i]=rand()%320;

printf("产生的需要调入的指令的条数为 ：%d\n",numberOfcode);

for(int i=0;i<numberOfcode;i++){

printf("%-5d",codeIndex[i]);

if(i%10==9)

printf("\n");

}

printf("\n");

}

void pre\_insert()

{

int usedPageNumber=0;

int page;

for(int i=0;i<32;i++)

codeToIndex[i]=-1;

for(int i=0;i<numberOfcode ;i++){

page=codeIndex[i]/10;

if(!isInMemry[page]){

codeIndexInMemry[usedPageNumber]=page;

codeToIndex[page]=usedPageNumber;

isInMemry[page]=1;

usedPageNumber++;

}

if(usedPageNumber>=10)

break;

}

printf("预调入完成，结果如下 ：\n");

for(int i=0;i<10;i++){

printf("%d ",codeIndexInMemry[i]);

}

printf("\n");

}

bool CheckInMemry(int page)

{

return isInMemry[page];

}

void change(int page)

{

int change\_one=0;

for(int i=1;i<10;i++)

if(useFre[change\_one]<useFre[i])

change\_one=i;

//讲change\_one中的东西置换出来

useFre[change\_one]=0;

int oldPage=codeIndexInMemry[change\_one];

printf("将%d换出，与此同时将%d换出\n",oldPage,page);

isInMemry[oldPage]=0;

isInMemry[page]=1;

codeIndexInMemry[change\_one]=page;//把当前的page放到原本来的地方

codeToIndex[page]=change\_one;

codeToIndex[oldPage]=-1;

}

void LRU()

{

for(int i=0;i<numberOfcode;i++){

int page=codeIndex[i]/10;

printf("需要用到的是第%d行代码，它在%d页\n",codeIndex[i],page);

if(CheckInMemry(page)){

//如果已经在虚存中了

for(int i=0;i<10;i++)

useFre[i]++;

useFre[codeToIndex[page]]=0;

printf("第%d页已经在虚存中\n",page);

}

else{

printf("第%d不在虚存中，启用LRU\n");

change(page);

printf("置换后在虚存中的情况如下：\n");

for(int i=0;i<10;i++)

printf("%d ",codeIndexInMemry[i]);

printf("\n");

lack\_page++;

}

}

}

int main()

{

Pro\_page();

//因为需要执行的指令是一次性产生的，所以我们在执行之前已经知道了那些指令在接下来需要呗调入，所以我们提前采用预调入算法

pre\_insert();

lack\_page=0;

LRU();//调用最近最久未使用算法

printf("缺页次数为： %d\n",lack\_page);

printf("命中率为：%f\n",1.0-lack\_page\*1.0/numberOfcode);

return 0;

}

**运行截图**：(因为这个模拟虚存管理的运行结果比较长，我就只截取了最后一部分，再这一部分中显示了缺页次数，并且计算了命中率)

