

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

#define PAGES 320 //页面号引用数

int M;

int page[PAGES]; //页面号引用串

/\*物理块结构体\*/

typedef struct

{

int num; //该块中所存的页面号

int tm; //从最近一次调入所经历的时间

int ntm;

}PBlock;

/\*初始化物理块和结果数组为-1\*/

void init(PBlock \*pb,int M)

{

int i,j;

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

{

pb[i].num=-1;

pb[i].tm=-1;

pb[i].ntm=-1;

/\*for(j=0; j<PAGES; j++)

rs[i][j]=-1;\*/

}

}

/\*根据实验要求构造一个长320的随机数数组，存放的指令序列满足实验要求\*/

void createData()

{

int s,i,j;

srand(time(0)); /\*每次运行时进程号不同，用来作为初始化随机数队列的“种子”\*/

s=rand()%319+1; //

for(i=0;i<PAGES;i+=4) /\*产生指令队列\*/

{

if(s<0||s>319)

{

printf("When i==%d,Error,s==%d\n",i,s);

exit(0);

}

page[i]=s; /\*任选一指令访问点m\*/

page[i+1]=page[i]+1; /\*顺序执行一条指令\*/

page[i+2]=rand( )%page[i]; /\*执行前地址指令m' \*/

page[i+3]=page[i+2]+1; /\*顺序执行一条指令\*/

s=rand()%(319-page[i+2])+page[i+2]+1;

if((page[i+2]>318)||(s>319))

printf("page[%d+2],a number which is :%d and s==%d\n",i,page[i+2],s);

}

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

page[i]=page[i]/10;

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

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

printf("\n");

}

/\*判断页面号为num的页面是否在内存块中,存在返回1\*/

int doExist(int num,PBlock \*pb,int M)

{

int i;

int b=0;

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

{

if(pb[i].num==num)

{

b=1;

break;

}

}

return b;

}

/\*FIFO 算法\*/

void FIFO(PBlock \*pb,int M)

{

int i;

int qy=0; //缺页次数

int p=0; //替换指针

int index=0; //页面号索引

while(index<PAGES)

{

if(!doExist(page[index],pb,M)) //如果物理块内不存在页面号，则按顺序写入

{

pb[p].num=page[index];

p=(p+1)%M;

qy++;

}

index++;

}

printf("FIFO:\n");

printf("\tFIFO算法缺页次数为: %d\n",qy);

printf("\thit rate= %.4f \n",1-(float)qy/320);

}

/\*获得最近最久未使用的页面的块号\*/

int selectByTm(PBlock \*pb,int p,int M)

{

int i,j;

PBlock temp;

bool notEmpty;

notEmpty=true;

for(i=0;i<M;i++) //如果物理块存在空，则直接选择空位置

{

if(pb[i].tm==-1)

{

notEmpty=false;

p=i;

break;

}

}

if(notEmpty) //如果全满，则比较它们的tm，选出tm最大的位置

{

for(i=0;i<M-1;i++)

{

if(pb[i].tm>pb[i+1].tm)

{

temp=pb[i];

pb[i]=pb[i+1];

pb[i+1]=temp;

}

}

p=M-1;

}

return p;

}

/\*LRU 算法\*/

void LRU(PBlock \*pb,int M)

{

int i;

int qy=0; //缺页次数

int p=0; //替换指针

int index=0; //页面号索引

while(index<PAGES)

{

if(!doExist(page[index],pb,M)) //如果物理块内不存在页面号，每个物理块的tm++

{

for(i=0;i<M;i++) //然后通过selectByTm方法选出要替换的，并置tm为0

if(pb[i].num!=-1)

pb[i].tm++;

p=selectByTm(pb,p,M);

pb[p].num=page[index];

pb[p].tm=0;

qy++;

}

else //如果已存在页面号，则使对应位置的tm++

{

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

{

if(pb[i].num==page[index])

{

pb[i].tm=0;

}

else

{

pb[i].tm++;

}

}

}

index++;

}

printf("LRU:\n");

printf("\tLRU算法缺页次数为: %d \n",qy);

printf("\thit rate= %.4f \n",1-(float)qy/320);

}

/\*获得最长时间内不被使用的页面的块号\*/

int selectByOpt(PBlock \*pb,int index,int p,int M)

{

int i,j;

PBlock temp;

bool notEmpty;

notEmpty=true;

for(i=0;i<M;i++) //如果物理块存在空，则直接选择空位置

{

if(pb[i].ntm==-1)

{

notEmpty=false;

p=i;

break;

}

}

if(notEmpty) //如果全满，则遍历页面号串找到再次出现的位置

{ //分别记录三个物理块的ntm，选出最大的作为要替换的页面号

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

{

for(j=index;j<PAGES;j++)

{

if(pb[i].num==page[j])

{

pb[i].ntm=j;

break;

}

if(j==(PAGES-1)&&pb[i].num!=page[j])

pb[i].ntm=1000;

}

}

for(i=0;i<M-1;i++)

{

if(pb[i].ntm>pb[i+1].ntm)

{

temp=pb[i];

pb[i]=pb[i+1];

pb[i+1]=temp;

}

}

p=M-1;

}

return p;

}

/\*OPT算法\*/

void OPT(PBlock \*pb,int M)

{

int i;

int qy=0; //缺页次数

int p=0; //替换指针

int index=0; //页面号索引

while(index<PAGES)

{

if(!doExist(page[index],pb,M)) //如果物理块内不存在页面号

{

p=selectByOpt(pb,index,p,M);

//printf("[%d] ",p);

pb[p].ntm=1000;

pb[p].num=page[index];

qy++;

}

index++;

}

printf("OPT:\n");

printf("\tOPT算法缺页次数为: %d \n",qy);

printf("\thit rate= %.4f \n",1-(float)qy/320);

}

int main()

{

createData();

for(M=4;M<33;M++)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*物理块数为: %d \*\*\*\*\*\*\*\*\*\*\*\n",M);

PBlock pb[M];

init(pb,M);

FIFO(pb,M);

init(pb,M);

LRU(pb,M);

init(pb,M);

OPT(pb,M);

}

return 0;