**模拟操作系统的页面置换**

1. 中文摘要

在地址映射过程中，若在页面中发现所要访问的页面不在内存中，则产生缺页中断。当发生缺页中断时，如果操作系统内存中没有空闲页面，则操作系统必须在内存选择一个页面将其移出内存，以便为即将调入的页面让出空间。而用来选择淘汰哪一页的规则叫做页面置换算法。实验通过程序模拟操作系统的页面置换过程，意在更加深入地了解其思想。

1. 关键词

页面置换 算法 OPT FIFO LRU

1. 前言

1、目的与意义

（1）、掌握操作系统的页面置换过程，加深理解页式虚拟存储器的实现原理。

（2）、掌握用随机数生成满足一定条件的指令地址流的方法。

（3）、掌握页面置换的模拟方法。

2、内容

（1）、模拟操作系统采用 OPT、FIFO 和LRU 算法进行页面置换的过程。

（2）、设程序中地址范围为 0 到 32767，采用随机数生成256 个指令地址，满足50%的地址是顺序执行，25%向前跳，25%向后跳。为满足上述条件， 可采取下列方法：设d0 =10000，第n个指令地址为dn，第n+1 个指令地址为dn+1，n的取值范围为0到255。每次生成一个1 到 1024 范围内的随机数a，如果a落在1到512范围内，则dn+1 =dn +1。如果a落在 513 到 768范围内，则设置dn+1为1到dn范围内一个随机数。如果a落在 769 到 1024范围内，则设置dn+1为dn到32767 范围内一个随机数。

例如：srand();初始化一个随机函数。

a[0]＝10\*rand()/32767\*255+1;a[1]=10\*rand()/32767\*a[0]…语句可用来

产生 a[0]与 a[1]中的随机数。

或采用以下方式：

<1>通过随机数产生一个指令序列，共 320 条指令。指令的地址按下述原则生成：

A：50%的指令是顺序执行的

B：25%的指令是均匀分布在前地址部分

C：25%的指令是均匀分布在后地址部分

具体的实施方法是：

A：在[0，319]的指令地址之间随机选取一起点 m

B：顺序执行一条指令，即执行地址为 m+1 的指令

C：在前地址[0,m+1]中随机选取一条指令并执行，该指令的地址为 m'

D：顺序执行一条指令，其地址为 m'+1

E：在后地址[m'+2，319]中随机选取一条指令并执行

F：重复步骤 A-E，直到 320 次指令

<2>将指令序列变换为页地址流

设：页面大小为 1K；

用户内存容量 4 页到 32 页；

用户虚存容量为 32K。

在用户虚存中，按每 K 存放 10 条指令排列虚存地址，即 320 条指令在虚存中的存放方式为：

第 0 条-第 9 条指令为第 0 页（对应虚存地址为[0，9]）

第 10 条-第 19 条指令为第 1 页（对应虚存地址为[10，19]）

………………………………

第 310 条-第 319 条指令为第 31页（对应虚存地址为[310，319]）

按以上方式，用户指令可组成 32 页。

（4）、页面大小的取值范围为 1K，2K，4K，8K，16K。按照页面大小将指令地址转化为页号。对于相邻相同的页号，合并为一个。

（5）、分配给程序的内存块数取值范围为 1 块，2 块，直到程序的页面数。

（6）、分别采用 OPT、FIFO 和 LRU 算法对页号序列进行调度，计算出对应的缺页中断率。

（7）、打印出页面大小、分配给程序的内存块数、算法名、对应的缺页中断率。

（8）、分析页面大小和分配给程序的内存块数对缺页中断率的影响。分析不同的页面置换算法的调度性能。

（9）、在上机实现该程序之后，要求写出实验报告，其中包括实验名称、实验 目的、实验内容、程序的主要流程图、实验心得和主要源程序清单等。

四、算法设计

1、采用关键技术

OPT：这是一种理想情况下的页面置换算法，但实际上是不可能实现的。该算法的基本思想是：发生缺页时，有些页面在内存中，其中有一页将很快被访问（也包含紧接着的下一条指令的那页），而其他页面则可能要到10、100或者1000条指令后才会被访问，每个页面都可[1] 以用在该页面首次被访问前所要执行的指令数进行标记。最佳页面置换算法只是简单地规定：标记最大的页应该被置换。这个算法唯一的一个问题就是它无法实现。当缺页发生时，操作系统无法知道各个页面下一次是在什么时候被访问。虽然这个算法不可能实现，但是最佳页面置换算法可以用于对可实现算法的性能进行衡量比较。

FIFO：最简单的页面置换算法是先入先出（FIFO）法。这种算法的实质是，总是选择在主存中停留时间最长（即最老）的一页置换，即先进入内存的页，先退出内存。理由是：最早调入内存的页，其不再被使用的可能性比刚调入内存的可能性大。建立一个FIFO队列，收容所有在内存中的页。被置换页面总是在队列头上进行。当一个页面被放入内存时，就把它插在队尾上。

这种算法只是在按线性顺序访问地址空间时才是理想的，否则效率不高。因为那些常被访问的页，往往在主存中也停留得最久，结果它们因变“老”而不得不被置换出去。FIFO的另一个缺点是，它有一种异常现象，即在增加存储块的情况下，反而使缺页中断率增加了。当然，导致这种异常现象的页面走向实际上是很少见的。

LRU：最近最久未使用算法，FIFO算法和OPT算法之间的主要差别是，FIFO算法利用页面进入内存后的时间长短作为置换依据，而OPT算法的依据是将来使用页面的时间。如果以最近的过去作为不久将来的近似，那么就可以把过去最长一段时间里不曾被使用的页面置换掉。它的实质是，当需要置换一页时，选择在之前一段时间里最久没有使用过的页面予以置换。这种算法就称为最久未使用算法（Least Recently Used，LRU）。在采用最少使用置换算法时，应为在内存中的每个页面设置一个移位寄存器，用来记录该页面被访问的频率。该置换算法选择在之前时期使用最少的页面作为淘汰页。

2、主要设计思想

（1）采用一个320长的数组来将随机产生的指令储存起来，用于后面的继续操作。

（2）通过输入的页面大小将指令划分出其所在的页面。

（3）合并相同的页面。

（4）OPT算法实现：通过一个数组将所执行的指令以后的后面指令页面最近的位置记录起来，若无则置为500（极大值），通过比较，将数组中位置最远的页面用于置换，并计算其置换率。

（5）FIFO算法实现：通过一个计数器将当前块总量中最早执行的位置记录下来，并在需要页面置换时将该块替换，毕竟通过自加1和取余操作不断更新最早执行页面位置，并计算其置换率。

（6）LRU算法实现：通过一个数组将执行块中的所有的最近执行的位置，通过比较，将数组中最久未被访问的页面用于置换，并计算其置换率。

3、设计流程

（1）编写初始化函数，利用随机数产生要求页面

（2）通过输入确定页面大小，并找出其所在的页面

（3）合并连续相同的页面

（4）编写OPT算法实现

（5）编写FIFO算法实现

（6）编写LRU算法实现

（7）通过main函数测试各个功能

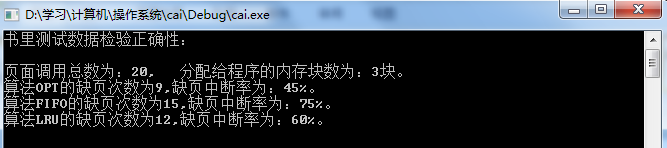
五、所采用软硬件平台

系统：win7

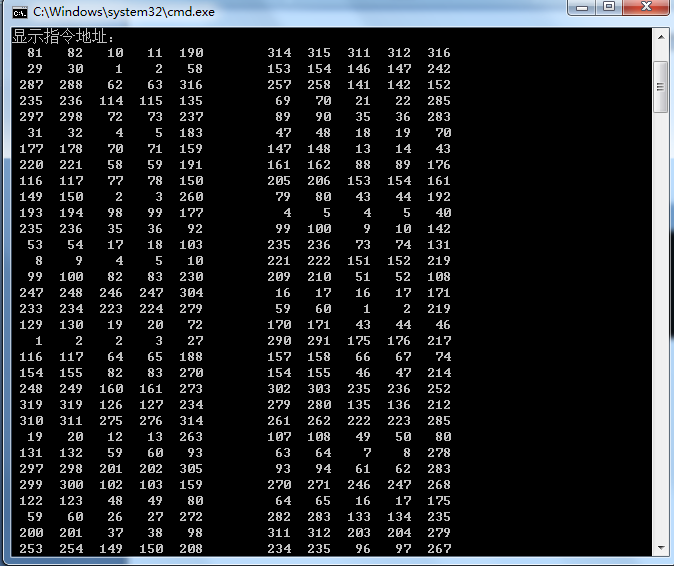
所用软件：vs2017

六、结果与结果分析

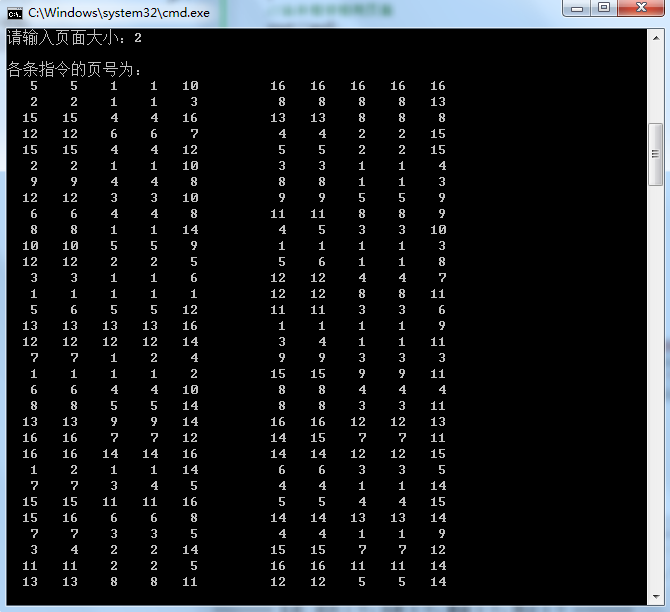
1、书里测试数据检测：



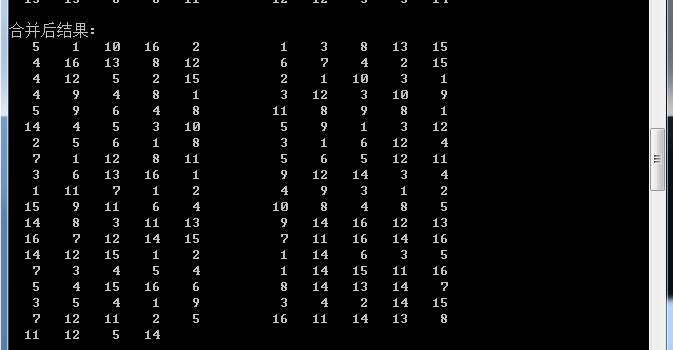
2、随机产生指令：



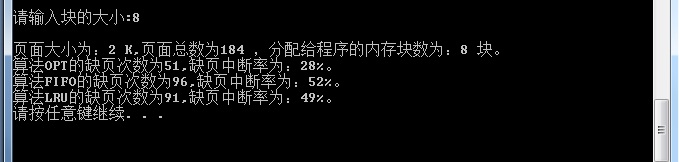
3、根据输入确定指令所在页面：



4、合并相同页面：



5、根据输入内存块大小计算缺页中断率：



6、结果分析：

（1）通过测试的测试数据，可以基本确定算法设计的正确性，测试数据的三种缺页中断率与书本中一致。

（2）通过观察，随机产生的指令基本符合实验要求的五种循环规律。

（3）通过输入页面大小计算的指令所在页面基本正确，且能够正确合并。

（4）通过输入内存块大小，计算的随机产生值与理论值的比例不是很相符，但基本满缺页中断率FIFO>LRU>OPT,基本满足要求。

七、参考文献

《操作系统教程》机械工业出版社，作者谢旭升、朱明华、张练兴、李宏伟

《百度百科》

附录：

主要功能模块

全局变量：

int aa[320];

int bb[320];

int cc[320]={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};

int dd[32][2];

double opt,fifo,lru;

double interrupt1,interrupt2,interrupt3;

1. 编写初始化函数，利用随机数产生要求页面

void setInstruction()

{

int n;

srand(time(NULL));

//产生320条指令

for(int i=0;i<320;i++)

{

n=320;

aa[i]=rand()%n+1;//产生随机指令

if(aa[i]==319)//顺序执行一条指令

aa[i+1]=aa[i];

else

aa[i+1]=aa[i]+1;

//往前跳产生指令

n=aa[i+1];

i=i+2;

aa[i]=rand()%n+1;

if(aa[i]==319)//顺序执行一条指令

aa[i+1]=aa[i];

else

aa[i+1]=aa[i]+1;

i=i+2;

//往后跳产生一条指令

while(1)

{

aa[i]=rand()%320+1;

if(aa[i]>aa[i-2]+2)

break;

}

}

return ;

}

1. 通过输入确定页面大小，并找出其所在的页面

void setPage(int n)//通过输入值确定其所在页面值

{

for(int i=0;i<320;i++)

bb[i]=aa[i]/(n\*10)+1;

return ;

}

1. 合并连续相同的页面

int reduced()

{

int i,n=1,c;

//合并相邻相同值

cc[0]=c=bb[0];

for(i=0;i<320;i++)

{

if(c!=bb[i])

{

cc[n]=c=bb[i];

n++;

}

}

return n;

}

1. 编写OPT算法实现

void OPT(int num,int Bnum)

{

opt=0;

int i=0,j,k=0,l,max,m;

bool flag;

memset(dd,0,sizeof(dd));

while(i<Bnum)//通过Bnum将总的块填满

{

flag=false;

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

{

if(dd[j][0]==cc[k])//判断是否已存在于内存块中

{

flag=true;

break;

}

}

if(!flag)//不存在则写入

{

dd[i][0]=cc[k];

opt++;//记录缺页中断

for(l=k+1;l<num;l++)//找出该页面最近的位置

{

if(cc[l]==dd[i][0])

{

dd[i][1]=l;

break;

}

if(cc[l]!=dd[i][0]&&l==num-1)

{

dd[i][1]=500;

}

}

i++;

}

else

{

for(l=k+1;l<num;l++)//存在则重新修改，找出该页面新的最近的位置

{

if(cc[l]==dd[i][0])

{

dd[i][1]=l;

break;

}

if(cc[l]!=dd[i][0]&&l==num-1)

{

dd[i][1]=500;

}

}

}

k++;

}

//内存块已经满了以后的操作

for(;k<num;k++)

{

//判断是否已存在于内存块中

max=-1;

flag=false;

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

{

if(dd[j][0]==cc[k])

{

flag=true;

//存在则重新修改，找出该页面新的最近的位置

for(l=k+1;l<num;l++)

{

if(cc[l]==dd[j][0])

{

dd[j][1]=l;

break;

}

if(cc[l]!=dd[j][0]&&l==num-1)

{

dd[j][1]=500;

}

}

break;

}

}

if(!flag)//不存在则置换页面

{

opt++;//记录缺页中断

for(i=0;i<Bnum;i++)//找出最远不会被调用的页面

{

if(dd[i][1]>max)

{

max=dd[i][1];

m=i;

}

}

dd[m][0]=cc[k];//将其置换

for(l=k+1;l<num;l++)//找出该页面新的最近的位置

{

if(cc[l]==dd[m][0])

{

dd[m][1]=l;

break;

}

if(cc[l]!=dd[m][0]&&l==num-1)

{

dd[m][1]=500;

}

}

}

}

interrupt1=opt/num\*100;//计算其缺页中断率

return ;

}

1. 编写FIFO算法实现

void FIFO(int num,int Bnum)

{

fifo=0;

int i=0,j,k=0,m;

bool flag;

memset(dd,0,sizeof(dd));

while(i<Bnum)//通过Bnum将总的内存块填满

{

flag=false;

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

{

if(dd[j][0]==cc[k])//判断是否已存在于内存块中

{

flag=true;

break;

}

}

if(!flag)//不存在则写入

{

dd[i][0]=cc[k];

fifo++;//记录缺页中断

i++;

}

k++;

}

m=0;//记录当前最早开始的内存块

for(;k<num;k++)

{

flag=false;

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

{

if(dd[j][0]==cc[k])//判断是否已存在于内存块中

{

flag=true;

break;

}

}

if(!flag)//不存在则置换

{

fifo++;//记录缺页中断

dd[m][0]=cc[k];//将其置换

m++;//确定新的最早开始的内存块

m=m%Bnum;

}

}

interrupt2=fifo/num\*100;//计算其缺页中断率

return ;

}

1. 编写LRU算法实现

void LRU(int num,int Bnum)

{

int i=0,j,k=0,min,m;

bool flag;

lru=0;

memset(dd,0,sizeof(dd));

while(i<Bnum)//通过Bnum将总的内存块填满

{

flag=false;

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

{

if(dd[j][0]==cc[k])//判断是否已存在于内存块中

{

dd[j][1]=k;//修改当前执行位置

flag=true;

break;

}

}

if(!flag)//不存在则写入

{

dd[i][0]=cc[k];

dd[i][1]=k;//记录当前执行位置

lru++;//记录缺页中断

i++;

}

k++;

}

//内存块已经满了以后的操作

for(;k<num;k++)

{

min=500;

flag=false;

for(j=0;j<Bnum;j++)//判断是否已存在于内存块中

{

if(dd[j][0]==cc[k])//存在则重新修改，记录其当前执行位置

{

dd[j][1]=k;

flag=true;

break;

}

}

if(!flag)//不存在则置换页面

{

lru++;//记录缺页中断

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

{

if(dd[i][1]<min)//找出最久未被执行的块内存

{

min=dd[i][1];

m=i;

}

}

dd[m][0]=cc[k];//将其置换出来

dd[m][1]=k;//记录当前执行位置

}

}

interrupt3=lru/num\*100;//计算缺页中断率

return ;

}

1. 通过main函数测试各个功能

int main()

{

int i,n,num,Bnum;

num=20;

Bnum=3;

n=0;

OPT(num,Bnum);

FIFO(num,Bnum);

LRU(num,Bnum);

cout<<"书里测试数据检验正确性：\n";

cout<<"\n页面调用总数为："<<num<<", 分配给程序的内存块数为："<<Bnum<<"块。\n";

cout<<"算法OPT的缺页次数为"<<(int)opt<<",缺页中断率为："<<setprecision(2)<<interrupt1<<"%。\n";

cout<<"算法FIFO的缺页次数为"<<(int)fifo<<",缺页中断率为："<<setprecision(2)<<interrupt2<<"%。\n";

cout<<"算法LRU的缺页次数为"<<(int)lru<<",缺页中断率为："<<setprecision(2)<<interrupt3<<"%。\n\n\n\n";

//随机产生指令

memset(aa,0,sizeof(aa));

setInstruction();

cout<<"显示指令地址：\n";

for(i=0;i<320;i++)

{

if(i%5==0&&i!=0)

if(i%10==0)

cout<<endl;

else

cout<<" ";

cout<<setw(4)<<aa[i]<<" ";

}

//根据输入页面大小确定指令所在页面

cout<<"\n请输入页面大小：";

cin>>n;

setPage(n);

cout<<"\n各条指令的页号为：\n";

for(i=0;i<320;i++)

{

if(i%5==0&&i!=0)

if(i%10==0)

cout<<endl;

else

cout<<" ";

cout<<setw(4)<<bb[i]<<" ";

}

//合并相邻相同页面

cout<<endl;

num=reduced();

cout<<"\n合并后结果:\n";

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

{

if(i%5==0&&i!=0)

if(i%10==0)

cout<<endl;

else

cout<<" ";

cout<<setw(4)<<cc[i]<<" ";

}

cout<<endl;

//输入内存块大小，计算缺页中断率

cout<<"\n请输入块的大小:";

cin>>Bnum;

OPT(num,Bnum);

FIFO(num,Bnum);

LRU(num,Bnum);

//输出测试结果

cout<<"\n页面大小为："<<n<<" K,页面总数为"<<num<<" ，分配给程序的内存块数为："<<Bnum<<" 块。\n";

cout<<"算法OPT的缺页次数为"<<(int)opt<<",缺页中断率为："<<setprecision(2)<<interrupt1<<"%。\n";

cout<<"算法FIFO的缺页次数为"<<(int)fifo<<",缺页中断率为："<<setprecision(2)<<interrupt2<<"%。\n";

cout<<"算法LRU的缺页次数为"<<(int)lru<<",缺页中断率为："<<setprecision(2)<<interrupt3<<"%。\n";

return 0;

}