OS实验4 页面置换算法

计科1601 16281035 陈琦

目录

一、前期准备工作

1. 基本原理概述

为什么会有页面置换算法？

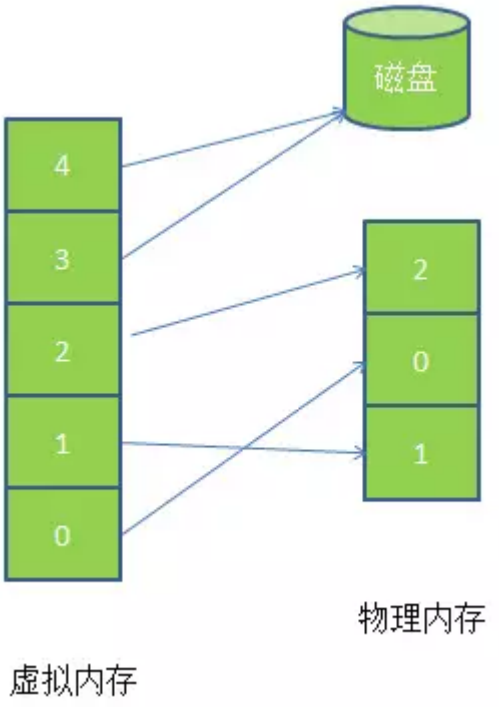


图5. 虚拟内存和物理内存以及磁盘的映射关系

由图5可以看出，虚拟内存实际上可以比物理内存大。当访问虚拟内存时，会访问**MMU**（内存管理单元）去匹配对应的物理地址（比如图5的0，1，2），而如果虚拟内存的页并不存在于物理内存中（如图5的3,4），会产生缺页中断，从磁盘中取得缺的页放入内存，如果内存已满，还会根据某种算法将磁盘中的页换出。

MMU中存储**页表**，用来匹配虚拟内存和物理内存。页表中每个项通常为32位，即4byte,除了存储虚拟地址和页框地址之外，还会存储一些标志位，比如是否缺页，是否修改过，写保护等。因为页表中每个条目是4字节，现在的32位操作系统虚拟地址空间是2^32，假设每页分为4k，也需(2^32/(4*2^10))4=4M的空间，为每个进程建立一个4M的页表并不明智。因此在页表的概念上进行推广，产生**二级页表**，虽然页表条目没有减少，但内存中可以仅仅存放需要使用的二级页表和一级页表，大大减少了内存的使用。

每个进程有4GB的虚拟地址空间，每个进程自己的一套页表。程序中使用的都是4GB地址空间中的虚拟地址。而访问物理内存，需要使用物理地址。

2. 课题假设前提说明

要求：模拟的虚拟内存的地址为16位，页面大小为1K，模拟的物理内存有32K。

经过计算可得该计算机的页面最大64个，物理块最大值为32个。所以在程序设计时，页面及物理块大小均不可以超过最大值，否则将在程序界面中进行提示，用户重新输入。

二、页面访问序列随机生成说明

1. 符合局部访问特性的随机生成算法概述

1. 确定虚拟内存的尺寸N，工作集的起始位置p，工作集中包含的页数e，工作集移动率m（每处理m个页面访问则将起始位置p +1），以及一个范围在0和1之间的值t；
2. 生成m个取值范围在p和p + e间的随机数，并记录到页面访问序列串中；
3. 生成一个随机数r，$0 \le r \le 1$；
4. 如果$r < t$，则为p生成一个新值，否则$p = (p + 1) \bmod N$；
5. 如果想继续加大页面访问序列串的长度，请返回第2步，否则结束。

2. 代码实现

为了验证方便，我将这个页面访问序列随机生成的代码作为另一个新主程序，这样验证起来比较直观。

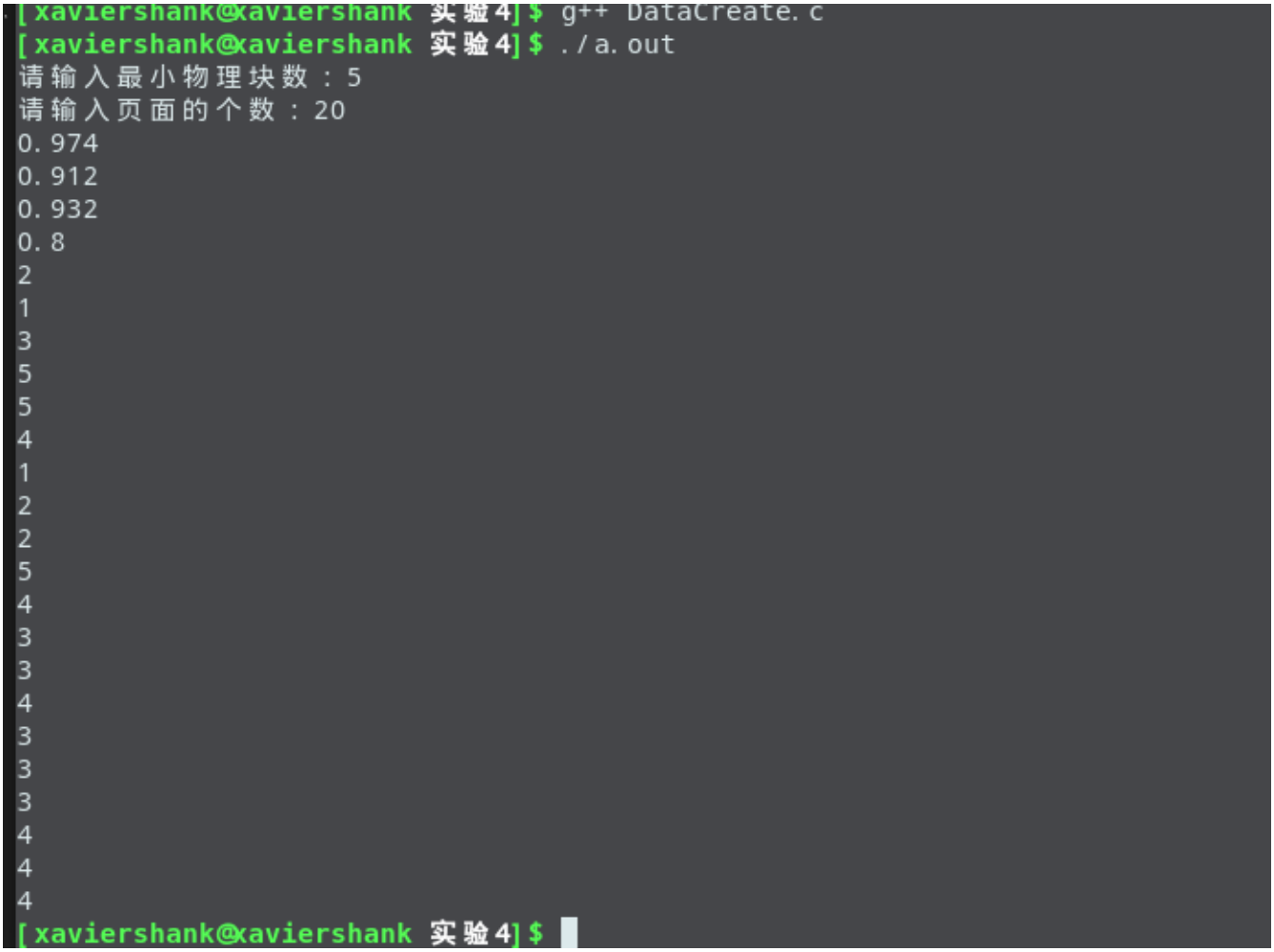
后边的源代码我会把这段程序作为一个**页面访问序列随机生成函数**放入主程序中。

代码的详细解释都在注释中：

```
#include<iostream>
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
const int DataMax = 64;
const int BlockNum = 32;
int Data[DataMax]; // 保存数据
int Block[BlockNum]; // 物理块
int count[BlockNum]; // 计数器
int N ; // 页面个数
int M;//最小物理块数
#define Q 999
using namespace std;

int main()
{
    int p = 1;//设置工作集的起始位置为1
    int e = 4;//工作集包含的页数为4
    int m = 5;//处理5个页面后工作集开始移动
    float t = 0.637;//判断p是否跳转的依据
    float r; //随机生成，与t比较
    int j = 0;//保存随机生成的页面的数组下标


    cout<<"请输入最小物理块数：";
    cin>>M;
    while(M > BlockNum) // 大于数据个数
    {
        cout<<"物理块数超过预定值，请重新输入：";
        cin>>M;
    }
    cout<<"请输入页面的个数：";
    cin>>N;
    while(N > DataMax) // 大于数据个数
    {
        cout<<"页面个数超过预定值，请重新输入：";
        cin>>N;
    }

    srand((int)time(NULL));//用time(0)的返回值当种子
    for(int o=0;o<(N/m-1);o++)
    {
    for(int i=0;i<m;i++)
    {
```

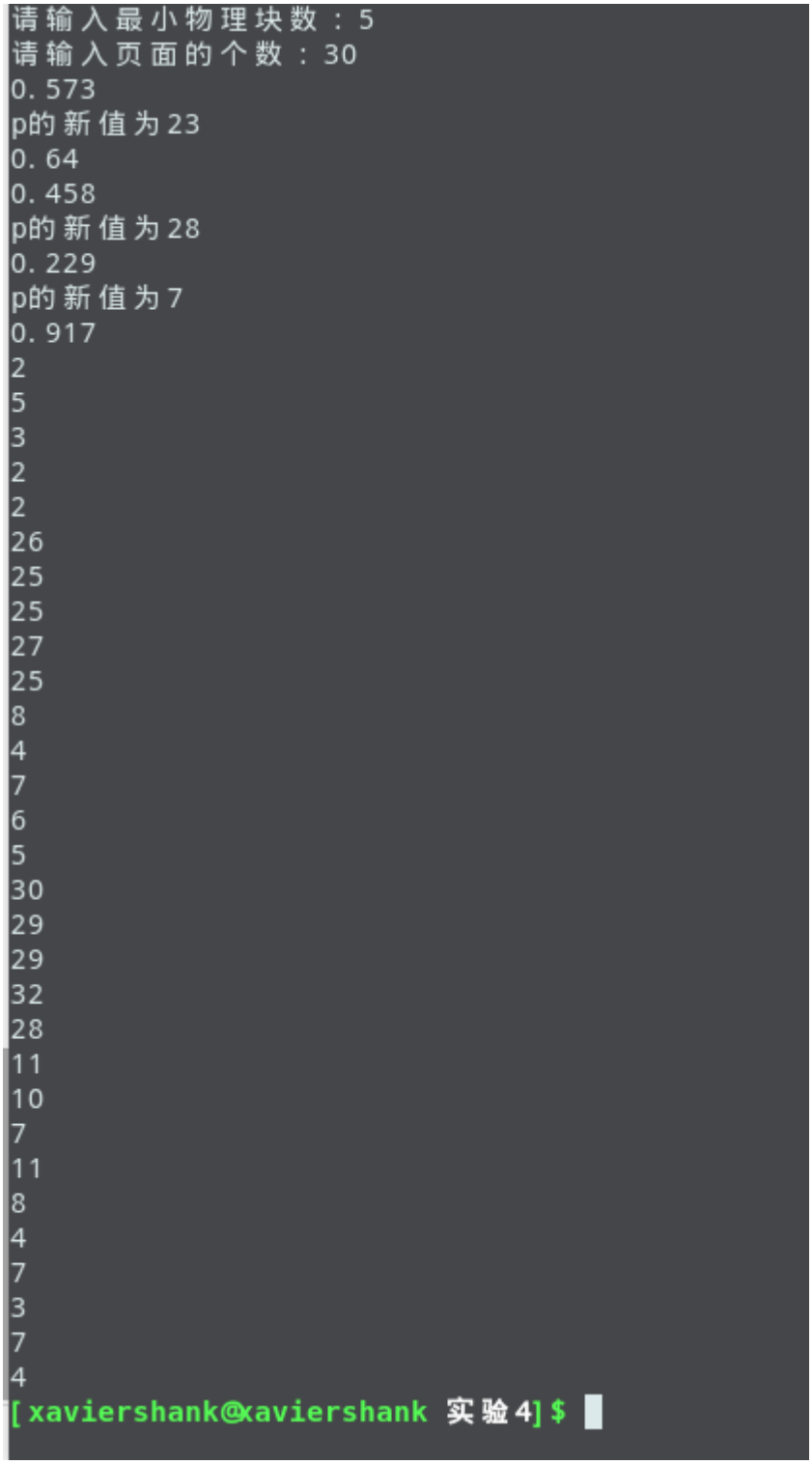
```
        Data[j] = rand() % (e+1) + 1;//生成p和p+e之间的5个随机数
        j++;

        if(j==N) break;//如果生成的页面数大于最大页面数，退出循环
        else if(i == 4) //每处理完5个页面就要决定p的值是否应当跳转
        {

             //生成随机数r
            r = rand() % (Q+1) /(float)(Q+1);//小数点后三位
            cout<<r<<endl;//输出每次随机生成的r的值

            if(r<t)
            {
                p = rand() % N +1; //为p生成一个新值，范围1到在最大页面数之间
            }

            else
            {
                p = (p+1) % M;//p的范围不变
            }
        }
     }
    for(int c=0;c<N;c++)
    {
        cout<<Data[c]<<endl; //输出随机生成的页面序列，检查是否正确
    }
    return 0;
}
```

3. 测试1



可以看到4次随机生成的r值都比t=0.637要大，所以p会一直在初始设定的p到p+e之间，也就是闭区间【1,5】之间。可见我们的生成序列是正确的。

4. 测试2



这次设定页面为30个。

(1) 最初始直接生成5个【p，p+e】之间的页面，也就是【1,5】闭区间上。

(2) 0.537<0.637，p的新值为23，生成页面在【23,27】闭区间上。

(3) 0.64>0.637，生成页面在【4,7】闭区间上。

(4) 0.458<0.637，p的新值为28，生成页面在【28,32】闭区间上。

(5) 0.229<0.637，p的新值为7，生成页面在【7,11】闭区间上。

(6) 0.917>0.637，生成页面在【3,7】闭区间上。

5. 代码部分问题剖析

1. 关于外层循环

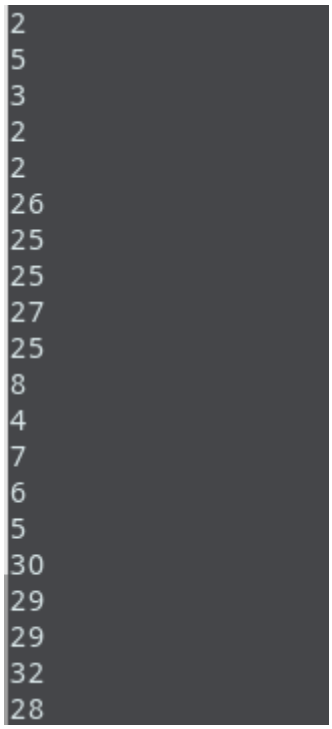
```
for(int o=0;o<(N/m);o++)
   {
   for(int i=0;i<m;i++)
       {
       }
}
```

外层循环的计数应当在0到（N/m-1）之间，因为第一次的五个数是随机生成的，不需要比较r和t的大小。比如页面最大为30个，则需要生成5个随机数r，而不是六个。

2. 关于p = (p + 1) mod N

```
if(r<t)
            {
                p = rand() % N +1; //为p生成一个新值，范围1到在最大页面数之间
            }

            else
            {
                p = (p+1) % M;//p的范围不变
            }
```

我认为p = (p+1) % M这样处理其实不太有利于页面的局部性原理



从上面测试2就可以看出这样生成其实页面的缺页率会很大。

我认为的改进办法是p = (p+1)，不用mod M，这样就能保证在原来页面的基础上增加，符合页面局部性原理。

三、输入输出函数

1. 页面序列随机生成及输入函数DataCreate()

```
void DataCreate()
{
   cout<<"请输入最小物理块数：";
    cin>>M;
 while(M > BlockNum) // 大于数据个数
 {
  cout<<"物理块数超过预定值，请重新输入：";
  cin>>M;
 }
 cout<<"请输入页面的个数：";
 cin>>N;
 while(N > DataMax) // 大于数据个数
 {
  cout<<"页面个数超过预定值，请重新输入：";
  cin>>N;
 }
    int p = 1;
    int e = 4;
    int m = 5;//处理4个页面后起始位置P+1
    float t = 0.637;
    float r;
    int j = 0;

  srand((int)time(NULL));//用time(0)的返回值当种子
    for(int o=0;o<(N/m);o++)
    {
        for(int i=0;i<m;i++)
        {

            Data[j] = rand() % (e+1) + p;//生成p和p+e之间的5个随机数
            j++;

            if(j == N) break;//退出循环
            else if(i == 4)
         {

             //生成随机数r
            r = rand() % (Q+1) /(float)(Q+1);
            cout<<r<<endl;

            if(r<t)
            {
                p = rand() % N +1;
                cout<<"p的新值为"<<p<<endl;
            }

            else
            {
```

```
                p = (p+1) % M;
            }
        }

    }
    }

}
```

2. main函数1

```
int main(int argc, char* argv[])
{
   DataCreate();//  DataInput();
  // FIFO();
  // Optimal();
 //  LRU();
 //  return 0;
      int menu;
     while(true)
     {
       cout<<endl;
       cout<<"*                      菜单选择                          *"<<endl;
       cout<<"*********************************************************"<<endl;
       cout<<"*                       1-FIFO                           *"<<endl;
       cout<<"*                       2-Optimal                        *"<<endl;
       cout<<"*                       3-LRU                            *"<<endl;
       cout<<"*                       0-EXIT                           *"<<endl;
       cout<<"*********************************************************"<<endl;
       cin>>menu;

       switch(menu)
       {
       case 1: FIFO();break;
       case 2: Optimal();break;
       case 3: LRU();break;
       default: break;
       }
       if(menu!=1&&menu!=2&&menu!=3) break;
     }
}
```

3. main函数2

```
int main(){
  char ch ;
  DataCreate();
  for(int i = 0; i < PageCount; i++){
        cout<<Page[i]<<" ";
```

```
    }
  cout<<endl;
  while(1){
    cout<<"------------1.Clock置换算法（CLOCK）-----"<<endl;
    cout<<"------------2.改进型Clock置换算法--------"<<endl;
    cout<<"------------0.退出-----------------------"<<endl;
    cout<<"-------------输入进行选择----------------"<<endl;
    cin>>ch;
    switch(ch){
    case '1':{
            lost = 0;
            count = 0;
            for(int m = 0; m < A; m++){
                state[m] = 0;
                }
                for(int j = 0; j < A; j++){
                    Inside[j] = 0;
                }
                for(int i = 0; i < PageCount; i++){
                    cout<<"读入Page["<<i<<"]="<<Page[i]<<endl;
                    CLOCK(i);
                }
                cout<<"\n页面访问次数"<<PageCount<<"\n缺页中断次数"<<lost<<"\n缺页率"
<<lost/(PageCount)<<"\n"<<endl;
    }break;
    case '2':{
            lost = 0;
            count = 0;
            for(int m = 0; m < A; m++){
                for(int n = 0; n < 2;n++)
                state2[m][n] = 0;
            }
            for(int j = 0; j < A; j++){
                Inside[j] = 0;
            }
            for(int i = 0; i < PageCount; i++){
                cout<<"读入Page["<<i<<"]="<<Page[i]<<endl;
                LCLOCK(i);
            }
            cout<<"\n页面访问次数"<<PageCount<<"\n缺页中断次数"<<lost<<"\n缺页率"
<<lost/(PageCount)<<"\n"<<endl;
    }break;
    case '0':{
        exit(0);
             }break;
    }
  }
  return 0;
}
```

四、页面置换算法

1. 最佳置换算法

```
 void Optimal()
{
 int i,j,k;
 bool find;
 int point;
 int temp; // 临时变量，比较离的最远的时候用
 ChangeTimes = 0;
 for(j=0;j<M;j++)
  for(i=0;i<N;i++)
   DataShowEnable[j][i] = false;  // 初始化为false，表示没有要显示的数据
// for(i=0;i<M;i++)
// {
  // count[i] = 0 ; //
// }
 for(i=0;i<N;i++) // 对有所数据操作
 {
  find = false; // 表示块中有没有该数据
  for(j=0;j<M;j++)
  {
   if( Block[j] == Data[i] )
    find = true;
  }
  if( find ) continue; // 块中有该数据，判断下一个数据
  // 块中没有该数据，最优算法
  ChangeTimes++; // 缺页次数++
  for(j=0;j<M;j++)
  {
   // 找到下一个值的位置
   find = false;
   for( k =i;k<N;k++)
   {
    if( Block[j] == Data[k] )
    {
     find = true;
     count[j] = k;
     break;
    }
   }
   if( !find ) count[j] = N;
  }
  if( (i+1) > M ) // 因为i是从0开始记，而BlockNum指的是个数，从1开始，所以i+1
  {
   //获得要替换的块指针
   temp = 0;
   for(j=0;j<M;j++)
   {
    if( temp < count[j] )
    {
```

```
      temp = count[j];
      point = j; // 获得离的最远的指针
     }
    }
   }
   else point = i;
   // 替换
   Block[point] = Data[i];

   // 保存要显示的数据
   for(j=0;j<M;j++)
   {
    DataShow[j][i] = Block[j];
    DataShowEnable[i<M?(j<=i?j:i):j][i] = true; // 设置显示数据
   }

 }
 // 输出信息
 cout<< endl;
 cout<<"Optimal => "<< endl;
 DataOutput();
}
```

2. 先进先出置换算法

```
void FIFO()
{
 int i,j;
 bool find;
 int point;
 int temp; // 临时变量
 ChangeTimes = 0;
 for(j=0;j<M;j++)
  for(i=0;i<N;i++)
   DataShowEnable[j][i] = false;  // 初始化为false，表示没有要显示的数据

 for(i=0;i<M;i++)
 {
   count[i] = 0; //  大于等于BlockNum，表示块中没有数据，或需被替换掉
        // 所以经这样初始化（3 2 1），每次替换>=3的块，替换后计数值置1，
       // 同时其它的块计数值加1 ，成了（1 3 2 ），见下面先进先出程序段
 }
 for(i=0;i<N;i++) // 对有所数据操作
 {
  // 增加count
  for(j=0;j<M;j++)
   count[j]++;
  find = false; // 表示块中有没有该数据
  for(j=0;j<M;j++)
  {
   if( Block[j] == Data[i] )
```

```
    {
     find = true;
    }
   }
   if( find ) continue; // 块中有该数据，判断下一个数据
   // 块中没有该数据
   ChangeTimes++; // 缺页次数++

   if( (i+1) > M ) // 因为i是从0开始记，而M指的是个数，从1开始，所以i+1
   {
    //获得要替换的块指针
    temp = 0;
    for(j=0;j<M;j++)
    {
     if( temp < count[j] )
     {
      temp = count[j];
      point = j; // 获得离的最远的指针
     }
    }
   }
   else point = i;
   // 替换
   Block[point] = Data[i];

   count[point] = 0; // 更新计数值

   // 保存要显示的数据
   for(j=0;j<M;j++)
   {
    DataShow[j][i] = Block[j];
    DataShowEnable[i<M?(j<=i?j:i):j][i] = true; // 设置显示数据
   }
  }
  // 输出信息
  cout<< endl;
  cout<<"FIFO => "<< endl;
  DataOutput();
}
```

3. 最近未使用置换算法

```
void LRU()
{
 int i,j;
 bool find;
 int point;
 int temp; // 临时变量
 ChangeTimes = 0;
 for(j=0;j<M;j++)
  for(i=0;i<N;i++)
```

```
    DataShowEnable[j][i] = false;  // 初始化为false，表示没有要显示的数据
  for(i=0;i<M;i++)
  {
    count[i] = 0 ;
  }
  for(i=0;i<N;i++) // 对有所数据操作
  {
   // 增加count
   for(j=0;j<M;j++)
    count[j]++;
   find = false; // 表示块中有没有该数据
   for(j=0;j<M;j++)
   {
    if( Block[j] == Data[i] )
    {
     count[j] = 0;
     find = true;
    }
   }
   if( find ) continue; // 块中有该数据，判断下一个数据
   // 块中没有该数据
   ChangeTimes++; // 缺页次数++
   if( (i+1) > M ) // 因为i是从0开始记，而BlockNum指的是个数，从1开始，所以i+1
   {
    //获得要替换的块指针
    temp = 0;
    for(j=0;j<M;j++)
    {
     if( temp < count[j] )
     {
      temp = count[j];
      point = j; // 获得离的最远的指针
     }
    }
   }
   else point = i;
   // 替换
   Block[point] = Data[i];
   count[point] = 0;

   // 保存要显示的数据
   for(j=0;j<M;j++)
   {
    DataShow[j][i] = Block[j];
    DataShowEnable[i<M?(j<=i?j:i):j][i] = true; // 设置显示数据
   }

  }
  // 输出信息
  cout<< endl;
  cout<<"LRU => "<< endl;
  DataOutput();
}
```

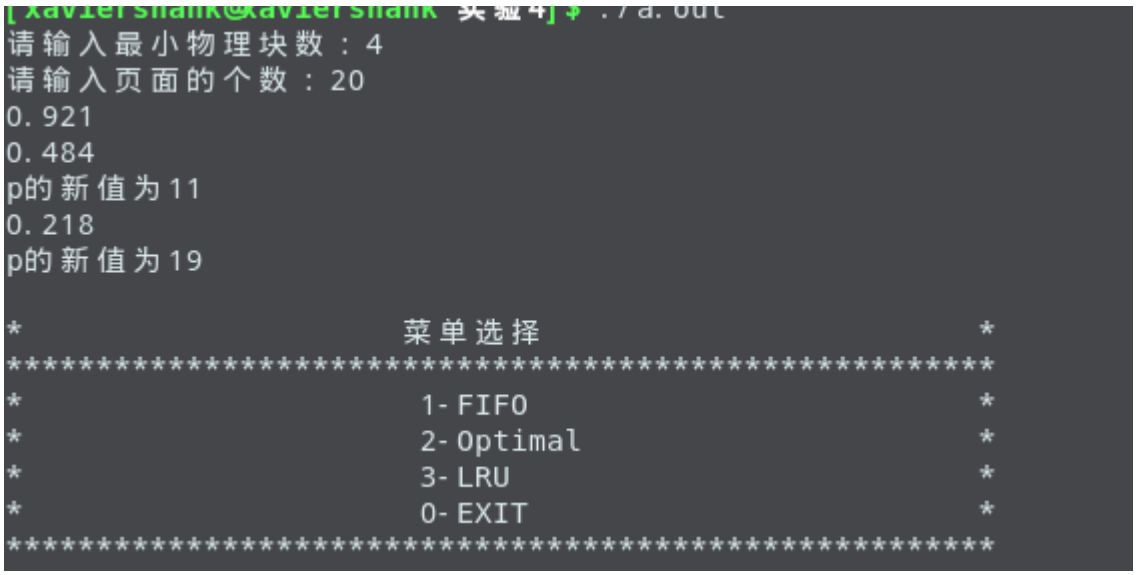
4. 改进型Clock算法

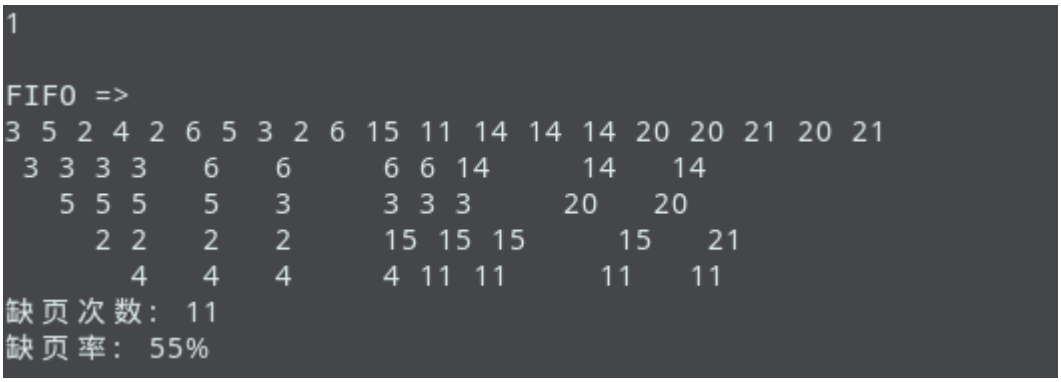
在将一个页面换出时，如果该页已被修改过，便须将该页重新写回到磁盘上；但如果该页未被修改过，则不必将它拷回磁盘。在改进型Clock算法中，除须考虑页面的使用情况外，还须在增加一个因素，即置换代价，这样页面换出时，既要是未使用过的页面，又要是未被修改过的页面。把同时满足这两个条件的页面作为首选淘汰的页面。由访问位A和修改位M可以组合成下面四种类型的页面: 1类（A=0，M=0）：表示该页最近既未被访问，又未被修改，是最佳淘汰页。 2类（A=0，M=0）：表示该页最近未被访问，但已被修改，并不是很好的淘汰页。 3类（A=1，M=0）：表示该页最近已被访问，但未被修改，该页有可能在被访问。 4类（A=1，M=1）：表示该页最近已被访问且被修改，该页可能再被访问。

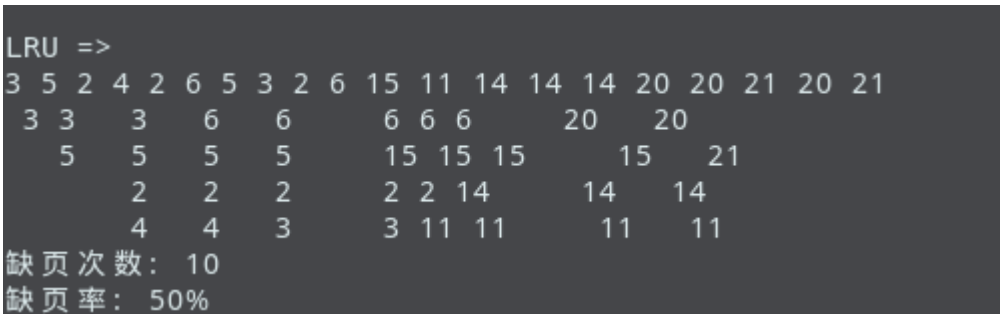
```
oid LCLOCK(int num){
    int j;

    if(isInside2(num)){
        cout<<"命中"<<endl;
        for(int i=0 ; i <A; i++)

        cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;
    }
    else
        if(count == A){
            lost++;
            j =whichpage();
            Inside[j] = Page[num];
            state2[j][0] = 1;
            for(int i=0 ; i <A; i++)

           cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;

        }

        else{
            Inside[count] = Page[num];
            count++;
            for(int i=0 ; i <A; i++)
            cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;
        }
}
```

五、实验结果截图

1. OPT,FIFO,LRU 实验结果





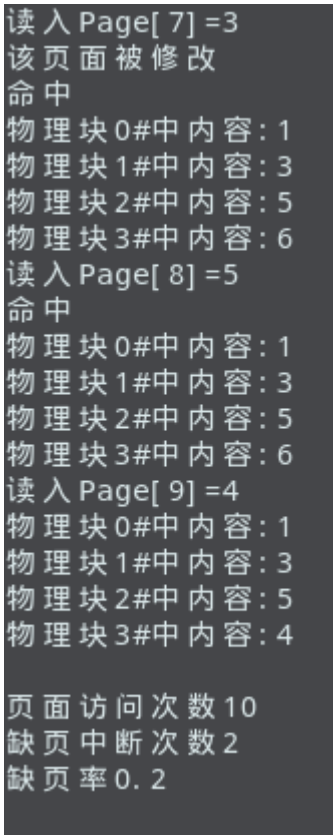


2. 改进型Clock算法截图

内含有一般的clock算法运行过程

```
1 1 2 5 5 6 3 3 5 4
------------1. Clock置换算法（CLOCK）-----
------------2. 改进型Clock置换算法--------
------------0. 退出------------------------
-------------输入进行选择-----------------
```

```
2
读入Page[ 0] =1
物理块0#中内容: 1
物理块1#中内容: 0
物理块2#中内容: 0
物理块3#中内容: 0
读入Page[ 1] =1
该页面被修改
命中
物理块0#中内容: 1
物理块1#中内容: 0
物理块2#中内容: 0
物理块3#中内容: 0
读入Page[ 2] =2
物理块0#中内容: 1
物理块1#中内容: 2
物理块2#中内容: 0
物理块3#中内容: 0
读入Page[ 3] =5
物理块0#中内容: 1
物理块1#中内容: 2
物理块2#中内容: 5
物理块3#中内容: 0
读入Page[ 4] =5
该页面被修改
命中
物理块0#中内容: 1
物理块1#中内容: 2
物理块2#中内容: 5
物理块3#中内容: 0
读入Page[ 5] =6
物理块0#中内容: 1
物理块1#中内容: 2
物理块2#中内容: 5
物理块3#中内容: 6
读入Page[ 6] =3
物理块0#中内容: 1
物理块1#中内容: 3
物理块2#中内容: 5
物理块3#中内容: 6
读入Page[ 7] =3
该页面被修改
命中
```



六、 完整源代码

1. OPT,FIFO,LRU 源代码

```
#include<iostream>
const int DataMax = 64;
const int BlockNum = 32;
int DataShow[BlockNum][DataMax]; // 用于存储要显示的数组
bool DataShowEnable[BlockNum][DataMax]; // 用于存储数组中的数据是否需要显示
int Data[DataMax]; // 保存数据
int Block[BlockNum]; // 物理块
int count[BlockNum]; // 计数器
int N ; // 页面个数
int M;//最小物理块数
int ChangeTimes;
void DataCreate();
void DataOutput();
void FIFO(); // FIFO 函数
void Optimal(); // Optimal函数
void LRU(); // LRU函数
void Clock();
#define Q 999
using namespace std;

///*
int main(int argc, char* argv[])
{
   DataCreate();//  DataInput();
```

```
  // FIFO();
  // Optimal();
 //  LRU();
 //  return 0;
      int menu;
     while(true)
     {
       cout<<endl;
       cout<<"*                      菜单选择                        *"<<endl;
       cout<<"*******************************************************"<<endl;
       cout<<"*                       1-FIFO                         *"<<endl;
       cout<<"*                       2-Optimal                      *"<<endl;
       cout<<"*                       3-LRU                          *"<<endl;
       cout<<"*                       0-EXIT                         *"<<endl;
       cout<<"*******************************************************"<<endl;
       cin>>menu;

       switch(menu)
       {
       case 1: FIFO();break;
       case 2: Optimal();break;
       case 3: LRU();break;
       if(menu!=1&&menu!=2&&menu!=3) break;
     }
}
}









//*/
void DataOutput()
{
 int i,j;
 for(i=0;i<N;i++) // 对所有数据操作
 {
  cout<<Data[i]<<" ";
 }
 cout<<endl;
 for(j=0;j<M;j++)
 {
  cout<<" ";
  for(i=0;i<N;i++) // 对所有数据操作
  {
   if( DataShowEnable[j][i] )
    cout<<DataShow[j][i]<<" ";
   else
    cout<<"  ";
  }
```

```
  cout<<endl;
 }
 cout<<"缺页次数: "<<ChangeTimes<<endl;
 cout<<"缺页率: "<<ChangeTimes*100/N<<"%"<<endl;
}

void DataCreate()
{
   cout<<"请输入最小物理块数：";
 cin>>M;
 while(M > BlockNum) // 大于数据个数
 {
  cout<<"物理块数超过预定值，请重新输入：";
  cin>>M;
 }
 cout<<"请输入页面的个数：";
 cin>>N;
 while(N > DataMax) // 大于数据个数
 {
  cout<<"页面个数超过预定值，请重新输入：";
  cin>>N;
 }
    int p = 1;
    int e = 4;
    int m = 5;//处理4个页面后起始位置P+1
    float t = 0.637;
    float r;
    int j = 0;

  srand((int)time(NULL));//用time(0)的返回值当种子
    for(int o=0;o<(N/m);o++)
    {
        for(int i=0;i<m;i++)
        {

            Data[j] = rand() % (e+1) + p;//生成p和p+e之间的5个随机数
            j++;

            if(j == N) break;//退出循环
            else if(i == 4)
         {

             //生成随机数r
            r = rand() % (Q+1) /(float)(Q+1);
            cout<<r<<endl;

            if(r<t)
            {
                p = rand() % N +1;
                cout<<"p的新值为"<<p<<endl;
            }

            else
```

```
            {
                p = (p+1) % M;
            }
        }

    }
    }

}
void FIFO()
{
 int i,j;
 bool find;
 int point;
 int temp; // 临时变量
 ChangeTimes = 0;
 for(j=0;j<M;j++)
  for(i=0;i<N;i++)
   DataShowEnable[j][i] = false;  // 初始化为false，表示没有要显示的数据

 for(i=0;i<M;i++)
 {
   count[i] = 0; //  大于等于BlockNum，表示块中没有数据，或需被替换掉
        // 所以经这样初始化（3 2 1），每次替换>=3的块，替换后计数值置1，
       // 同时其它的块计数值加1 ，成了（1 3 2 ），见下面先进先出程序段
 }
 for(i=0;i<N;i++) // 对有所数据操作
 {
  // 增加count
  for(j=0;j<M;j++)
   count[j]++;
  find = false; // 表示块中有没有该数据
  for(j=0;j<M;j++)
  {
   if( Block[j] == Data[i] )
   {
    find = true;
   }
  }
  if( find ) continue; // 块中有该数据，判断下一个数据
  // 块中没有该数据
  ChangeTimes++; // 缺页次数++

  if( (i+1) > M ) // 因为i是从0开始记，而M指的是个数，从1开始，所以i+1
  {
   //获得要替换的块指针
   temp = 0;
   for(j=0;j<M;j++)
   {
    if( temp < count[j] )
    {
     temp = count[j];
     point = j; // 获得离的最远的指针
```

```
    }
   }
  }
  else point = i;
  // 替换
  Block[point] = Data[i];

  count[point] = 0; // 更新计数值

  // 保存要显示的数据
  for(j=0;j<M;j++)
  {
   DataShow[j][i] = Block[j];
   DataShowEnable[i<M?(j<=i?j:i):j][i] = true; // 设置显示数据
  }
 }
 // 输出信息
 cout<< endl;
 cout<<"FIFO => "<< endl;
 DataOutput();
}
void Optimal()
{
 int i,j,k;
 bool find;
 int point;
 int temp; // 临时变量，比较离的最远的时候用
 ChangeTimes = 0;
 for(j=0;j<M;j++)
  for(i=0;i<N;i++)
   DataShowEnable[j][i] = false;  // 初始化为false，表示没有要显示的数据
// for(i=0;i<M;i++)
// {
  // count[i] = 0 ; //
// }
 for(i=0;i<N;i++) // 对有所数据操作
 {
  find = false; // 表示块中有没有该数据
  for(j=0;j<M;j++)
  {
   if( Block[j] == Data[i] )
    find = true;
  }
  if( find ) continue; // 块中有该数据，判断下一个数据
  // 块中没有该数据，最优算法
  ChangeTimes++; // 缺页次数++
  for(j=0;j<M;j++)
  {
   // 找到下一个值的位置
   find = false;
   for( k =i;k<N;k++)
   {
    if( Block[j] == Data[k] )
```

```
    {
     find = true;
     count[j] = k;
     break;
    }
   }
   if( !find ) count[j] = N;
  }
  if( (i+1) > M ) // 因为i是从0开始记，而BlockNum指的是个数，从1开始，所以i+1
  {
   //获得要替换的块指针
   temp = 0;
   for(j=0;j<M;j++)
   {
    if( temp < count[j] )
    {
     temp = count[j];
     point = j; // 获得离的最远的指针
    }
   }
  }
  else point = i;
  // 替换
  Block[point] = Data[i];

  // 保存要显示的数据
  for(j=0;j<M;j++)
  {
   DataShow[j][i] = Block[j];
   DataShowEnable[i<M?(j<=i?j:i):j][i] = true; // 设置显示数据
  }

 }
 // 输出信息
 cout<< endl;
 cout<<"Optimal => "<< endl;
 DataOutput();
}

void LRU()
{
 int i,j;
 bool find;
 int point;
 int temp; // 临时变量
 ChangeTimes = 0;
 for(j=0;j<M;j++)
  for(i=0;i<N;i++)
   DataShowEnable[j][i] = false;  // 初始化为false，表示没有要显示的数据
 for(i=0;i<M;i++)
 {
   count[i] = 0 ;
 }
```

```
 for(i=0;i<N;i++) // 对有所数据操作
 {
  // 增加count
  for(j=0;j<M;j++)
   count[j]++;
  find = false; // 表示块中有没有该数据
  for(j=0;j<M;j++)
  {
   if( Block[j] == Data[i] )
   {
    count[j] = 0;
    find = true;
   }
  }
  if( find ) continue; // 块中有该数据，判断下一个数据
  // 块中没有该数据
  ChangeTimes++; // 缺页次数++
  if( (i+1) > M ) // 因为i是从0开始记，而BlockNum指的是个数，从1开始，所以i+1
  {
   //获得要替换的块指针
   temp = 0;
   for(j=0;j<M;j++)
   {
    if( temp < count[j] )
    {
     temp = count[j];
     point = j; // 获得离的最远的指针
    }
   }
  }
  else point = i;
  // 替换
  Block[point] = Data[i];
  count[point] = 0;

  // 保存要显示的数据
  for(j=0;j<M;j++)
  {
   DataShow[j][i] = Block[j];
   DataShowEnable[i<M?(j<=i?j:i):j][i] = true; // 设置显示数据
  }

 }
 // 输出信息
 cout<< endl;
 cout<<"LRU => "<< endl;
 DataOutput();
}
```

2. 改进型Clock源代码

```
#include<iostream>
#include<stdlib.h>

using namespace std;
#define M 2
#define Q 999
void DataCreate();
int const A = 4;//内存中存放的页面数
int count = 0;
int Inside[A];
int const PageCount =10;//总的页面数
int Page[PageCount];
int insert = 0;//先到先出置换算法fcfo中表示 当内存满的时候，新进入的页号放的位置
int suiji = 0; //随机置换算法randchange  当内存满的时候，新进入的页号放的位置
int state[A];//clock置换算法中，内存中的每个页面号对应的状态
int state2[A][M];// 二维数组，第一行第一列为访问位，第一行的第二列为修改位
double lost = 0.0;

void DataCreate()
{

    int p = 1;
    int e = 4;
    int m = 5;//处理4个页面后起始位置P+1
    float t = 0.637;
    float r;
    int j = 0;

  srand((int)time(NULL));//用time(0)的返回值当种子
    for(int o=0;o<(PageCount/m);o++)
    {
        for(int i=0;i<m;i++)
        {

            Page[j] = rand() % (e+1) + p;//生成p和p+e之间的5个随机数
            j++;
            if(j == PageCount) break;//退出循环
            else if(i == 4)
         {

             //生成随机数r
            r = rand() % (Q+1) /(float)(Q+1);
            cout<<r<<endl;

            if(r<t)
            {
                p = rand() % PageCount +1;
                cout<<"p的新值为"<<p<<endl;
            }

            else
            {
                p = (p+1) % A;
```

```
            }
        }

    }
    }

}

//检测页号是否在内存中
bool isInside(int num){
    for(int i = 0; i < A; i++){
        if(Inside[i] == Page[num]){
            state[i] = 1;
            return true;
        }
    }
    return false;
}

//判断页面是否已经被修改
bool change(){
    if((rand()%2+1) == 1 ){
        cout<<"该页面被修改"<<endl;
        return true;
    }
    else
        return false;
}

//用于改进型clock置换算法，检测页号是否在内存中并把访问位和修改位置1
bool isInside2(int num){
    for(int i = 0; i < A; i++){
        if(Inside[i] == Page[num]){
            if(change()){
                state2[i][0] = 1;
                state2[i][1] = 1;
            }
            else{
                state2[i][0] = 1;
            }
            return true;
        }
    }
    return false;
}

//用于改进型clock置换算法，判断内存中第几个需要被置换
int whichpage(){
    int j;

    for(j=0; j < A;j++){
        if(state2[j][0] == 0&&state2[j][1] == 0){
            return j;
```

```
        }
    }
    for(j=0; j < A;j++ ){
        if(state2[j][0] == 0&&state2[j][1] == 1){
            return j;
        }
        state2[j][0] = 0 ;
    }
    for(j=0; j < A;j++ ){
        state2[j][0] = 0 ;
    }
    return whichpage();
}


//简单Clock置换算法
void CLOCK(int num){
    int j;

    if(isInside(num)){
        cout<<"命中"<<endl;
        for(int i=0 ; i <A; i++)
        cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;
    }
    else
        if(count == A){
            lost++;
            for(j=0; j < A; ){
                if(state[j] == 0){
                    break;
                }
                else{
                    state[j] = 0;
                }
                j++;
                j = j %3;
            }
            Inside[j] = Page[num];
            state[j] = 1;
            for(int i=0 ; i <A; i++)
            cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;
        }
        else{
            Inside[count] = Page[num];
            count++;
            for(int i=0 ; i <A; i++)
            cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;
        }
}

//改进型clock置换算法
void LCLOCK(int num){
    int j;
```

```
    if(isInside2(num)){
        cout<<"命中"<<endl;
        for(int i=0 ; i <A; i++)

        cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;
    }
    else
        if(count == A){
            lost++;
            j =whichpage();
            Inside[j] = Page[num];
            state2[j][0] = 1;
            for(int i=0 ; i <A; i++)

           cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;

        }

        else{
            Inside[count] = Page[num];
            count++;
            for(int i=0 ; i <A; i++)
            cout<<"物理块"<<i<<"#中内容:"<<Inside [i]<<endl;
        }
}

int main(){
  char ch ;
  DataCreate();
  for(int i = 0; i < PageCount; i++){
        cout<<Page[i]<<" ";
    }
  cout<<endl;
  while(1){
    cout<<"------------1.Clock置换算法（CLOCK）-----"<<endl;
    cout<<"------------2.改进型Clock置换算法--------"<<endl;
    cout<<"------------0.退出-----------------------"<<endl;
    cout<<"-------------输入进行选择-----------------"<<endl;
    cin>>ch;
    switch(ch){
    case '1':{
            lost = 0;
            count = 0;
            for(int m = 0; m < A; m++){
                state[m] = 0;
                }
                for(int j = 0; j < A; j++){
                    Inside[j] = 0;
                }
                for(int i = 0; i < PageCount; i++){
                    cout<<"读入Page["<<i<<"]="<<Page[i]<<endl;
                    CLOCK(i);
```

```
                }
                cout<<"\n页面访问次数"<<PageCount<<"\n缺页中断次数"<<lost<<"\n缺页率"
<<lost/(PageCount)<<"\n"<<endl;
    }break;
    case '2':{
            lost = 0;
            count = 0;
            for(int m = 0; m < A; m++){
                for(int n = 0; n < 2;n++)
                state2[m][n] = 0;
            }
            for(int j = 0; j < A; j++){
                Inside[j] = 0;
            }
            for(int i = 0; i < PageCount; i++){
                cout<<"读入Page["<<i<<"]="<<Page[i]<<endl;
                LCLOCK(i);
            }
            cout<<"\n页面访问次数"<<PageCount<<"\n缺页中断次数"<<lost<<"\n缺页率"
<<lost/(PageCount)<<"\n"<<endl;
    }break;
    case '0':{
        exit(0);
             }break;
    }
  }
  return 0;
}
```