# 实验七、内存页面置换算法实验

## 7.1 实验目的

加深对于存储管理的了解，掌握虚拟存储器的实现原理；观察和了解重要的页面置换算法和置换过程。练习模拟算法的编程技巧，锻炼分析试验数据的能力。

## 7.2 实验说明

1. 示例实验程序中模拟两种置换算法：LRU 算法和 FIFO 算法
2. 能对两种算法给定任意序列不同的页面引用串和任意帧实内存块数的组合测试，显示页置换的过程。
3. 能统计和报告不同置换算法情况下依次淘汰的页号、缺页次数（页错误数）和缺页率。比较两种置换算法在给定条件下的优劣。
4. 为了能方便的扩充页面置换算法，更好的描述置换过程，示例实验程序采用了 C++语言用 Replace 类描述了置换算法及其属性。

## 7.3 示例实验

**1**）在新建文件夹中建立以下 **vmrp.h** 文件：

```
/*
   *    Filename              :    vmrp.h
   *    copyright             : (C) 2006 by zhonghonglie
    * Function               : 声明虚拟内存页置换类
*/
#include <iostream>
#include <iomanip.h>
#include <malloc.h>

class Replace{
public:
      Replace();
      ~Replace();
      void InitSpace(char * MethodName); //初始化页号记录
      void Report(void); //  报告算法执行情况
      void Fifo(void); //先进先出算法
      void Lru(void);    //最近最旧未用算法
      void Clock(void); //时钟(二次机会）置换算法
      void Eclock(void); //增强二次机会置换算法
      void Lfu(void); //最不经常使用置换算法
      void Mfu(void);    //最经常使用置换算法
private:
      int * ReferencePage ; //存放要访问到的页号
      int * EliminatePage ; //存放淘汰页号
```

```
    int * PageFrames ;      //存放当前正在实存中的页号
    int PageNumber;          //访问页数
    int FrameNumber;         //实存帧数
    int FaultNumber;         //失败页数
};
```

**2**）在新建文件夹中建立以下 **vmrp.cc** 文件：

```
/*
   *    Filename              :    vmrp.cc
   *    copyright             : (C) 2006 by zhonghonglie
    * Function               : 模拟虚拟内存页置换算法的程序
*/
#include "vmrp.h"

Replace::Replace(){
  int i;
   //设定总得访问页数,并分配相应的引用页号和淘汰页号记录数组空间
   cout << "Please input page numbers :" ;
   cin >> PageNumber;
   ReferencePage = new int[sizeof(int) * PageNumber];
   EliminatePage = new int[sizeof(int) * PageNumber];

    //输入引用页号序列(页面走向),初始化引用页数组
   cout << "Please input reference page string :";
   for (i = 0; i < PageNumber; i++)
       cin >> ReferencePage[i];//引用页暂存引用数组

   //设定内存实页数(帧数),并分配相应的实页号记录数组空间(页号栈)
   cout << "Please input page frames :";
   cin >> FrameNumber;
   PageFrames = new int[sizeof(int) * FrameNumber];
}

Replace::~Replace(){

}

void Replace::InitSpace(char * MethodName)
{
   int i;
   cout << endl << MethodName << endl;
   FaultNumber=0;
   //引用还未开始,-1 表示无引用页
   for (i = 0; i < PageNumber; i++)
       EliminatePage[i] = -1;
   for(i = 0; i < FrameNumber; i++)
       PageFrames[i] = -1;
```

```
}

//分析统计选择的算法对于当前输入的页面走向的性能
void Replace::Report(void){
   //报告淘汰页顺序
   cout << endl << "Eliminate page:";
   for(int i=0; EliminatePage[i]!=-1; i++)    cout << EliminatePage[i] << " ";
   //报告缺页数和缺页率
   cout << endl << "Number of page faults = " << FaultNumber << endl;
   cout << setw(6) << setprecision(3) ;
   cout << "Rate of page faults = " << 100*(float)FaultNumber/(float)PageNumber <<
"%" <<endl;
}

//最近最旧未用置换算法
void Replace::Lru(void)
{
   int i,j,k,l,next;

   InitSpace("LRU");
   //循环装入引用页
   for(k=0,l=0; k < PageNumber; k++){
         next=ReferencePage[k];
         //检测引用页当前是否已在实存
         for (i=0; i<FrameNumber; i++){
           if(next == PageFrames[i]){
                //引用页已在实存将其调整到页记录栈顶
                next= PageFrames[i];
                for(j=i;j>0;j--)    PageFrames[j] =    PageFrames[j-1];
                PageFrames[0]=next;
                break;
                  }
         }

         if(PageFrames[0] == next){
              //如果引用页已放栈顶，则为不缺页，报告当前内存页号
              for(j=0; j<FrameNumber; j++)
                 if(PageFrames[j]>=0) cout << PageFrames[j] << " ";
              cout << endl;
              continue;   //继续装入下一页
         }
         else
               // 如果引用页还未放栈顶，则为缺页，缺页数加 1
               FaultNumber++;
         //栈底页号记入淘汰页数组中
         EliminatePage[l] = PageFrames[FrameNumber-1];
         //向下压栈
```

```
        for(j=FrameNumber-1;j>0;j--) PageFrames[j]= PageFrames[j-1];
        PageFrames[0]=next; //引用页放栈顶
        //报告当前实存中页号
        for(j=0; j<FrameNumber; j++)
            if(PageFrames[j]>=0) cout << PageFrames[j] << " ";
        //报告当前淘汰的页号
        if(EliminatePage[l]>=0)
            cout << "->" << EliminatePage[l++] << endl;
        else
       cout << endl;

   }
   //分析统计选择的算法对于当前引用的页面走向的性能
   Report();
}

//先进先出置换算法
void Replace::Fifo(void){
    int i,j,k,l,next;

    InitSpace("FIFO");
   //循环装入引用页
    for(k=0,j=l=0; k < PageNumber; k++){
        next=ReferencePage[k];
        //如果引用页已在实存中，报告实存页号
       for (i=0; i<FrameNumber; i++) if(next==PageFrames[i]) break;
        if (i<FrameNumber){
            for(i=0; i<FrameNumber; i++) cout << PageFrames[i] << " ";
        cout << endl;
        continue;   // 继续引用下一页
        }
        //引用页不在实存中，缺页数加 1
        FaultNumber++;
        EliminatePage[l]= PageFrames[j]; //最先入页号记入淘汰页数组
        PageFrames[j]=next;   //引用页号放最先入页号处
        j = (j+1)%FrameNumber; //最先入页号循环下移
        //报告当前实存页号和淘汰页号
        for(i=0; i<FrameNumber; i++)
            if(PageFrames[i]>=0) cout << PageFrames[i] << " ";
        if(EliminatePage[l]>=0)
            cout << "->" << EliminatePage[l++] << endl;
        else
        cout << endl;
   }
   //分析统计选择的算法对于当前引用的页面走向的性能
   Report();
}
```

```
//未实现的其他页置换算法入口
void Replace::Clock(void)
{

}

void Replace::Eclock (void)
{

}


void Replace::Lfu(void)
{

}

void Replace::Mfu(void)
{

}


int main(int argc,char *argv[]){

   Replace * vmpr = new Replace();

            vmpr->Lru();
            vmpr->Fifo();

   return 0;
}
```

**3**）在新建文件夹中建立以下 **Makefile** 文件

```
head = vmrp.h
srcs = vmrp.cc
objs = vmrp.o
opts = -w -g -c
all:   vmrp
vmrp:      $(objs)
            g++    $(objs) -o vmrp
vmrp.o:   $(srcs) $(head)
            g++ $(opts) $(srcs)
clean:
            rm vmrp *.o
```

**4)** 执行 **make** 命令编译连接，生成可执行文件 **vmpr**

```
$ gmake
   g++ -g -c vmrp.cc vmrp.h
   g++    vmrp.o -o vmrp
```

**5)** 执行 **vmpr** 命令，输入引用页数为 **12**，引用串为 **Belady** 串，内存页帧数为 **3**

```
$ ./vmpr

Please input reference page numbers :12
Please input reference page string :1 2 3 4 1 2 5 1 2 3 4 5
Please input page frames :3

FIFO
1
1 2
1 2 3
4 2 3 ->1
4 1 3 ->2
4 1 2 ->3
5 1 2 ->4
5 1 2
5 1 2
5 3 2 ->1
5 3 4 ->2
5 3 4

Eliminate page:1 2 3 4 1 2
Number of page faults = 9
Rate of page faults = 75%

LRU
1
2 1
3 2 1
4 3 2 ->1
1 4 3 ->2
2 1 4 ->3
5 2 1 ->4
1 5 2
2 1 5
3 2 1 ->5
4 3 2 ->1
5 4 3 ->2

Eliminate page:1 2 3 4 5 1 2
Number of page faults = 10
Rate of page faults = 83.3%
```

以上输出报告了FIFO和LRU两种算法的页置换情况。其中每一行数字为当前实存

中的页号，->右边的数字表示当前被淘汰的页号。每种算法最后 3 行输出为：依次淘汰页号，缺页数，页出错率。

**6)** 再次执行 **vmpr** 命令，仍然输入 **Belady** 串，仅将页帧数改为 **4**

```
 $ ./vmpr
Please input reference page numbers :12
Please input reference page string :1 2 3 4 1 2 5 1 2 3 4 5
Please input page frames :4

FIFO
1
1 2
1 2 3
1 2 3 4
1 2 3 4
1 2 3 4
5 2 3 4 ->1
5 1 3 4 ->2
5 1 2 4 ->3
5 1 2 3 ->4
4 1 2 3 ->5
4 5 2 3 ->1

Eliminate page:1 2 3 4 5 1
Number of page faults = 10
Rate of page faults = 83.3%

LRU
1
2 1
3 2 1
4 3 2 1
1 4 3 2
2 1 4 3
5 2 1 4 ->3
1 5 2 4
2 1 5 4
3 2 1 5 ->4
4 3 2 1 ->5
5 4 3 2 ->1

Eliminate page:3 4 5 1
Number of page faults = 8
Rate of page faults = 66.7%
```

从以上输出中可以看出 FIFO 置换算法的 Belady 异常现象，即当在相同的引用串下内存页帧数从 3 帧增加到 4 帧，页出错率反而从 75％增加到了 83.3%。而在相同的情况下 LUR 置换算法无此异常现象。

## 7.4 独立实验

请在以上示例实验程序中补充¡¥增强二次机会¡¦等置换算法的模拟程序。输入不同的内存页面引用串和实存帧数，观察并分析其页面置换效果和性能，并将其与 LRU 和 FIFO 算法进行比较。改进以上示例实验程序，使之能够随机的产生内存页面引用串，以便能动态的观测各种置换算法的性能。

## 7.5 实验要求

1. 说明您做了哪些不同的引用串在不同实存帧中的测试，发现了哪些现象？
2. 选择一些典型的现象作出不同算法中帧数与缺页数的曲线图。
3. 说明您的程序是怎样模拟¡¥增强二次机会¡¦等置换算法的？
4. 综合分析实验结果中各种页面置换算法各适应于怎样的页面引用串和和内存帧数。
5. 根据实验程序、调试过程和结果分析写出实验报告。