# 页面置换竞争实验学生文档



## 实验内容

经过操作系统理论课的学习，想必同学们已经了解了页面置换的相关知识。

*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。*

下面我们将模拟一个简单的页面置换场景，允许同学们设计自己的页面置换算法,并且对同学们的算法进行竞争排名。假设操作系统允许我们使用的物理页框数为 4 个，访问的虚拟页面依次为,Page1,Page7,Page8,Page9,Page10,Page8,那么当访问到 Page10 的时候就必须将当前一个物理页框中的内容换出，然后将需要的 Page10 换入，所以将产生中断，然后进行页面置换。在本页面置换实验模拟中，我们将考虑上述两部分的代价:

$$总代价 = 中断的代价 + 单页面置换代价 * 置换页面数$$

## 样例说明

以物理页框为 4 的情况为例，使用 clock 算法来进一步说明:

测评数据:

4095
4095
8192
16384
32768
4097
4095
32768
16384
8192

初始物理页框　physic_memery={-1,-1,-1,-1}

(1) 输入第一个数据(由评测机完成)4095,评测程序调用 pageReplace(physic_memery,4095)函数,物理页框修改为 physic_memery={0,-1,-1,-1},本次代价=中断代价+置换代价=1+2 = 3;

(2) 输入第二个数据 4095,调用 pageReplace(physic_memery,4095)函数，物理页框不变，本次代价=0+0;
(3) 输入地址 8192, physic_memery ={0,2,-1,-1}, cost+=(1+2*1)
(4) 输入地址 16384,physic_memery={0,2,4,-1},cost+=3
(5) 输入地址 32768,physic_memery={0,2,4,8},cost+=3
(6) 输入地址 4097, physic_memery={1,2,4,8}, cost+=3
(7) 输入地址 4095, physic_memery ={1,0,4,8}, cost+=3
(8) 输入地址 32768,physic_memery={1,0,4,8},cost+=0
(9) 输入地址 16384,physic_memery={1,0,4,8},cost+=0
Cost = 18

假设在上述例子中可以实现某种预调度
(1) 输入 4095, physic_memery={0,2,4,8}, cost+=(1+2*4)
(2) 输入 4095, physic_memery={0,2,4,8}, cost+=0
(3) 输入地址 8192, physic_memery ={0,2,4,8}, cost+=0
(4) 输入地址 16384,physic_memery={0,2,4,8},cost+=0
(5) 输入地址 32768,physic_memery={0,2,4,8},cost+=0
(6) 输入地址 4097, physic_memery={1,2,4,8}, cost+=3
(7) 输入地址 4095, physic_memery ={1,0,4,8}, cost+=3
(8) 输入地址 32768,physic_memery={1,0,4,8},cost+=0
(9) 输入地址 16384,physic_memery={1,0,4,8},cost+=0
Cost = 15

学生实现代码样例:

```
#include "pageReplace.h" // 测评需求，请务必包含该头文件
#define MAX_PHY_PAGE 8 // 这里只使用了8个物理页框
#define MAX_PAGE 12
#define get_Page(x) (x>>MAX_PAGE)
void pageReplace(long    * physic_memery, long    nwAdd) {
        int flag = 0;
        static int clock = 0;
        for (int i = 0; i < MAX_PHY_PAGE; i++) {
                if ((nwAdd >> MAX_PAGE) == physic_memery[i]) {
                        return;
                }
        }
        for (int i = 0; i < MAX_PHY_PAGE; i++) {
                if (physic_memery[i] == 0) {
                        physic_memery[i] = get_Page(nwAdd);
                        flag = 1;
                        break;
                }
        }
        if (flag == 0) physic_memery[(clock++) % MAX_PHY_PAGE] = get_Page(nwAdd);
}
```

## 实验要求:

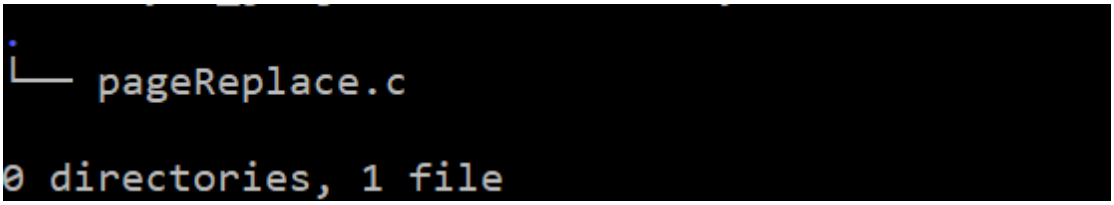
1、 物理页框数量为 **64**（页框编号 0~63），要求初始页表为空.页面大小为 4KB
2、 实现页面置换算法，函数命名为 void pageReplace (long[] physic_memery, int nwAdd)（不需要实现 main 函数），该函数需要接收物理页框指针 physic_memery(初始物理页框为空)。其中物理页框为一个数组,physic_memery[p] = v,表示当前物理页框 physic_memery 的第 p 个页框对应虚拟页 v。nwAdd 表示当前程序访问的虚拟地址，执行页面置换之后，该虚拟地址所在的虚拟页必须在物理页框中有对应。
3、 要求学生实现的函数被调用时接收传入的虚拟地址，并且立即执行分配，修改 physic_memery 数组,评测会检测当前使用的虚拟地址是否在物理页框中有对应。
4、 内存空间限制 128M。pageReplace 函数总执行时间限制为 300s(共有约 166 万条使用地址，所以会调用约 166 万次该函数)。测试地址由**真实程序使用地址**和**模拟生成的程序使用地址**组成。
5、 评分会综合**程序运行时间**和**总置换代价**给出。中断的代价为 1,单页面置换代价为 2。

$$\text{Score} = \frac{\text{Stu}_{\text{time}} - Stu_{\text{max}time}}{Stu_{mintime} - Stu_{\text{max}\text{time}}} \times 0.4 + \frac{\text{Stu}_{cost} - Stu_{\text{max}cost}}{Stu_{mincost} - Stu_{\text{max}\text{cost}}} \times 0.6$$

6、 每名同学每天只有 **3** 次提交机会,以服务器时间为准，并且**仅保留最近一次测评结果。**
7、 参与竞争实验的同学需要完成一份**竞争实验的相关实验报告**，发送至助教邮箱 16231092@buaa.edu.cn，实验报告中要求至少包括:算法设计思想、算法实现技巧、以及竞争实验过程中的优化与改进、本地测试情况等内容。
8、 竞争实验取最终排名前 30%的同学作为优秀作业，视为完成一项挑战性任务(竞争实验结束时间为答辩前两天),并且可以进行申优答辩。

## 提交步骤

1. 创建 racing 分支 git branch racing
2. 切换到 racing 分支 git checkout racing
3. 实现 pageReplace 函数,并且保存到 pageReplace.c 文件(评测程序只识别”pageReplace.c”).



```c
#define MAX_PHY_PAGE 8
#define MAX_PAGE 12
#define getPage(x) (x>>MAX_PAGE)
#include "pageReplace.h"
#include <unistd.h>
void pageReplace(long  * physic_memery, long  nwAdd) {
        int flag = 0;
        //for(int i = 0;i<40000;i++);
        static int clock = 0;
        for (int i = 0; i < MAX_PHY_PAGE; i++) {
                if ((nwAdd >> MAX_PAGE) == physic_memery[i]) {
                        return;
                }
        }
        for (int i = 0; i < MAX_PHY_PAGE; i++) {
                if (physic_memery[i] == 0) {
                        physic_memery[i] = getPage(nwAdd);
                        flag = 1;
                        break;
                }
        }
        if (flag == 0) {
                physic_memery[(clock++) % MAX_PHY_PAGE] = getPage(nwAdd);
        }
}
```

4. 提交分支 git push origin racing:racing

```
remote:
remote: [ You are changing the branch: refs/heads/racing. ]
remote:
remote: Autotest: Begin at Sat May 25 22:15:19 CST 2019
remote:
remote: perl: warning: Setting locale failed.
remote: perl: warning: Please check that your locale settings:
remote:         LANGUAGE = (unset),
remote:         LC_ALL = (unset),
remote:         LC_PAPER = "zh_CN",
remote:         LC_ADDRESS = "zh_CN",
remote:         LC_MONETARY = "zh_CN",
```

提交评测后首先会返回编译信息和本日提交次数
(如果提交次数超过限制，将不会触发测评)
(评测使用 g++进行编译，编译错误不计入评测次数)

```
remote: warning: remote HEAD refers to nonexistent ref, unable to checkout.
remote:
remote: Switched to a new branch 'racing'
remote: Branch racing set up to track remote branch racing from origin.
remote: 1
remote: [ you have upload 1 times ]
remote: [ compile successfully ]
```

测评会返回调度代价、调度函数执行总时间、分数、以及当前排名信息。
(测评分数 100(of 100)在本次测评中是没有意义的)


## 测试集更新说明

为防止出现过拟合的现象，鼓励同学们自己创造新的测试集并且进行测试，比如可以针对一个程序输出其访问变量的地址来模拟程序使用的虚拟地址。比如我们可以使用简单的矩阵乘法来获取某程序运行的访问地址（直接输出变量地址）。

在本竞争实验中鼓励同学们“分享”自己的测试用例，将**类似下面程序的源程序**发送到助教邮箱(16231092@buaa.edu.cn)，如果该样例程序比较有代表性，则可能**增加到现有的测评数据集**中。同时为了公平起见，防止部分同学代码对测试集过拟合，我们也将在竞速排名结束一周前，对模拟生成的随机数据更换随机种子重新生成。

(下为一个简单的矩阵乘法程序，打印其变量地址来模拟地址使用情况)

```
#include<iostream>
#include<cstdio>
#include<cstring>
#define test(x) printf("%d\n",&(x))
#include <cstdlib>
#include<time.h>
using namespace std;
int num = 0;
int index = 0;
int key = 8000;
int *myArr;
int random[10000];
const int maxn = 105;
struct Matrix{
    int n,m;
    int v[maxn][maxn];
    Matrix(int n,int m):n(n),m(m){}
    void init(){                    //初始化矩阵
        memset(v,0,sizeof v);
    }
    Matrix operator* (const Matrix B) const {
```

```
            Matrix C(n,B.m);
            C.init();
            for(int i = 0;i < n;i++)
            for(int j = 0;j < B.m;j++)
            for(int k = 0;k < m;k++){
                test(i);
                test(i);
                test(i);
                test(j);
                test(j);
                test(j);
                test(B.m);

                test(k);
                test(k);
                test(k);
                test(m);

                test(i);
                test(j);
                test(i);
                test(j)
                test(k);
                test(k);
                test(C.v[i][j]);
                test(B.v[k][j]);
                test(v[i][k]);

                C.v[i][j] += v[i][k]*B.v[k][j];
        }
            return C;
        }
        void print(){//用于测试输出

            for(int i = 0;i < n;i++){
            for(int j = 0;j < m;j++)
                cout << v[i][j] << " ";
            cout << endl;
            }
        }
};
int main(){
        srand((int)time(0));
        freopen("case2.txt", "w", stdout);
```

```
    int n1 = 40,m1 =40 ,n2 = 40,m2 = 40;
    Matrix A(n1,m1);
    for(int i = 0;i < n1;i++)
        for(int j = 0;j < m1;j++){
            test(i);
            test(j);
            test(i);
            test(j);
            test(i);
            test(j);
            test(n1);
            test(m1);
            test(A.v[i][j]);
            A.v[i][j]= (int)rand();
        }

    Matrix B(n2,m2);
    for(int i = 0;i < n2;i++)
        for(int j = 0;j < m2;j++){
            B.v[i][j]= (int)rand();
            test(i);
            test(j);
            test(i);
            test(j);
            test(i);
            test(j);
            test(n2);
            test(m2);
            test(B.v[i][j]);
        }
    Matrix C = A*B;
    printf("-1\n");
    return 0;
}
```