页面置换竞争性实验

一、实验介绍

操作系统理论课介绍了页面置换的相关知识。我们来回顾一下：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。

我们在lab4中也实现了处理缺页中断的过程。但是，lab4中的页面置换算法是由硬件系统实现的，并不需要我们干预。lab2的挑战性任务中也有一个页面置换，但那只是实现，并没有进行竞速排名。下面我们将模拟一个简单的页面置换场景，对同学们的算法进行竞争排名。大家在数据结构大作业中也玩过一次竞速排名，惊不惊险，刺不刺激？这次让大家再体验一把坐过山车的乐趣！

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Page10 换入。所以，先产生缺页中断，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。**

二、评测样例

1、样例说明

以物理页框为 4 的情况为例。

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仅用于样例说明，在实际实验中为方便起见，初始页框初始化为 0，并且**不会调用需要 0 号页面的虚拟地址**)

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

2、代码示例

```
// pageReplace.cpp
#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) { //初始物理页框内容全部为0

            physic_memery[i] = get_Page(nwAdd);
            flag = 1;
            break;
        }
    }
    if (flag == 0)
        physic_memery[(clock++) % MAX_PHY_PAGE] =
get_Page(nwAdd);
}
```

三、实验要求

- 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。
- 实现页面置换算法，函数命名为 `void pageReplace (long* physic_memery, long  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 `physic_memery`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（按字节编址），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。在实 际实验中，`nwAdd < INT_MAX`。

- 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。
- 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。
- 评分会综合程序运行时间和总置换代价给出。程序运行时间的权重为0.3，页面置换代价的权重为0.7。中断的代价为1，单页面置换代价为2。 打分公式如下：

$$Score = \frac{t - t_{max}}{t_{min} - t_{max}} * 0.3 + \frac{c - c_{max}}{c_{min} - c_{max}} * 0.7 \quad (1)$$

  其中t和c表示你的运行时间和置换代价；t_{max}和t_{min}分别表示当前所有学生提交中最长的和最短的运行时间，c_{max}和c_{min}分别表示当前所有学生提交中最大的和最小的置换代价。所以，不难看出，你的排名是在动态变化的。
- 该实验占用评测资源较大，为防止滥用评测机，**每名同学每天只有10次提交机会**，每天0时开始重新计算，以服务器时间为准，并且**仅保留最近一次测评结果**。但是助教会每三天在cscore上公布一次所有提交同学的排名信息，以供参考。
- 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 wancong@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**。推荐用markdown书写。
- 页面置换竞争性实验属于一项挑战性任务，可作为申优的条件之一。**原则上，以页面置换竞争性实验申优的同学，在竞争排名中不得低于前30%。**
- **页面置换竞争性实验的截止时间为4月27日17:00**

四、提交步骤

1、提交代码

1. 创建`racing-1`分支，可以用`git branch racing-1`
2. 切换到`racing-1`分支，`git checkout racing-1`
3. 清空当前目录，`git rm -rf .`
4. 在当前目录下创建`pageReplace.cpp`文件。此文件必须`#include "pageReplace.h"`，注意**双引号不要改成尖括号**！然后在`pageReplace.cpp`中实现`void pageReplace (long* physic_memery, long  nwAdd)`函数。评测机只识别该文件，其他任何文件对评测不起作用。

5. `git add --all, git commit -m “blabla”, git push origin racing-1:racing-1`。
6. 提交评测后首先会返回编译信息和当天提交次数。如果当天提交次数超过10次，将不会触发测评。**评测时间较长，约1-2分钟，请大家耐心等待。**
7. 编译环境：`g++ x86_64_linux_gnu, std=c++11`。

评测记录示例：

```
remote: ************************************************************************
remote:
remote:                      BUAA OSLAB AUTOTEST SYSTEM
remote:                     Copyright (c) BUAA 2015-2019
remote:
remote: ************************************************************************
remote:
remote: [ You are changing the branch: refs/heads/racing-1. ]
remote:
remote: Autotest: Begin at Sat Apr 11 19:43:22 CST 2020
remote:
remote: warning: remote HEAD refers to nonexistent ref, unable to checkout.
remote:
remote: Switched to a new branch 'racing-1'
remote: Branch racing-1 set up to track remote branch racing-1 from origin.
remote: [ find your pageReplace.cpp ]
remote: [ compile successfully ]
remote: 1
remote: [ you have upload 1 times ]
remote: [ your results are as follows ]
remote: your id is TNNDOB6OSPDE4
remote: your cost is 288072
remote: your time is 0.700475
remote: your score is 1.000000
remote: your rank is 1 of 1
remote: [ You got 100 (of 100) this time. Sat Apr 11 19:44:21 CST 2020 ]
remote:
remote:
remote: >>>>>> Collecting autotest results >>>>>>
```

2、查看当前排名的方法

1. 在 `pageReplace.cpp` **同路径下**创建 `getScore` 文件(文件内容可有可无)
2. `git add --all, git commit -m “blabla”, git push origin racing-1:racing-1`，提交后若检测到 getScore 文件，则不触发编译，只返回当前你的运行时间、代价和排名。
3. 正常提交评测时，务必将 getScore 文件删 除再提交。

显示排名示例：

```
remote: ************************************************************
remote:
remote:                    BUAA OSLAB AUTOTEST SYSTEM
remote:                  Copyright (c) BUAA 2015-2019
remote:
remote: ************************************************************
remote:
remote: [ You are changing the branch: refs/heads/racing-1. ]
remote:
remote: Autotest: Begin at Sat Apr 11 19:47:13 CST 2020
remote:
remote: warning: remote HEAD refers to nonexistent ref, unable to checkout.
remote:
remote: Switched to a new branch 'racing-1'
remote: Branch racing-1 set up to track remote branch racing-1 from origin.
remote: [ find your getScore ]
remote: [ your nearest results are as follows ]
remote: your id is TNNDOB6OSPDE4
remote: your cost is 288072
remote: your time is 0.700475
remote: your score is 1.000000
remote: your rank is 1 of 1
remote: [ You got 100 (of 100) this time. Sat Apr 11 19:47:14 CST 2020 ]
```

五、测试集更新和征集说明

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。使用一个程序来体现内存的实际使用情况，并将内存调用情况输出到文件中（也就是**编写一个数据生成器**）。例如这个代码（matrix_mult.cpp，提取码：6s9t）就是一个典型的例子，它输出矩阵乘法中内存调用的过程。

在本竞争实验中鼓励同学们“分享”自己的测试用例，将数据生成器**源程序（请务必保证编译通过，运行正确）**发送到助教邮箱 wancong@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。被选中评测数据的同学可酌情加分。

数据生成器输出的要求：

- 输出到文件`case-学号.txt`中，不要输出到标准输出
- 源程序用C++编写，命名为`generate-学号.cpp`
- 输出文件的格式为：一行一个虚拟地址，最后一行为-1，且虚拟地址必须在`[4096, 2147483647]`之间，如下：

```
6355260
6355256
4184
2147483647
1926817
19260817
6487608
6443496
19260817
6355256
6355260
19260817
-1
```

- 同时附上一个README，说明该数据生成器生成的数据的特点（**单纯的随机数生成器一律拒绝**）

同时为了公平起见，防止部分同学“投机取巧”，我们也将在竞速排名结束之后，对模拟生成的随机数据更换随机种子重新生成，并以**更换之后的数据对所有同学的代码进行多次重测**，取运行时间和代价的平均值参与最终排名。