操作系统原理课程设计实验报告

学生姓名	王云鹏
学生学号	8213180228
指导教师	胡小龙
专业班级	物联网 1802
完成日期	2020.7.11

计算机学院

目　录

实验

一、目的与要求

题目：页面置换算法模拟

目的：

1、 增强学生对计算机操作系统基本原理、基本理论、基本算法的理解

2、 提高和培养学生的动手能力

要求：

1、 每人至少选作 1 题，多做不限。

2、 每人单独完成，可以讨论，但每人的设计内容不得完全相同，抄袭或有 2 人/多人设计完全一样者，不能通过。

3、 设计完成后，应上交课程设计文档，文档格式应是学校课程设计的标准格式，所有学生的封面大小、格式也必须一样

4、 同时上交设计的软盘(或以班刻录光盘)

二、操作环境

硬件：

软件：

Visual Studio Code

三、实验内容

```
int test[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};//测试序列
int test_size = sizeof(test) / sizeof(int);//测试序列大小
// int page_num=3;
// int memory[page_num]={0};

double hit_rate(int miss_num)//计算命中率…

int is_miss(int memory[], int size_of_memory, int order)//检查order是否在memory中，是则返回位置,否则返回-1…

void print_array(int memory[], int size_of_memory)//打印数组…

void init_memory(int memory[], int memory_size, int test[])//内存初始化，预装入…

int find_max_index(int array[], int array_size)//找到数组中最大数字，返回其下标…

class OPT//最佳置换算法…

class FIFO//先进先出算法…

class LRU//最近最久未使用算法…

class CLOCK//Clock算法…

int main()//测试…
```

四、实验结果

不同算法对应的内存中页号：

```
OPT内存中页号
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 0 3
2 0 3
2 4 3
2 4 3
2 4 3
2 0 3
2 0 3
2 0 3
2 0 1
2 0 1
2 0 1
2 0 1
7 0 1
7 0 1
7 0 1
OPT命中率0.7
```

```
FIFO内存中页号
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 3 1
2 3 0
4 3 0
4 2 0
4 2 3
0 2 3
0 2 3
0 2 3
0 1 3
0 1 2
0 1 2
0 1 2
7 1 2
7 0 2
7 0 1
FIFO命中率0.4
```

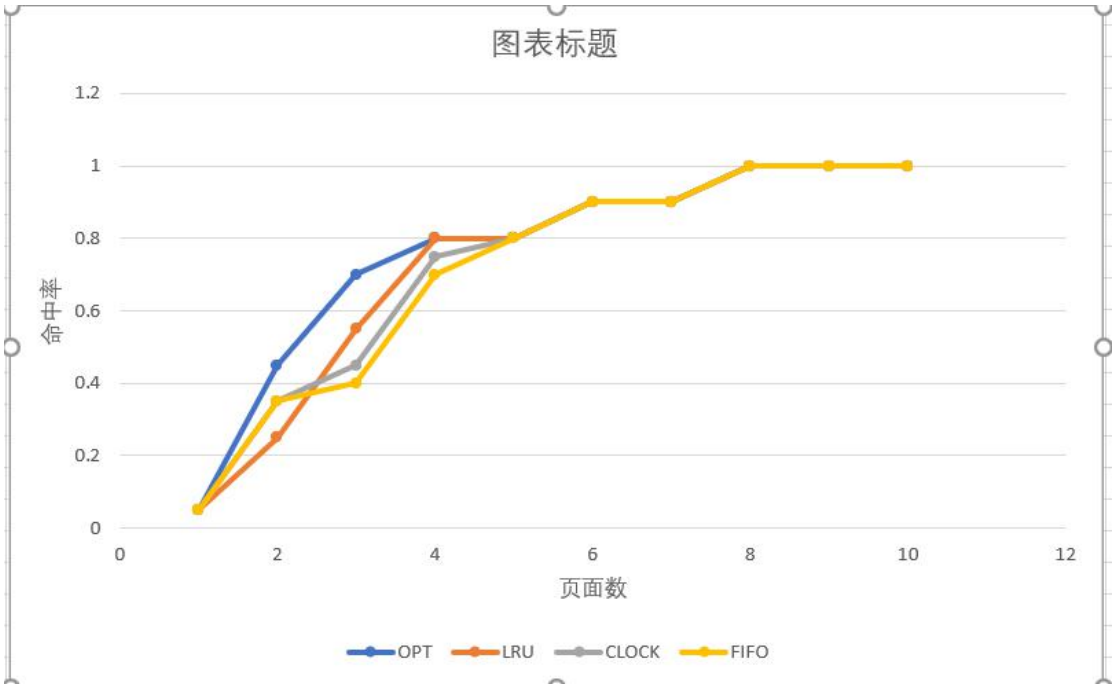
```
LRU内存中页号
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 0 3
2 0 3
4 0 3
4 0 2
4 3 2
0 3 2
0 3 2
0 3 2
1 3 2
1 3 2
1 0 2
1 0 2
1 0 7
1 0 7
1 0 7
LRU命中率0.55
```

```
CLOCK内存中页号
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 0 3
2 0 3
4 0 3
4 2 3
4 2 3
4 2 0
3 2 0
3 2 0
3 1 0
3 1 2
0 1 2
0 1 2
0 7 2
0 7 2
0 7 1
CLOCK命中率0.45
```

不同算法的命中率随着页面数的变化情况（测试序列）：

----------命中率----------

页面数	OPT	LRU	CLOCK	FIFO
1	0.05	0.05	0.05	0.05
2	0.45	0.25	0.35	0.35
3	0.7	0.55	0.45	0.4
4	0.8	0.8	0.75	0.7
5	0.8	0.8	0.8	0.8
6	0.9	0.9	0.9	0.9
7	0.9	0.9	0.9	0.9
8	1	1	1	1
9	1	1	1	1
10	1	1	1	1
11	1	1	1	1
12	1	1	1	1
13	1	1	1	1
14	1	1	1	1
15	1	1	1	1
16	1	1	1	1
17	1	1	1	1
18	1	1	1	1
19	1	1	1	1



五、实验源代码

```
#include <iostream>
#include <cstring>
using namespace std;

int test[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};
int test_size = sizeof(test) / sizeof(int);
// int page_num=3;
// int memory[page_num]={0};
```

```
double hit_rate(int miss_num)
{
    return 1 - (float)miss_num / test_size;
}
```

```
int is_miss(int memory[], int size_of_memory, int order)
{ //检查order是否在memory中，是则返回位置,否则返回-1
    for (int i = 0; i < size_of_memory; i++)
    {
        if (memory[i] == order)
        {
            return i;
        }
    }
    return -1;
}
```

```
void print_array(int memory[], int size_of_memory)
{
    for (int i = 0; i < size_of_memory; i++)
    {
        cout << memory[i] << " ";
    }
    cout << endl;
}
```

```
void init_memory(int memory[], int memory_size, int test[])
{
    for (int i = 0; i < memory_size; i++)
    { //memory 初始化
        memory[i] = -1;
    }
    for (int i = 0; i < memory_size; i++)
    {
```

```
        memory[i] = test[i]; //预装
        // print_array(memory, memory_size);
    }
}
```

```
int find_max_index(int array[], int array_size)
{
    int max_index = 0;
    for (int i = 0; i < array_size; i++)
    {
        if (array[i] > array[max_index])
        {
            max_index = i;
        }
    }
    return max_index;
}
```

```
class OPT
{
public:
    int miss_count = 0;
    OPT(int memory_size, int test[], int test_size)
    {
        // cout<<"OPT 内存中页号"<<endl;
        int memory[memory_size]; //主存
        init_memory(memory, memory_size, test);
        for (int i = memory_size; i < test_size; i++)
        { //遍历test
            if (is_miss(memory, memory_size, test[i]) == -1)
            { //若当前order 不在主存中,寻找换入。若在,do nothing
                miss_count++;
                //寻找最久不会使用的页面
                int flag[memory_size];
                // int index_long=-1;
                memset(flag, 0, sizeof(flag));
                int count = 0;
                for (int j = i; j < test_size; j++)
                {
                    int index_find = is_miss(memory, memory_size, test[j]);
                    if (index_find != -1)
                    {
                        flag[index_find] = 1; //找到的下标flag 都置为1
                        count++;
```

```
                    }
                    if (count == memory_size - 1)
                    {
                        break; //剩下的那一个就是最久未使用的
                    }
                }
                //换入
                for (int k = 0; k < memory_size; k++)
                {
                    if (flag[k] == 0)
                    {
                        memory[k] = test[i];
                        break;
                    }
                }
            }
            // print_array(memory, memory_size);
        }
        // cout << "OPT 命中率" << hit_rate(miss_count) << endl;
    }
};
```

```
class FIFO
{
public:
    int miss_count = 0;
    FIFO(int memory_size, int test[], int test_size)
    {
        // cout<<"FIFO 内存中页号"<<endl;
        int memory[memory_size]; //主存
        init_memory(memory, memory_size, test);
        int times[memory_size];
        for (int i = 0; i < memory_size; i++)
        {
            times[i] = memory_size - i;
        }
        for (int i = memory_size; i < test_size; i++) //遍历test
        {
            if (is_miss(memory, memory_size, test[i]) == -1) //若当前order不在主存中,寻找换入。若在,所有页面存在时间++
            {
                miss_count++;
                int max_index = find_max_index(times, memory_size);
                memory[max_index] = test[i];
```

```
                for (int j = 0; j < memory_size; j++)
                { //存在时间++
                    times[j]++;
                }
                times[max_index] = 1; //换入的存在时间为1
            }
            else
            {
                for (int j = 0; j < memory_size; j++)
                {
                    times[j]++;
                }
            }
            // print_array(memory, memory_size);
            // print_array(times,memory_size);
        }
        // cout << "FIFO 命中率" << hit_rate(miss_count) << endl;
    }
};
```

```
class LRU
{
public:
    int miss_count = 0;
    LRU(int memory_size, int test[], int test_size)
    {
        // cout<<"LRU 内存中页号"<<endl;
        int memory[memory_size]; //主存
        init_memory(memory, memory_size, test);
        int times[memory_size];
        for (int i = 0; i < memory_size; i++)
        {
            times[i] = memory_size - i;
        }
        for (int i = memory_size; i < test_size; i++) //遍历test
        {
            if (is_miss(memory, memory_size, test[i]) == -1) //若当前order不在主存中,寻找换入。若在,在的时间置1，其他++
            {
                miss_count++;
                int max_index = find_max_index(times, memory_size);
                memory[max_index] = test[i];
                for (int j = 0; j < memory_size; j++)
                { //存在时间++
```

```
                    times[j]++;
                }
                times[max_index] = 1; //换入的存在时间为1
            }
            else
            {
                for (int j = 0; j < memory_size; j++)
                {
                    times[j]++;
                }
                times[is_miss(memory, memory_size, test[i])] = 1;
            }
            // print_array(memory, memory_size);
            // print_array(times,memory_size);
        }
        // cout << "LRU 命中率" << hit_rate(miss_count) << endl;
    }
};
```

```
class CLOCK
{
public:
    int miss_count = 0;
    CLOCK(int memory_size, int test[], int test_size)
    {
        // cout<<"CLOCK 内存中页号"<<endl;
        int memory[memory_size]; //主存
        init_memory(memory, memory_size, test);
        int flag[memory_size];
        for (int i = 0; i < memory_size; i++)
        {
            flag[i] = 1;
        }
        int pointer = 0;                                //下标指针（伪）
        for (int i = memory_size; i < test_size; i++) //遍历test
        {
            if (is_miss(memory, memory_size, test[i]) == -1) //若当前order不在主存中,寻找换入。若在,仅置位
            {
                miss_count++;
                for (int j = 0; j <= memory_size; j++)
                {
                    if (flag[pointer] == 1)
                    {
```

```
                        flag[pointer] = 0;
                        pointer = (pointer + 1) % memory_size; //移动指针
                    }
                    else
                    {
                        memory[pointer] = test[i];
                        flag[pointer] = 1;
                        pointer = (pointer + 1) % memory_size;
                        break;
                    }
                }
            }
            else
            {
                flag[is_miss(memory, memory_size, test[i])] = 1;
            }
            // print_array(memory, memory_size);
            // print_array(flag,memory_size);
        }
        // cout << "CLOCK 命中率" << hit_rate(miss_count) << endl;
    }
};
```

```
int main()
{
    int page_num = 3;
    OPT(page_num, test, test_size);
    FIFO(page_num, test, test_size);
    LRU(page_num, test, test_size);
    CLOCK(page_num, test, test_size);
```

```
cout<<"------------命中率--------------"<<endl;
cout<<"页面数\t"<<"OPT\t"<<"LRU\t"<<"CLOCK\t"<<"FIFO\t"<<endl;
    for (int i = 1; i < test_size; i++)
    {
        OPT opt=OPT(i,test,test_size);
        LRU lru=LRU(i,test,test_size);
        CLOCK clock=CLOCK(i,test,test_size);
        FIFO fifo=FIFO(i,test,test_size);
        cout<<i<<"\t"
        <<hit_rate(opt.miss_count)<<"\t"
        <<hit_rate(lru.miss_count)<<"\t"
        <<hit_rate(clock.miss_count)<<"\t"
        <<hit_rate(fifo.miss_count)<<"\t"<<endl;
```

```
    }
    return 0;
}
```

实验总结

这次实验总体难度不是很大，需要实现的算法数目虽然不少，但基本思路较为相似，因此实现起来也并不是十分困难。通过完成这次实验，除了加深了我对几种策略的理解，锻炼了我的编程能力。通过阅读课件再加上自己的理解，我了解了老师的设计思路，感觉这个思路极其巧妙，设计中用到的方法和体现出的很多思想值得我们学习。

一开始做这个实验时，首先是看书，先把书上的替换算法知识点弄明白，要明白各种算法的优缺点和相互之间衍生互补关系。这四个算法中，难以实现的是LRU算法，因为它涉及到访问时间的计算，而且它的开销也比较大。OPT算法次难，它需要计算最近访问时间，并替换最近访问时间最大的页。而FIFO和CLOCK实现起来比较容易，FIFO算法的实现和CLOCK算法的实现很相似，FIFO可视为CLOCK的退化版。我先写了CLOCK算法，再删去一些约束条件就退化为FIFO算法。这就是两者的相同之处。

参考资料

计算机操作系统第四版（汤小丹）

https://www.cnblogs.com/fkissx/p/4712959.html

https://www.cnblogs.com/schips/p/10920145.html

https://blog.csdn.net/qq_41209741/article/details/99586257

https://blog.csdn.net/springtostring/article/details/85331177