华南理工大学

实 验 报 告

课程名称：	操 作 系 统
学生姓名：	陈卓文
学生学号：	**201936380215**
学生专业：	软件工程
开课学期：	**2020-2021 第二学期**

2021 年 05 月

实验四 存储管理算法模拟实验

地　　点： B7 楼 233 房； **评　　分：**

实验日期与时间： 2021.05.22 19：00-21：35 **实验教师：**

一、 实验目标

1. 请求页式管理是一种常用的虚拟存储管理技术，本实验的目的是通过模拟请求页式存储管理中的页面替换算法，了解虚拟存储技术的特点，感性认识操作系统对内存的管理，加深对内存管理的概念理解。

二、 实验内容

设计一个虚拟存储区和内存工作区，并使用下述页面替换算法计算页面访问的命中率。其中，命中率＝页面命中次数 / 页地址流长度，或者命中率＝ 1 －页面失效次数 / 页地址流长度。

1. 先进先出算法（FIFO）
2. 最近最少使用算法（LRU）
3. Optimal 算法

注：第三个算法在实际场景中无法使用。但是在仿真过程中是必要的，定义为：在已知全部页地址流情况下做出的最优调度策略，使得命中率最高并且替换少，作为算法1，2的baseline使用。

三、 实验设备及环境

PC （ubuntu 操作系统）；编程语言 C 语言。

四、 实验要求

本实验要求编程模拟一个拥有若干个虚页的进程在给定的若干实页中运行、并在缺页中断发生时分别使用 FIFO 和 LRU 算法进行页面置换的情形。同时实现 optimal 最优调度作为 baseline 进行结果对比。其中虚页的个数可以事先给定，对这些虚页访问的页地址流（其长度可以事先给定，例如 30 次虚页访问）可以由程序随机产生。

注：为了方便检查正确性，本次实验的页地址流由固定种子产生。

要求程序运行时屏幕能显示出置换过程中的**状态信息**并输出访问结束时的**页面命中率**。程序应允许为该进程分配不同的实页数，来比较两种置换算法的稳定性。

注：为测试结果具有普适性，请在完成编码后使用不少于 5 组给定的种子来验证算法的正确性并比较命中率。

五、 问题与算法

1. 问题描述

本次实验需要模拟页面替换算法中的三种算法，即FIFO、LRU和optimal算法。

2. 算法思路

(1) FIFO

当虚页置入内存时，记录下置入的时间。在释放内存时，将置入时间最前的页面释放。

(2) LRU

当虚页首次放入内存或者再次调用时，更新当前页面的上次调用时间。在释放内存时，将上次调用时间最前的页面释放。

(3) Optimal

当虚页首次放入内存或者再次调用时，更新当前页面的下次调用时间，如果没有下次调用，置为 INT_MAX。在释放内存时，将下次调用时间最后的页面释放。

3. 算法实现关键点

为了方便的储存各个页面的插入时间或调用时间，以及方便的查询当前虚页是否存在与内存中，以及统计内存占用页数，在此使用 C++ STL 库中的 std::map。并且每次通过遍历 map 中所有值以获取最前时间或者最后时间的页面。

六、 实验结果与分析

1. 实验数据及结果

本代码通过 argv 获取 Seed 的值，以下是种子为1， 333， 256， 512，1000的运行结果

```
simon@Aliyun:~/code/homework/operating_system/exp4$ ./a.out 1
The data input is:
3 1 2 0 3 0 1 2 4 1 2 2 0 4 3 1 0 1 2 1 1 3 2 4 2 0 2 3 2 0
when cache size is 2
The hit rate of FIFO is: 0.2
The output in FIFO is:
-1 -1 3 1 2 -1 0 3 1 2 4 -1 1 2 0 4 3 -1 1 0 -1 2 1 3 -1 2 4 0 -1 2
The hit rate of LRU is: 0.266667
The output in LRU is:
-1 -1 3 1 2 -1 3 0 1 2 4 -1 1 2 0 4 3 -1 0 -1 -1 2 1 3 -1 4 -1 0 -1 3
The hit rate of optimal is: 0.433333
The output in optimal is:
-1 -1 1 2 -1 -1 3 0 2 -1 1 -1 2 -1 4 3 -1 -1 0 -1 -1 1 -1 3 -1 4 -1 0 -1 2

when cache size is 3
The hit rate of FIFO is: 0.466667
The output in FIFO is:
-1 -1 -1 3 1 -1 2 0 3 -1 -1 -1 1 -1 2 4 -1 -1 0 -1 -1 -1 -1 3 -1 1 -1 2 4 -1
The hit rate of LRU is: 0.4
The output in LRU is:
-1 -1 -1 3 1 -1 2 3 0 -1 -1 -1 4 1 2 0 4 -1 3 -1 -1 0 -1 1 -1 3 -1 4 -1 -1
The hit rate of optimal is: 0.633333
The output in optimal is:
-1 -1 -1 2 -1 -1 -1 3 0 -1 -1 -1 2 -1 4 -1 -1 -1 0 -1 -1 -1 -1 1 -1 4 -1 -1 -1 -1

when cache size is 4
The hit rate of FIFO is: 0.7
The output in FIFO is:
-1 -1 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 -1 -1 1 2 -1 -1 0 -1 -1 -1 -1 -1 -1 4 -1 -1 -1 -1
The hit rate of LRU is: 0.666667
The output in LRU is:
-1 -1 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 -1 -1 1 2 -1 -1 4 -1 -1 -1 -1 0 -1 1 -1 -1 -1 -1
The hit rate of optimal is: 0.766667
The output in optimal is:
-1 -1 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1
```

图1　Seed = 1

```
simon@Aliyun:~/code/homework/operating_system/exp4$ ./a.out 333
The data input is:
1 4 0 3 1 4 4 3 2 0 4 3 4 1 0 3 2 3 0 4 3 1 0 0 1 1 2 1 4 3
when cache size is 2
The hit rate of FIFO is: 0.2
The output in FIFO is:
-1 -1 1 4 0 3 -1 1 4 3 2 0 -1 4 3 1 0 -1 3 2 0 4 3 -1 -1 -1 1 0 2 1
The hit rate of LRU is: 0.233333
The output in LRU is:
-1 -1 1 4 0 3 -1 1 4 3 2 0 -1 3 4 1 0 -1 2 3 0 4 3 -1 -1 -1 0 -1 2 1
The hit rate of optimal is: 0.4
The output in optimal is:
-1 -1 4 0 -1 1 -1 -1 3 2 -1 0 -1 4 1 -1 0 -1 2 0 -1 3 4 -1 -1 -1 0 -1 1 2

when cache size is 3
The hit rate of FIFO is: 0.333333
The output in FIFO is:
-1 -1 -1 1 4 0 -1 -1 3 1 -1 4 2 0 3 4 1 -1 -1 0 -1 3 2 -1 -1 -1 4 -1 1 0
The hit rate of LRU is: 0.333333
The output in LRU is:
-1 -1 -1 1 4 0 -1 -1 1 4 3 2 -1 0 3 4 1 -1 -1 2 -1 0 4 -1 -1 -1 3 -1 0 2
The hit rate of optimal is: 0.6
The output in optimal is:
-1 -1 -1 0 -1 -1 -1 -1 1 2 -1 -1 -1 4 -1 -1 1 -1 -1 2 -1 3 -1 -1 -1 -1 0 -1 -1 1

when cache size is 4
The hit rate of FIFO is: 0.7
The output in FIFO is:
-1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 0 -1 -1 3 -1 -1 -1 -1 -1 -1 2
The hit rate of LRU is: 0.566667
The output in LRU is:
-1 -1 -1 -1 -1 -1 -1 -1 0 1 -1 -1 -1 2 -1 -1 4 -1 -1 1 -1 2 -1 -1 -1 -1 4 -1 3 0
The hit rate of optimal is: 0.733333
The output in optimal is:
-1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 2 -1 -1 -1 -1 -1 -1 0 -1 -1 -1
```

图2　Seed = 333

```
simon@Aliyun:~/code/homework/operating_system/exp4$ ./a.out 256
The data input is:
3 2 0 0 0 0 3 1 4 0 0 0 2 3 1 1 1 0 1 3 1 2 1 1 1 2 0 3 2 2
when cache size is 2
The hit rate of FIFO is: 0.433333
The output in FIFO is:
-1 -1 3 -1 -1 -1 2 0 3 1 -1 -1 4 0 2 -1 -1 3 -1 1 0 3 -1 -1 -1 -1 1 2 0 -1
The hit rate of LRU is: 0.466667
The output in LRU is:
-1 -1 3 -1 -1 -1 2 0 3 1 -1 -1 4 0 2 -1 -1 3 -1 0 -1 3 -1 -1 -1 -1 1 2 0 -1
The hit rate of optimal is: 0.6
The output in optimal is:
-1 -1 2 -1 -1 -1 -1 3 1 -1 -1 -1 4 2 3 -1 -1 -1 -1 0 -1 3 -1 -1 -1 -1 1 0 -1 -1

when cache size is 3
The hit rate of FIFO is: 0.633333
The output in FIFO is:
-1 -1 -1 -1 -1 -1 -1 3 2 -1 -1 -1 0 1 4 -1 -1 2 -1 -1 -1 3 -1 -1 -1 -1 -1 1 -1 -1
The hit rate of LRU is: 0.566667
The output in LRU is:
-1 -1 -1 -1 -1 -1 -1 -1 2 0 3 -1 -1 1 4 0 -1 -1 2 -1 -1 -1 0 -1 -1 -1 -1 3 1 -1 -1
The hit rate of optimal is: 0.7
The output in optimal is:
-1 -1 -1 -1 -1 -1 -1 3 1 -1 -1 -1 -1 4 2 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 -1 -1 0 -1 -1

when cache size is 4
The hit rate of FIFO is: 0.733333
The output in FIFO is:
-1 -1 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 -1 2 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 -1 -1 1 -1 -1 -1
The hit rate of LRU is: 0.733333
The output in LRU is:
-1 -1 -1 -1 -1 -1 -1 -1 2 -1 -1 -1 3 1 4 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
The hit rate of optimal is: 0.8
The output in optimal is:
-1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

图3　Seed = 256

```
simon@Aliyun:~/code/homework/operating_system/exp4$ ./a.out 512
The data input is:
0 3 3 1 1 2 0 0 2 0 2 4 1 4 0 2 1 0 1 2 0 4 3 0 2 2 3 4 1 0
when cache size is 2
The hit rate of FIFO is: 0.333333
The output in FIFO is:
-1 -1 -1 0 -1 3 1 -1 -1 -1 -1 2 0 -1 4 1 0 2 -1 1 -1 0 2 4 3 -1 0 2 3 4
The hit rate of LRU is: 0.3
The output in LRU is:
-1 -1 -1 0 -1 3 1 -1 -1 -1 -1 0 2 -1 1 4 0 2 -1 0 1 2 0 4 3 -1 0 2 3 4
The hit rate of optimal is: 0.466667
The output in optimal is:
-1 -1 -1 3 -1 1 -1 -1 -1 -1 -1 2 0 -1 4 0 -1 2 -1 1 -1 2 4 -1 0 -1 -1 2 3 1

when cache size is 3
The hit rate of FIFO is: 0.533333
The output in FIFO is:
-1 -1 -1 -1 -1 0 3 -1 -1 -1 -1 1 2 -1 -1 0 -1 4 -1 -1 -1 1 2 -1 0 -1 -1 -1 4 3
The hit rate of LRU is: 0.466667
The output in LRU is:
-1 -1 -1 -1 -1 0 3 -1 -1 -1 -1 1 0 -1 2 1 4 -1 -1 -1 -1 1 2 -1 4 -1 -1 0 2 3
The hit rate of optimal is: 0.666667
The output in optimal is:
-1 -1 -1 -1 -1 3 -1 -1 -1 -1 -1 2 -1 -1 -1 4 -1 -1 -1 -1 -1 1 4 -1 -1 -1 -1 2 3 -1

when cache size is 4
The hit rate of FIFO is: 0.733333
The output in FIFO is:
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 3 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 2 -1
The hit rate of LRU is: 0.733333
The output in LRU is:
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 0 2
The hit rate of optimal is: 0.766667
The output in optimal is:
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 2 -1
```

图4　Seed = 512

```
simon@Aliyun:~/code/homework/operating_system/exp4$ ./a.out 1000
The data input is:
0 4 3 3 4 1 3 0 3 0 3 3 4 2 3 3 1 0 1 2 2 2 1 1 1 3 2 0 4 1
when cache size is 2
The hit rate of FIFO is: 0.4
The output in FIFO is:
-1 -1 0 -1 -1 4 -1 3 1 -1 -1 -1 0 3 4 -1 2 3 -1 1 -1 -1 0 -1 -1 2 1 3 2 0
The hit rate of LRU is: 0.433333
The output in LRU is:
-1 -1 0 -1 -1 3 4 1 -1 -1 -1 -1 0 3 4 -1 2 3 -1 0 -1 -1 -1 -1 -1 2 1 3 2 0
The hit rate of optimal is: 0.533333
The output in optimal is:
-1 -1 0 -1 -1 4 -1 1 -1 -1 -1 -1 0 4 -1 -1 3 2 -1 0 -1 -1 -1 -1 -1 1 -1 2 0 3

when cache size is 3
The hit rate of FIFO is: 0.533333
The output in FIFO is:
-1 -1 -1 -1 -1 0 -1 4 -1 -1 -1 -1 3 1 0 -1 4 2 -1 3 -1 -1 -1 -1 -1 1 -1 -1 0 2
The hit rate of LRU is: 0.533333
The output in LRU is:
-1 -1 -1 -1 -1 0 -1 4 -1 -1 -1 -1 1 0 -1 -1 4 2 -1 3 -1 -1 -1 -1 -1 0 -1 1 3 2
The hit rate of optimal is: 0.666667
The output in optimal is:
-1 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 -1 0 4 -1 -1 -1 3 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 0 2

when cache size is 4
The hit rate of FIFO is: 0.766667
The output in FIFO is:
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 -1 4 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 3 -1
The hit rate of LRU is: 0.7
The output in LRU is:
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 0 4 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 3
The hit rate of optimal is: 0.8
The output in optimal is:
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1
```

图5　Seed = 1000

2. 实验分析及结论

由 Hit Rate 可以看出，FIFO与LRU同作为最优解的Optimal相比，命中率虽有差距，但相差不大。FIFO与LRU两者，不相上下，可能是数据集太少，数据量太少，从而分辨不出优劣。

七、 心得与展望

1. 自我评价及心得体会

本次实验很成功的实现了三种算法，并且了解到了三种算法的区别与优劣。

2. 展望

无

八、 附录

1. 主要界面

Ubuntu 18.04 LTS, gcc version 7.5.0

2. 源程序

项目目录结构

```
└── exp4
    └── page-management.cpp
```

```
// page-management.cpp C++14 require
/**
 * @file page-management.cpp
 * @author zhaokangming (952917537@qq.com)
 * @author zhangyuanes (zhangyuanes@gmail.com)
 * @brief 存储管理框架代码
 * @version 0.2
 * @date 2021-05-17
 *
 * @copyright Copyright (c) 2021
 * @note
 1、 请勿修改本框架代码的函数定义
 2、 允许添加合适的C++头文件来进行编码
 3、 编程版本选择C或者C++其中一种即可，对应框架代码逻辑没有变化。

实验手册在线地址： https://www.zybuluo.com/yanbo01haomiao/note/1794341

 */
#include <time.h>
#include <iostream>
using namespace std;

#define DATA_NUM 30       // 页地址长度

/**
 * @brief 创建一个大小为N的页地址流数组
 * @param data 用于存储页地址流的数组
 * @param N 页地址流长度
```

```
 * @param seed 随机种子，相同种子产生的数组相同
 * @note 请勿修改本函数定义，如需改变不同页地址流请修改种子输入
 */
void create_data(int data[], int N, int seed) {
	srand(seed);
	for (int i = 0; i < N; i++) {
		data[i] = rand() % 5;
	}
}
/**
 * @brief 重置页地址流数据为-1
 * @param data 用于存储页地址流的数组
 * @param N 页地址流长度
 */
void reset_data(int data[], int N) {
	for (int i = 0; i < N; i++) {
		data[i] = -1;
	}
}
/**
 * @brief 输出页地址流数据
 * @param data 用于存储页地址流的数组
 * @param N 页地址流长度
 */
void output_data(int data[], int N) {
	for (int i = 0; i < N; i++) {
		cout << data[i] << " ";
	}
	cout << endl;
}
/**
 * @brief 创建一个大小为N的页地址流数据
 * @param output 用于存储页面置换的过程数组
 * @param size 过程数组长度
```

```
 */
void show_output(int output[], const int size) {
	for (int i = 0; i < size; ++i) {
		cout << output[i] << " ";
	}
	cout << endl;
}

/**
 * @brief FIFO具体算法实现
 * @param data  页地址流数组
 * @param N  页地址流长度
 * @param memory_Size cache长度
 * @param output  记录页面置换的过程数组
 * @return  页面命中率，范围[0,1]
 * @note  需要自行补充完成,完成对应逻辑和输出
 */
//   pre define
#include <algorithm>
#include <map>
float FIFO(int data[], const int N, const int memory_Size, int output[]) {
	int hit_cnt = 0;
	map<int, int> mp;		// Number insert_time
	for (int i = 0; i < N; i++) {
		if (mp.count(data[i])) hit_cnt++;
		if (mp.count(data[i]) || mp.size() < memory_Size) {
			output[i] = -1;
		}
		else {
			auto it = min_element(mp.begin(), mp.end(), [](auto a, auto b) {
				return a.second < b.second;
			});		// C++14 require
			output[i] = it->first;
			mp.erase(it);
```

```
            }
            mp.emplace(data[i], i);
        }
        return 1.0 * hit_cnt / N;
}

float LRU(int data[], const int N, const int memory_Size, int output[]) {
        int hit_cnt = 0;
        map<int, int> mp;       // Number last_call
        for (int i = 0; i < N; i++) {
            if (mp.count(data[i])) hit_cnt++;
            if (mp.count(data[i]) || mp.size() < memory_Size) {
                output[i] = -1;
            }
            else {
                auto it = min_element(mp.begin(), mp.end(), [](auto a, auto b) {
                    return a.second < b.second;
                });       // C++14 require
                output[i] = it->first;
                mp.erase(it);
            }
            mp[data[i]] = i;
        }
        return 1.0 * hit_cnt / N;
}

float optimal(int data[], const int N, const int memory_Size, int output[]) {
        int hit_cnt = 0;
        map<int, int> mp;       // Number next_used
        for (int i = 0; i < N; i++) {
            if (mp.count(data[i])) hit_cnt++;
            if (mp.count(data[i]) || mp.size() < memory_Size) {
                output[i] = -1;
            }
```

```
            else {
                auto it = max_element(mp.begin(), mp.end(), [](auto a, auto b) {
                    return a.second < b.second;
                });       // C++14 require
                output[i] = it->first;
                mp.erase(it);
            }
            mp[data[i]] = [&]() -> int {
                for (int j = i + 1; j < N; j++)
                    if (data[j] == data[i]) return j;
                return 0x7FFFFFFF;
            }();
        }
        return 1.0 * hit_cnt / N;
}

int main(int argc, char** argv) {
        int seed = 333;       // 随机种子，请依次测试随机种子 1， 333， 256， 512，
1000
        if (argc > 1) seed = stoi(argv[1]);
        int max_memory_size = 5;       // 最大cache长度
        int data_num = DATA_NUM;       // 页地址流长度
        int data[DATA_NUM];            // 页地址流数组
        int output
            [DATA_NUM];       // 页面置换的过程数组，-1表示首次置入cache或者
命中，对应地址数值表示从cache中置换出去的页面地址
        int i, j;
        float rate;
        // 利用随机种子产生随机页地址流，然后展示
        create_data(data, data_num, seed);
        cout << "The data input is: " << endl;
        for (i = 0; i < data_num; ++i) {
            cout << data[i] << " ";
        }
```

```
    cout << endl;

    for (j = 2; j < max_memory_size; ++j) {
        cout << "when cache size is " << j << endl;

        reset_data(output, data_num);
        rate = FIFO(data, data_num, j, output);
        cout << "The hit rate of FIFO is: " << rate << endl;
        cout << "The output in FIFO is: " << endl;
        show_output(output, data_num);

        reset_data(output, data_num);
        rate = LRU(data, data_num, j, output);
        cout << "The hit rate of LRU is: " << rate << endl;
        cout << "The output in LRU is: " << endl;
        show_output(output, data_num);

        reset_data(output, data_num);
        rate = optimal(data, data_num, j, output);
        cout << "The hit rate of optimal is: " << rate << endl;
        cout << "The output in optimal is: " << endl;
        show_output(output, data_num);

        cout << endl;
    }
    return 0;
}
```

九、 参考文献

[1] 《计算机操作系统教程》张尧学等，清华大学出版社，2006 年 10 月第 3 版