**华 南 农 业 大 学 数 学 与 信 息（软 件） 学 院**

《操作系统综合性与设计性实验》成绩单

开设时间：2020-2021学年第二学期

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **学号** | 201925310125 | **姓名** | | 于邦霆 | | **专业班级** | **2019级计算机科学与技术专业1班** |
| 实 验 题 目 | 题目二 磁盘调度算法的模拟实现及对比 | | | | | | | |
| 自 我 评 价 | 本次做的是实现磁盘调度算法模拟的实验，完成实验后，我对磁盘如何调度有了更深刻的理解，其中FCFS算法、SSTF算法、SCAN算法以及C-SCAN算法，其中的算法内涵和应当注意的关键细节有了一定的思考和掌握。操作系统中实现磁盘调度也有着关键的作用，是将磁盘信息与电脑主存中信息交换的具体实现方式，采用不同的磁盘调度算法，会影响磁盘的存取效率，所以一个好的磁盘调度算法对磁盘的存取效率也有着至关重要的作用。 | | | | | | | |
| 教 师 评 语 | 评价指标：   * 题目内容和要求完成情况 优 □ 良 □ 中 □ 差 □ * 对算法原理的理解程度 优 □ 良 □ 中 □ 差 □ * 程序设计水平 优 □ 良 □ 中 □ 差 □ * 程序运行效果及正确性 优 □ 良 □ 中 □ 差 □ * 课程设计报告结构清晰 优 □ 良 □ 中 □ 差 □ * 报告中总结和分析详尽 优 □ 良 □ 中 □ 差 □ | | | | | | | |
| **成绩** |  | | | **教师签名** | | **张丽霞** | | |

目录

[**一、** **需求分析** 3](#_Toc90298427)

[**1.** **输入形式及输入值范围** 3](#_Toc90298428)

[5) 重复步骤 1~4，直到产生 400 个指令地址。1.2 算法的输入 3](#_Toc90298429)

[**2.** **输出形式** 3](#_Toc90298430)

[**3.** **程序功能** 3](#_Toc90298431)

[**二、** **概要设计** 4](#_Toc90298432)

[**1.** **抽象数据类型的定义** 4](#_Toc90298433)

[**2.** **主程序的流程** 4](#_Toc90298434)

[**3.** **各程序模块之间的关系** 4](#_Toc90298435)

[**三、** **详细设计** 4](#_Toc90298436)

[1. Random函数 4](#_Toc90298437)

[2. FIFO函数 8](#_Toc90298438)

[3. LRU函数 10](#_Toc90298439)

[4. OPT函数 12](#_Toc90298440)

[5. replace函数 15](#_Toc90298441)

[**四、** **调试分析** 17](#_Toc90298442)

[1. 设计与实现分析 17](#_Toc90298443)

[2. 时间复杂度 18](#_Toc90298444)

[2.1 FIFO算法 18](#_Toc90298445)

[2.2 LRU算法 18](#_Toc90298446)

[2.3 OPT算法 18](#_Toc90298447)

[3. 主要问题及解决方法 19](#_Toc90298448)

[3.1 数据的随机产生 19](#_Toc90298449)

[3.2 程序所能达到的功能 19](#_Toc90298450)

[**五、** **用户使用说明** 19](#_Toc90298451)

[1. 使用说明 19](#_Toc90298452)

[**六、** **测试与运行结果** 19](#_Toc90298453)

[1. 运行截图 20](#_Toc90298454)

[2. 命令行输出 20](#_Toc90298455)

[**七、** **经验和体会** 22](#_Toc90298456)

1. **需求分析**
2. **输入形式及输入值范围**

1、首先用随机数生成函数产生一个“指令将要访问的地址序列”，然后将地址序列变换成相应

的页地址流（即页访问序列），再计算不同算法下的命中率。

2、通过随机数产生一个地址序列，共产生 400 条。其中 50%的地址访问是顺序执行的，另外

50%就是非顺序执行。且地址在前半部地址空间和后半部地址空间均匀分布。具体产生方法如下：

1) 在前半部地址空间，即[0，199]中随机选一数 m，记录到地址流数组中（这是非顺序

执行）；

2) 接着“顺序执行一条指令”，即执行地址为 m+1 的指令，把 m+1 记录下来；

3) 在后半部地址空间，[200，399]中随机选一数 m’，作为新指令地址；

4) 顺序执行一条指令，其地址为 m’+1；

### 5) 重复步骤 1~4，直到产生 400 个指令地址。1.2 算法的输入

1. **输出形式**

循环运行，使用户内存容量从 4 页框到 40 页框。计算每个内存容量下不同页面置换算法的

命中率。输出结果可以为：

页框数 OPT 缺页率 FIFO 缺页率 LRU 缺页率

[4] OPT：0.5566 FIFO：0.4455 LRU：0.5500

[5] OPT：0.6644 FIFO：0.5544 LRU：0.5588

…… ……

…… ……

[39] OPT：0.9000 FIFO：0.9000 LRU：0.9000

[40] OPT：1.0000 FIFO：1.0000 LRU：1.0000

注 1：在某一次实验中，可能 FIFO 比 LRU 性能更好，但足够多次的实验表明 LRU 的平均

性能比 FIFO 更好。

注 2：计算缺页率时，以页框填满之前和之后的总缺页次数计算。

1. **程序功能**

OpenEuler 操作系统在地址映射过程中，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。通过请求页式管理方式中页面置换算法的模拟设计，了解虚拟存储技术的特点，掌握请求页式存储管理中的页面置换算法。

在 OpenEuler 操作系统中，使用高级编程语言模拟实现最佳置换法（OPT）、先进先出法(FIFO) 和最近最少使用法(LRU)，并计算每种算法的缺页率。

1. **概要设计**
2. **抽象数据类型的定义**
3. **主程序的流程**

首先通过程序生成随机数，并将随机数转化为Position类（地址类），接着将所有Position存入Position数组，通过此来

## 时间复杂度

### FIFO算法

FIFO 算法的基本操作以及其复杂性分析：

(1)建立两个数组，一个表示页地址流数组，一个表示页框数组；

(2)循环页地址流数组里的元素，将页框数组当成队列来使用，达到先进先出的效果；

(3)相关的数组类的方法：push()、shift()、indexOf()

(4)因此 FIFO 算法的时间复杂性为：O(n)

### LRU算法

(1)建立两个数组，并且将第二个数组当队列使用，即将每次进来的从后面进入，删除 的从前面删除，并且关键是将不缺页时重新访问的那个元素调出来重新插到最后一个

(2)相关的数组类的方法：splice():指定数组下标删除某元素

(3)因此 LRU 算法的时间复杂性为：O(n2)

### OPT算法

建立两个数组，并且将第二个数组当队列使用，即将每次进来的从后面进入，删除的从 前面删除；并且关键是当缺页且页框队列里没有相同的页号时，从页号流当前位置出发 往后循环页号流数组，找到接下来最不可能使用到的页号，将其从页框数组中置换出来， 并将新的页号插入到页框队列的尾部

（1）相关的数组类的方法：相关的数组类的方法：push()、shift()、indexOf()

（2）splice():指定数组下标删除某元素

（3）因此 OPT 算法的时间复杂性为：O(n2)

## 主要问题及解决方法

### 3.1 数据的随机产生

① 程序的输入值是执行指令的地址流以及其对应的页号流，该指令的地址流包含 400 个随机产生的地址数据，页号流也包含 400 个页号，页号流有地址流转换得到。其中 地 址流中 50% 的地址访问是顺序执行的，另外 50% 就是非顺序执行，且地址在前半部地 址空间和后半部地址空间均匀分布.

②400 个地址流输入值的范围在[0,399]之间，其中 50% 在[0,199],50% 在[200,399].

③400 个页号流输入值的范围在[0,40]之间，页号流由地址流转换而来.

### 程序所能达到的功能

① 程序能按要求随机产生符合要求的地址流，并将其转换为对应页号流.

② 程序利用产生的页号流，模拟页面置换的三个算法——FIFO、LUR、OPT 算法，计算 三个置换算法在页框号在 4-40 的范围内的命中率，并将其以表格的形式输出显示到图 形[用户界面](https://www.zhihu.com/search?q=%E7%94%A8%E6%88%B7%E7%95%8C%E9%9D%A2&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A437637134%7D)上.

1. **用户使用说明**

## 1. 使用说明

用户无需做任何操作。打开源代码后找到Run软件包中的Main类后点击运行，即可在控制台找到输出。

1. **测试与运行结果**

## 运行截图





## 命令行输出

页框数 FIFO缺页率 LRU 缺页率 OPT 缺页率

[ 4] FIFO缺页率：0.4750 LRU 缺页率：0.4900 OPT缺页率：0.2775

[ 5] FIFO缺页率：0.4675 LRU 缺页率：0.4775 OPT缺页率：0.2925

[ 6] FIFO缺页率：0.4550 LRU 缺页率：0.4525 OPT缺页率：0.2950

[ 7] FIFO缺页率：0.4450 LRU 缺页率：0.4400 OPT缺页率：0.3050

[ 8] FIFO缺页率：0.4325 LRU 缺页率：0.4375 OPT缺页率：0.3050

[ 9] FIFO缺页率：0.4050 LRU 缺页率：0.4125 OPT缺页率：0.3125

[10] FIFO缺页率：0.4000 LRU 缺页率：0.3975 OPT缺页率：0.3075

[11] FIFO缺页率：0.3900 LRU 缺页率：0.3875 OPT缺页率：0.3150

[12] FIFO缺页率：0.3825 LRU 缺页率：0.3800 OPT缺页率：0.3100

[13] FIFO缺页率：0.3775 LRU 缺页率：0.3700 OPT缺页率：0.3225

[14] FIFO缺页率：0.3725 LRU 缺页率：0.3625 OPT缺页率：0.3075

[15] FIFO缺页率：0.3675 LRU 缺页率：0.3550 OPT缺页率：0.3200

[16] FIFO缺页率：0.3650 LRU 缺页率：0.3500 OPT缺页率：0.3125

[17] FIFO缺页率：0.3575 LRU 缺页率：0.3400 OPT缺页率：0.3050

[18] FIFO缺页率：0.3550 LRU 缺页率：0.3350 OPT缺页率：0.3000

[19] FIFO缺页率：0.3175 LRU 缺页率：0.3225 OPT缺页率：0.2950

[20] FIFO缺页率：0.2950 LRU 缺页率：0.3150 OPT缺页率：0.2825

[21] FIFO缺页率：0.2800 LRU 缺页率：0.3075 OPT缺页率：0.2750

[22] FIFO缺页率：0.2900 LRU 缺页率：0.2875 OPT缺页率：0.2700

[23] FIFO缺页率：0.2525 LRU 缺页率：0.2850 OPT缺页率：0.2600

[24] FIFO缺页率：0.2450 LRU 缺页率：0.2650 OPT缺页率：0.2450

[25] FIFO缺页率：0.2450 LRU 缺页率：0.2525 OPT缺页率：0.2400

[26] FIFO缺页率：0.2450 LRU 缺页率：0.2450 OPT缺页率：0.2275

[27] FIFO缺页率：0.2100 LRU 缺页率：0.2300 OPT缺页率：0.2125

[28] FIFO缺页率：0.2000 LRU 缺页率：0.2200 OPT缺页率：0.1975

[29] FIFO缺页率：0.1850 LRU 缺页率：0.2050 OPT缺页率：0.1900

[30] FIFO缺页率：0.1675 LRU 缺页率：0.1925 OPT缺页率：0.1800

[31] FIFO缺页率：0.1675 LRU 缺页率：0.1825 OPT缺页率：0.1650

[32] FIFO缺页率：0.1575 LRU 缺页率：0.1675 OPT缺页率：0.1575

[33] FIFO缺页率：0.1500 LRU 缺页率：0.1550 OPT缺页率：0.1500

[34] FIFO缺页率：0.1375 LRU 缺页率：0.1525 OPT缺页率：0.1400

[35] FIFO缺页率：0.1350 LRU 缺页率：0.1500 OPT缺页率：0.1275

[36] FIFO缺页率：0.1300 LRU 缺页率：0.1375 OPT缺页率：0.1275

[37] FIFO缺页率：0.1200 LRU 缺页率：0.1350 OPT缺页率：0.1175

[38] FIFO缺页率：0.1100 LRU 缺页率：0.1150 OPT缺页率：0.1100

[39] FIFO缺页率：0.1050 LRU 缺页率：0.1100 OPT缺页率：0.1000

[40] FIFO缺页率：0.1000 LRU 缺页率：0.1000 OPT缺页率：0.1000

页框数 FIFO命中率 LRU 命中率 OPT 命中率

[ 4] FIFO命中率：0.5350 LRU 命中率：0.5200 OPT命中率：0.7325

[ 5] FIFO命中率：0.5450 LRU 命中率：0.5350 OPT命中率：0.7200

[ 6] FIFO命中率：0.5600 LRU 命中率：0.5625 OPT命中率：0.7200

[ 7] FIFO命中率：0.5725 LRU 命中率：0.5775 OPT命中率：0.7125

[ 8] FIFO命中率：0.5875 LRU 命中率：0.5825 OPT命中率：0.7150

[ 9] FIFO命中率：0.6175 LRU 命中率：0.6100 OPT命中率：0.7100

[10] FIFO命中率：0.6250 LRU 命中率：0.6275 OPT命中率：0.7175

[11] FIFO命中率：0.6375 LRU 命中率：0.6400 OPT命中率：0.7125

[12] FIFO命中率：0.6475 LRU 命中率：0.6500 OPT命中率：0.7200

[13] FIFO命中率：0.6550 LRU 命中率：0.6625 OPT命中率：0.7100

[14] FIFO命中率：0.6625 LRU 命中率：0.6725 OPT命中率：0.7275

[15] FIFO命中率：0.6700 LRU 命中率：0.6825 OPT命中率：0.7175

[16] FIFO命中率：0.6750 LRU 命中率：0.6900 OPT命中率：0.7275

[17] FIFO命中率：0.6850 LRU 命中率：0.7025 OPT命中率：0.7375

[18] FIFO命中率：0.6900 LRU 命中率：0.7100 OPT命中率：0.7450

[19] FIFO命中率：0.7300 LRU 命中率：0.7250 OPT命中率：0.7525

[20] FIFO命中率：0.7550 LRU 命中率：0.7350 OPT命中率：0.7675

[21] FIFO命中率：0.7725 LRU 命中率：0.7450 OPT命中率：0.7775

[22] FIFO命中率：0.7650 LRU 命中率：0.7675 OPT命中率：0.7850

[23] FIFO命中率：0.8050 LRU 命中率：0.7725 OPT命中率：0.7975

[24] FIFO命中率：0.8150 LRU 命中率：0.7950 OPT命中率：0.8150

[25] FIFO命中率：0.8175 LRU 命中率：0.8100 OPT命中率：0.8225

[26] FIFO命中率：0.8200 LRU 命中率：0.8200 OPT命中率：0.8375

[27] FIFO命中率：0.8575 LRU 命中率：0.8375 OPT命中率：0.8550

[28] FIFO命中率：0.8700 LRU 命中率：0.8500 OPT命中率：0.8725

[29] FIFO命中率：0.8875 LRU 命中率：0.8675 OPT命中率：0.8825

[30] FIFO命中率：0.9075 LRU 命中率：0.8825 OPT命中率：0.8950

[31] FIFO命中率：0.9100 LRU 命中率：0.8950 OPT命中率：0.9125

[32] FIFO命中率：0.9225 LRU 命中率：0.9125 OPT命中率：0.9225

[33] FIFO命中率：0.9325 LRU 命中率：0.9275 OPT命中率：0.9325

[34] FIFO命中率：0.9475 LRU 命中率：0.9325 OPT命中率：0.9450

[35] FIFO命中率：0.9525 LRU 命中率：0.9375 OPT命中率：0.9600

[36] FIFO命中率：0.9600 LRU 命中率：0.9525 OPT命中率：0.9625

[37] FIFO命中率：0.9725 LRU 命中率：0.9575 OPT命中率：0.9750

[38] FIFO命中率：0.9850 LRU 命中率：0.9800 OPT命中率：0.9850

[39] FIFO命中率：0.9925 LRU 命中率：0.9875 OPT命中率：0.9975

[40] FIFO命中率：1.0000 LRU 命中率：1.0000 OPT命中率：1.0000

1. **经验和体会**

① 本程序的编写设计思路是先实现底层的逻辑算法，即第一步是先实现随机产生地址 流，并将其转换为页号流，并对数据的正确性进行验证与调试；第二步是先单独地实现 3 个页面置换算法——FIFO、LRU、OPT 算法的逻辑代码部分并逐个对其进行测试与维 护；第三步便模拟三个置换算法的置换过程，并计算它们对应的命中率；第四步便是将 模拟的页面置换过程以及输入输出等数据显示到图形用户界面上，并验证调试正确性； 第五步是实现页面的交互动作以及页面的样式风格；最后再进行整体的多次整合和调试

② 在第一步时遇到很多困难，比如前期对三个页面置换算法的不熟悉，核心思路掌握 得不牢靠，导致实现起来的逻辑思路比较乱，后来重新认真地看了各个算法的定义以及 对应着各个算法走一遍流程.

③ 后来在实现置换算法时一个致命的错误便是一开始一直没有考虑到页框还未初始化 的情况，走了几遍算法以及反复调试后最后才意识到这点，页框数组在初始化前后对页 号的操作方法完全不同，不应该只是单单地考虑核心算法部分而忽略了基础的部分，细 节是十分重要的，有时甚至直接影响了整个程序.

④ 三个页面置换算法中最难实现、错误率最多的无疑是 OPT，它作为最佳置换算法，便有它的优势与难点。我用的方法是当遇到缺页且无重复页号时从当前的 地址流数组开始，往后循环直到找到那个在接下来的置换过程中最晚用到的甚至可能都 用不到的。思路一开始虽然可行，但在调试的过程中我发现有一种情况下回一直报错， 就是当页框数组中有两个页号的情况条件是完全一致的情况下，即算法也无法确定是两个哪个先被[置换](https://www.zhihu.com/search?q=%E7%BD%AE%E6%8D%A2&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A437637134%7D)出来，后来调试了多次，拿了多组数据进行反复测试后才找到这个错误， 最后从二者中选一来进行置换.

⑤ 当在进行实现图形用户界面时，当我把单击响应函数绑定到按钮上，以该按钮来进 行交互，但当我反复测试点击按钮都没有任何反应，便想可能是逻辑层出现了问题，所 以一直调试逻辑层，最后在反复否测试中发现，当我点击按钮时，页面的 DOM 节点还 没加载完成，即相当于我还没定义变量前便开始使用该变量，结果当然会报错。所以最 后发现是这个问题后，我便将按钮的单击响应函数放到了 jQuery 的页面加载函数： $(document).ready( ) 中执行，便解决了这个问题.

⑥ 在调试的过程中还遇到了许多大大小小的问题，有对 JavaScript 语言语法本身的不熟 悉，也有对算法逻辑的错误理解，最后在自己的努力与思考之下成功地实现了该程序， 收获了很多，更加深入地理解了软件开发的流程以及实现过程.

改进设想；

①改进设想：

结合其他的页面置换算法，除了计算其命中率一以外，我觉得还可以研究几个的页面置 换规律，找到它们之间的异同，这样也更有利于理解内存与进程之间的调度关系.

设计过程的经验和体会；

① 本程序的编写设计思路是先实现底层的逻辑算法，即第一步是先实现随机产生地址 流，并将其转换为页号流，并对数据的正确性进行验证与调试；第二步是先单独地实现 3 个页面置换算法——FIFO、LRU、OPT 算法的逻辑代码部分并逐个对其进行测试与维 护；第三步便模拟三个置换算法的置换过程，并计算它们对应的命中率；第四步便是将 模拟的页面置换过程以及输入输出等数据显示到图形用户界面上，并验证调试正确性； 第五步是实现页面的交互动作以及页面的样式风格；最后再进行整体的多次整合和调试

② 在调试的过程中还遇到了许多大大小小的问题，有对 JavaScript 语言语法本身的不熟 悉，也有对算法逻辑的错误理解，最后在自己的努力与思考之下成功地实现了该程序， 收获了很多，更加深入地理解了软件开发的流程以及实现过程.

③ 所以编程还是应该多多上机实现以下，毕竟机器设备相对我们而言在代码的语法容 错性上还是更加严谨的。不过在这基础上更应该打好基础，有更加丰富的知识才能编写 出更加完善的程序.

实现过程中出现的主要问题及解决方法。

主要问题有几个方面，一个是自身本身对语言的语法知识掌握不牢靠二导致，二是对算 法的思维理解不透彻，三是在实现程序时遇到的逻辑层面的错误：

① 本程序的编写设计思路是先实现底层的逻辑算法，即第一步是先实现随机产生地址 流，并将其转换为页号流，并对数据的正确性进行验证与调试；第二步是先单独地实现 3 个页面置换算法——FIFO、LRU、OPT 算法的逻辑代码部分并逐个对其进行测试与维 护；第三步便模拟三个置换算法的置换过程，并计算它们对应的命中率；第四步便是将 模拟的页面置换过程以及输入输出等数据显示到图形用户界面上，并验证调试正确性； 第五步是实现页面的交互动作以及页面的样式风格；最后再进行整体的多次整合和调试

② 在第一步时遇到很多困难，比如前期对三个页面置换算法的不熟悉，核心思路掌握 得不牢靠，导致实现起来的逻辑思路比较乱，后来重新认真地看了各个算法的定义以及 对应着各个算法走一遍流程.

③ 后来在实现置换算法时一个致命的错误便是一开始一直没有考虑到页框还未初始化 的情况，走了几遍算法以及反复调试后最后才意识到这点，页框数组在初始化前后对页 号的操作方法完全不同，不应该只是单单地考虑核心算法部分而忽略了基础的部分，细 节是十分重要的，有时甚至直接影响了整个程序.

④ 三个页面置换算法中最难实现、错误率最多的无疑是 OPT 最佳置换算法，它作为最 佳置换算法，便有它的优势与难点。我用的方法是当遇到缺页且无重复页号时从当前的 地址流数组开始，往后循环直到找到那个在接下来的置换过程中最晚用到的甚至可能都 用不到的。思路一开始虽然可行，但在调试的过程中我发现有一种情况下回一直报错， 就是当页框数组中有两个页号的情况条件是完全一致的情况下，即算法也无法确定是两 个哪个先被置换出来，后来调试了多次，拿了多组数据进行反复测试后才找到这个错误， 最后从二者中选一来进行置换.

⑤ 当在进行实现图形用户界面时，当我把单击响应函数绑定到按钮上，以该按钮来进 行交互，但当我反复测试点击按钮都没有任何反应，便想可能是逻辑层出现了问题，所 以一直调试逻辑层，最后在反复否测试中发现，当我点击按钮时，页面的 DOM 节点还 没加载完成，即相当于我还没定义变量前便开始使用该变量，结果当然会报错。所以最 后发现是这个问题后，我便将按钮的单击响应函数放到了 jQuery 的页面加载函数： $(document).ready( ) 中执行，便解决了这个问题.

⑥ 在调试的过程中还遇到了许多大大小小的问题，有对 JavaScript 语言语法本身的不熟 悉，也有对算法逻辑的错误理解，最后在自己的努力与思考之下成功地实现了该程序， 收获了很多，更加深入地理解了软件开发的流程以及实现过程.