

**操作系统实验报告**

|  |  |
| --- | --- |
| **学 院：** | **计算机科学学院** |
| **专 业：** | **计算机科学与技术（创新实验班）** |
| **学 号：** | **42112255** |
| **姓 名：** | **何佳民** |
| **指导教师：** | **孙增国** |

**2023 年 12月 15日**

**实验五 虚拟内存页面置换算法**

**一、实验目的**

通过这次实验，加深对虚拟内存页面置换概念的理解，进一步掌握先进先出FIFO、最佳置换OPI和最近最久未使用LRU页面置换算法的实现方法。

**二、实验器材**

微机

**三、实验要求**

1）上机前认真复习页面置换算法，熟悉FIFO，OPI，LRU三种页面分配和置换算法的过程；

2）上机时独立编程、调试程序；

3）根据具体实验要求，完成好实验报告（包括实验的目的、内容、要求、源程序、实例运行结果截图、发现的问题以及解决方法）。

**四、实验内容**

**1.问题描述：**

设计程序模拟先进先出FIFO、最佳置换OPI和最近最久未使用LRU页面置换算法的工作过程。假设内存中分配给每个进程的最小物理块数为m，在进程运行过程中要访问的页面个数为n，页面访问序列为P1, … ,Pn，分别利用不同的页面置换算法调度进程的页面访问序列，给出页面访问序列的置换过程，计算每种算法缺页次数和缺页率。

**2.程序要求：**

1）利用先进先出FIFO、最佳置换OPI和最近最久未使用LRU三种页面置换算法模拟页面访问过程。

2）模拟三种算法的页面置换过程，给出每个页面访问时的内存分配情况。

3）输入：最小物理块数m，页面个数n，页面访问序列P1, … ,Pn，算法选择1-FIFO，2-OPI，3-LRU。

4）输出：每种算法的缺页次数和缺页率。

**3.问题分析：**

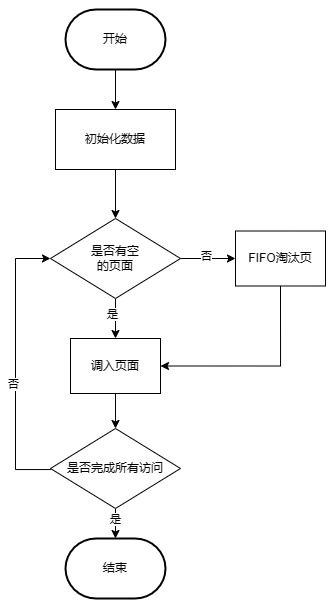


图 5-1 FIFO页面置换算法流程图

该算法实现简单，只需把一个进程已调入内存的页面，按先后次序链接成一个队列，并设置一个指针，称为替换指针，使它总是指向最老的页面。但该算法与进程实际运行的规律不相适应，因为在进程中，有些页面经常被访问，比如，含有全局变量、常用函数、例程等的页面，FIFO算法并不能保证这些页面不被淘汰。

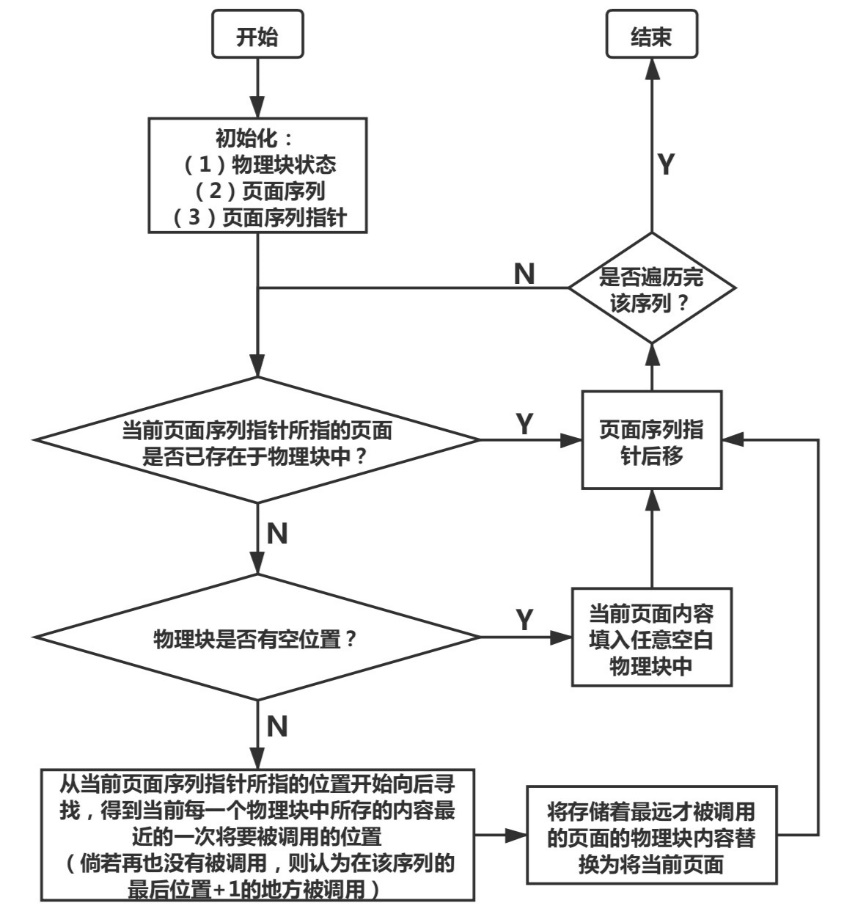


图 5-2 最佳（Optimal）页面置换算法流程图

这个算法所选择的被淘汰页面，将是以后永不使用的，或是在最长(未来)时间内不再被访问的页面。采用最佳置换算法，通常可保证获得最低的缺页率。但由于人们目前还无法预知一个进程在内存的若干个页面中，哪一个页面是未来最长时间内不再被访问的，因而该算法是无法实现的，但可以利用该算法去评价其它算法。

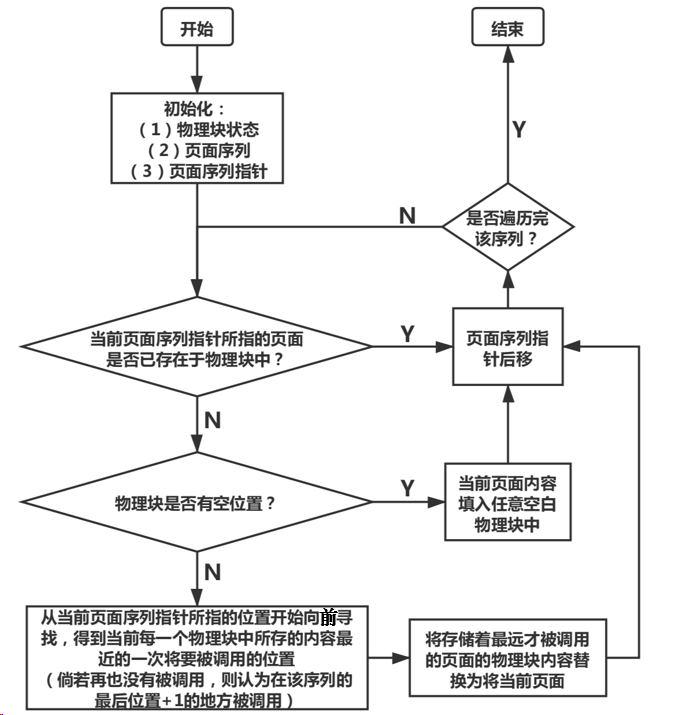


图 5-3 最近最久未使用(LRU)置换算法流程图

FIFO置换算法性能之所以较差，是因为它所依据的条件是各个页面调入内存的时间，而页面调入的先后并不能反映页面的使用情况。最近最久未使用(LRU)的页面置换算法，是根据页面调入内存后的使用情况进行决策的。由于无法预测各页面将来的使用情况，只能利用“最近的过去”作为“最近的将来”的近似，因此，LRU置换算法是选择最近最久未使用的页面予以淘汰。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来所经历的时间t，当须淘汰一个页面时，选择现有页面中其t值最大的，即最近最久未使用的页面予以淘汰。

**4.源程序：**

data.txt内容：

3

20

4 3 2 1 4 3 5 4 3 2 1 5 6 2 3 7 1 2 6 1

Python代码（Python版本3.9）：

# 全局变量定义

MaxNumber = 100

PageOrder = [0] \* MaxNumber # 页面序列

PageNum = LackNum = MinBlockNum = 0 # 页面个数，缺页次数, 最小物理块数

PageDisCount = [0] \* MaxNumber # 当前内存距离下一次出现的距离

LRUtime = [0] \* MaxNumber # 存储队列中各个页面最近使用情况

LackPageRate = 0.0 # 缺页率

LackPageNum = 0 # 缺页数

VirtualQueue = [-1] \* MaxNumber # 虚拟队列

def input\_data():

"""读取数据并初始化全局变量"""

global MinBlockNum, PageNum, PageOrder

with open("data.txt", "r") as readData:

MinBlockNum = int(readData.readline())

PageNum = int(readData.readline())

PageOrder = [int(x) for x in readData.readline().split()] # 修改此行

print("读取数据结果如下：")

print(f"最小物理块数 = {MinBlockNum}")

print(f"页面个数 = {PageNum}")

print("页面序列如下:")

for i in range(PageNum):

print(PageOrder[i], end=" ")

print()

def initial():

"""初始化页面序列和队列"""

global LackPageNum, LackPageRate, VirtualQueue, LRUtime

LackPageNum = MinBlockNum

LackPageRate = 0.0

for i in range(PageNum):

PageDisCount[i] = 0 # 初始化距离都为0

VirtualQueue[i] = -1 # 初始化队列的值都为负数

for i in range(MinBlockNum):

isInQueue2 = False

for j in range(MinBlockNum):

if VirtualQueue[j] == PageOrder[i]:

isInQueue2 = True

if not isInQueue2: # 当有新的进程进入到队列时，便计算其对应的距离

VirtualQueue[i] = PageOrder[i]

for k in range(i):

LRUtime[k] += 1 # 之前的页面对应的时间+1

display()

else:

LRUtime[i] = 0 # 重新更新为0，表示最近刚刚使用

def FIFO():

"""先进先出页面置换算法"""

print("\*\*\*\*\*\*\*\*\* 你选择了FIFO算法：\*\*\*\*\*\*\*\*\* ")

print("页面置换情况如下:")

initial()

isInQueue = False

point = 0 # 指向最老的页面

for i in range(MinBlockNum, PageNum):

isInQueue = False

for k in range(MinBlockNum):

if VirtualQueue[k] == PageOrder[i]: # 如果当前页面在队列中

isInQueue = True

if not isInQueue: # 如果当前页面不在队列中，则进行相应的处理

global LackPageNum

LackPageNum += 1 # 缺页数加1

VirtualQueue[point] = PageOrder[i]

display()

point += 1

if point == MinBlockNum:

point = 0 # 当point指向队尾后一位的时候，将point重新指向队首

global LackPageRate

LackPageRate = LackPageNum / PageNum

print(f"缺页数LackPageNum = {LackPageNum}")

print(f"缺页率LackPageRate = {LackPageRate}")

def OPI():

"""最佳置换算法"""

print("\*\*\*\*\*\*\*\*\* 你选择了OPI算法：\*\*\*\*\*\*\*\*\* ")

print("页面置换情况如下:")

initial()

isInQueue = False

dis = 0 # 表示队列每个值距离下一次访问的距离

point = 0 # 指向最长时间未被访问的下标

for i in range(MinBlockNum, PageNum):

isInQueue = False

for k in range(MinBlockNum):

if VirtualQueue[k] == PageOrder[i]: # 如果当前页面在队列中

isInQueue = True

if not isInQueue:

global LackPageNum

LackPageNum += 1

# 计算当前队列每一页对应的下一次出现的距离

for s in range(MinBlockNum):

dis = 1

for t in range(i, PageNum): # 从页面序列的第i个位置开始找起

if VirtualQueue[s] != PageOrder[t]:

dis += 1

else:

break

PageDisCount[s] = dis

point = 0

for m in range(1, MinBlockNum):

if PageDisCount[point] < PageDisCount[m]:

point = m

VirtualQueue[point] = PageOrder[i]

display()

LackPageRate = LackPageNum / PageNum

print(f"缺页数LackPageNum = {LackPageNum}")

print(f"缺页率LackPageRate = {LackPageRate}")

def LRU():

"""最近最久未使用LRU页面置换算法"""

print("\*\*\*\*\*\*\*\*\* 你选择了LRU算法：\*\*\*\*\*\*\*\*\* ")

print("页面置换情况如下:")

initial()

isInQueue = False

point = 0 # 指向最长时间未被访问的下标

for i in range(MinBlockNum, PageNum):

isInQueue = False

for k in range(MinBlockNum):

if VirtualQueue[k] == PageOrder[i]: # 如果当前页面在队列中

isInQueue = True

if not isInQueue:

global LackPageNum

LackPageNum += 1

point = 0

for j in range(1, MinBlockNum):

if LRUtime[point] < LRUtime[j]:

point = j

for s in range(MinBlockNum): # 其余页面对应的时间要+1

if VirtualQueue[s] != VirtualQueue[point]:

LRUtime[s] += 1

VirtualQueue[point] = PageOrder[i]

LRUtime[point] = 0

display()

else: # 负责更新当前对应页面的时间

for s in range(MinBlockNum): # 其余页面对应的时间要+1

if VirtualQueue[s] != PageOrder[i]:

LRUtime[s] += 1

else:

LRUtime[s] = 0

global LackPageRate

LackPageRate = LackPageNum / PageNum

print(f"缺页数LackPageNum = {LackPageNum}")

print(f"缺页率LackPageRate = {LackPageRate}")

def display():

"""显示当前的虚拟队列状态"""

for i in range(MinBlockNum):

if VirtualQueue[i] >= 0:

print(VirtualQueue[i], end=" ")

print()

if \_\_name\_\_ == "\_\_main\_\_":

input\_data()

isContinue = 1

while isContinue:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\* 请选择算法 \*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\* 1代表FIFO算法 \*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\* 2代表OPI算法 \*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\* 3代表LRU算法 \*\*\*\*\*\*\*\*\*\*")

chooseAlgorithm = int(input())

if chooseAlgorithm == 1:

FIFO()

elif chooseAlgorithm == 2:

OPI()

elif chooseAlgorithm == 3:

LRU()

else:

print("请输入正确的序号进行选择：")

print("\*\*\*\*\*\*\*\*\*\* 是否继续选择算法? \*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\* 输入1代表继续，输入0代表退出！ \*\*\*\*\*\*\*\*\*\*")

isContinue = int(input())

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**5.运行结果与分析：**

为了检查算法运行结果是否正确，用作业例题1来说明，如下图。

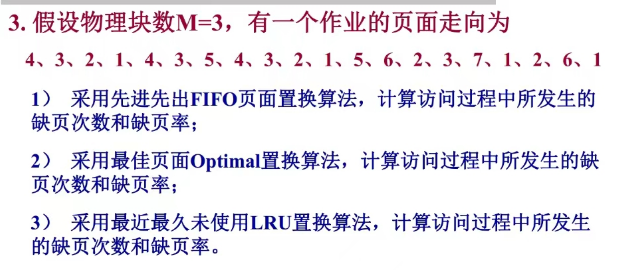


图 5-4 作业例题1

算法运行结果：

输出界面如下：

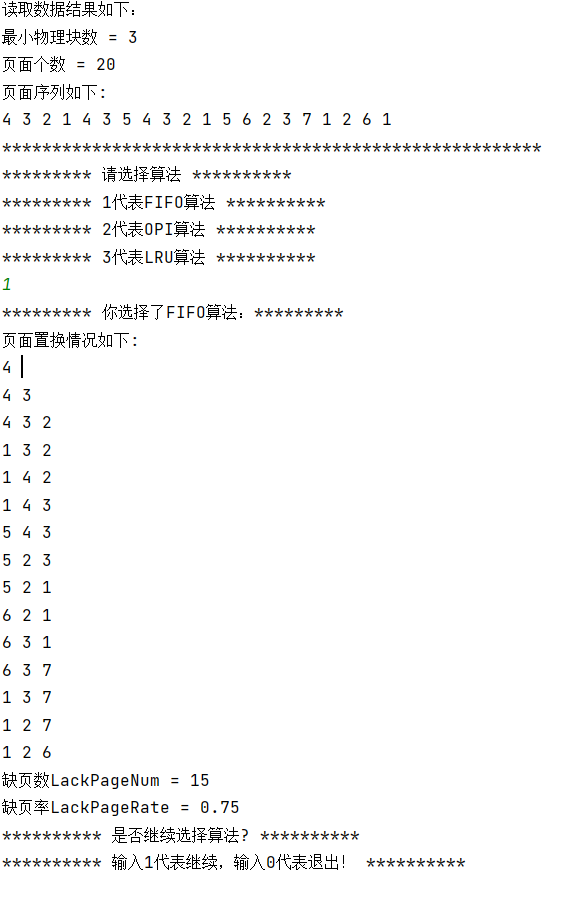


图 5-5 FIFO算法运行结果

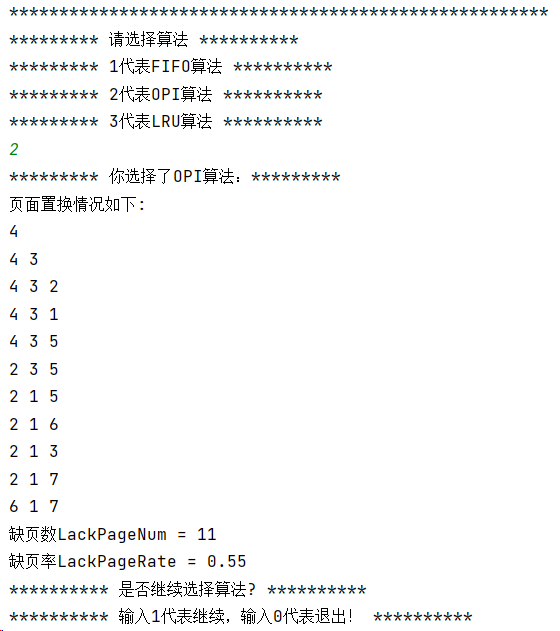


图 5-6 Optimal算法运行结果

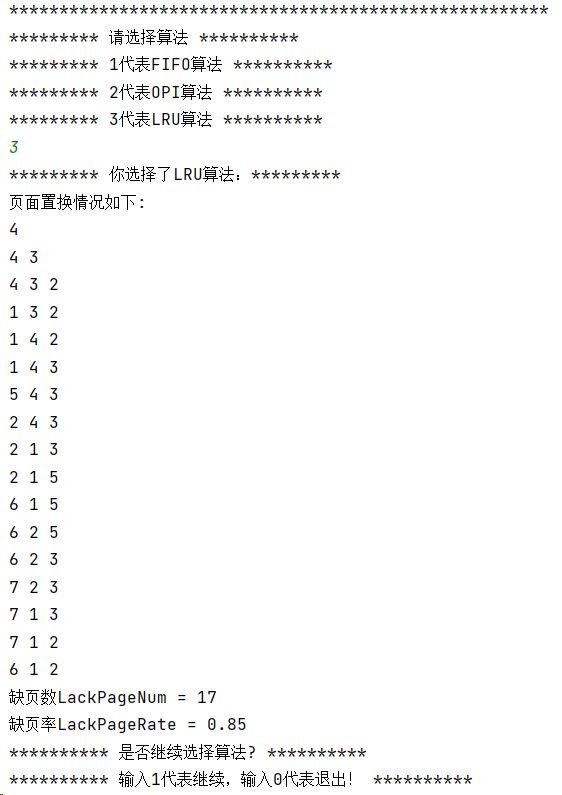
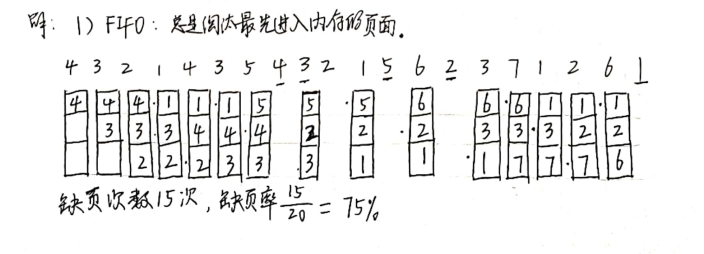


图 5-7 LRU算法运行结果

用于参考的正确答案是：  


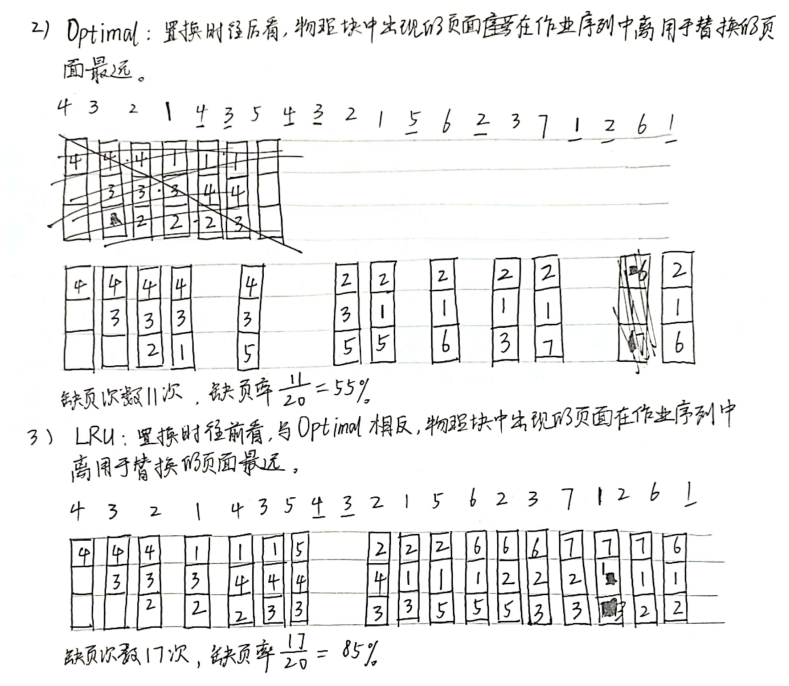


图 5-8 用于参考的正确结果

由三个算法运行结果和用于参考的正确结果可知，两者一致，算法的代码运行结果是有效的。其中，要注意，Optimal算法的运行结果里面，最后面的三个物理块中分别是6，1，7，用于参考的正确结果对应位置是2，1，6。这里都是正确的，因为Optimal算法是向后看，被置换的是2或7，在6之前，2和7并没有发生在6之后，所以6可以置换2，也可以置换7。

FIFO算法：简单但不总是最优，特别是在页面使用频繁变动的情况下。

OPI算法：理论上最优，但在实际中难以实现，因为无法准确知道未来的页面访问情况。

LRU算法：更加接近实际应用，通过使用最近的访问记录来预测未来的页面访问。

**五、实验总结**

1.

代码不够健壮:

问题：代码设计不够健壮，例如缺乏对特殊情况（如空输入文件）的处理。

解决办法：增加对特殊情况的检查和处理。

2.

数组索引越界:

如果PageNum的值大于MaxNumber，在访问PageOrder, PageDisCount, LRUtime等数组时可能会出现索引越界。

解决办法：在读取数据时检查PageNum是否超出了MaxNumber的限制。

3.

FIFO算法：简单但不总是最优，特别是在页面使用频繁变动的情况下。

OPI算法：理论上最优，但在实际中难以实现，因为无法准确知道未来的页面访问情况。

LRU算法：更加接近实际应用，通过使用最近的访问记录来预测未来的页面访问。