OS-lec9-exercise

张盛豪计 41 2014011450

2017-03-24 21:53:35

Lec9 练习题

1. 物理页帧数量为 3，且初始时没有对应的虚拟页。虚拟页访问序列为 0,1,2,0,1,3,0,3,1,0,3，请问采用最优置换算法的缺页次数为（4）

最优置换算法：置换未来最长时间不访问的页面。

时间	1	2	3	4	5	6	7	8	9	10
物理页帧	0	1	2	0	1	3	0	3	1	0
0	0	0	0	0	0	0	0	0	0	0
1		1	1	1	1	1	1	1	1	1
2			2	2	2	3	3	3	3	3
缺页?	缺页	缺页	缺页			缺页 2->3				

缺页次数为 4 次

2. 物理页帧数量为 3，且初始时没有对应的虚拟页。虚拟页访问序列为 0,1,2,0,1,3,0,3,1,0,3，请问采用 FIFO 置换算法的缺页次数为（6）

时间	1	2	3	4	5	6	7	8	9	10
物理页帧	0	1	2	0	1	3	0	3	1	0
0	0	0	0	0	0	3	3	3	3	3
1		1	1	1	1	1	0	0	0	0
2			2	2	2	2	2	2	1	1
缺页?	缺页	缺页	缺页			缺页 0->3	缺页 1->0		缺页 2->1	

缺页 6 次

3. 物理页帧数量为 4，且初始时没有对应的虚拟页。虚拟页访问序列为 0,3,2,0,1,3,4,3,1,0,3,2,1,3,4，请问采用 CLOCK 置换算法（用 1 个 bit 表示存在时间）的缺页次数为（9）

缺页次数 9

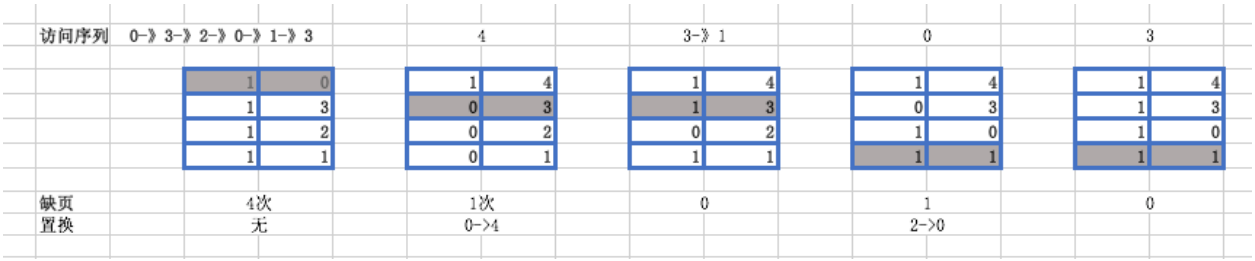

Figure 1: 题 3Clock 算法 1

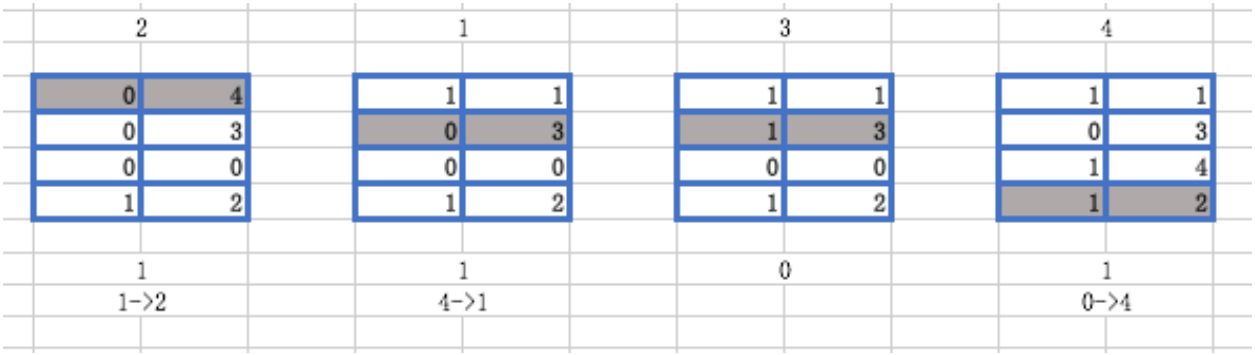

Figure 2: 题 3Clock 算法 2

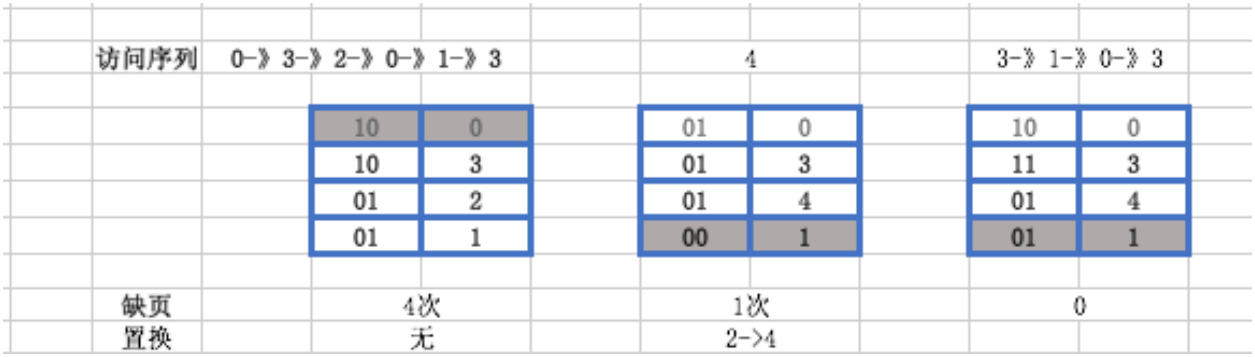

Figure 3: 题 4-1

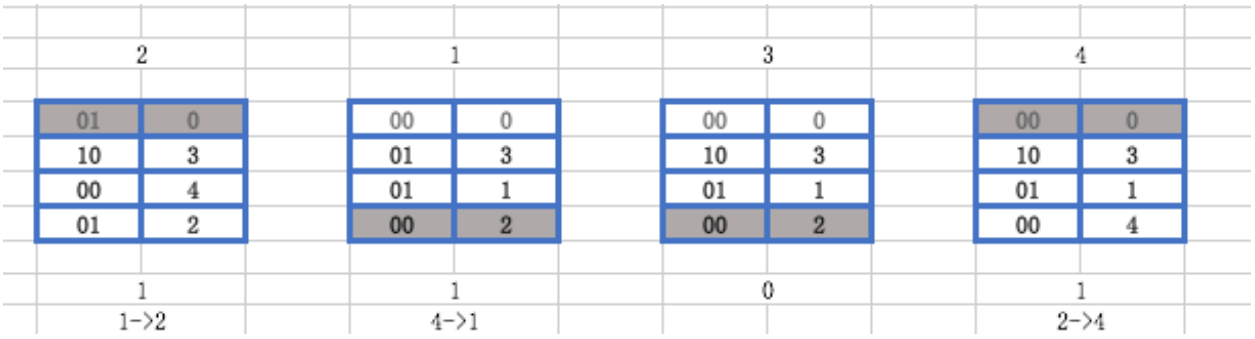

Figure 4: 题 4-2

4. 物理页帧数量为 4，且初始时没有对应的虚拟页。虚拟页访问序列为 0,3,2,0,1,3,4,3,1,0,3,2,1,3,4，请问采用 CLOCK 置换算法（用 2 个关联，bit 表示存在时间，可以表示 4,）的缺页次数为 ()

缺页次数为 8

5.（spoc）根据你的学号 mod 4 的结果值，确定选择四种页面置换算法（0：LRU 置换算法，1：改进的 clock 页置换算法，2：工作集页置换算法，3：缺页率置换算法）中的一种来设计一个应用程序（可基于 python,ruby,C，C++，LISP 等）模拟实现，并给出测试用例和测试结果。请参考如 python 代码或独自实现。

工作集页置换算法

```
# coding: utf-8
# ! /usr/bin/env python

# 工作集页面置换算法

# 访存页面链表
work_list = []
# 工作集
work_set = set(work_list)
# 窗口大小
t = 4


def access(page):
    """
    访问页面函数,针对每一个页,进行工作集的更新,访存链表的更新,页面置换
    :param page:
    :return:
    """
    global work_set
    global work_list
    # 是否出现页缺失
    miss = False
    if page in work_set:
        # 当前页面在工作集中,表示命中
        print "命中页面"
    else:
        miss = True
        print "页缺失"
    work_list.append(page)
    # 删除第一个页面
    first = work_list[0]
    if len(work_list) > t:
        # 已经超出工作集,去除列表首页
        del work_list[0]
    work_set = set(work_list)
    if first not in work_set:
        print 'page %s 换出' % first
    if miss:
        print "page %s 换入" % page
    print "当前访存链表:", work_list
    print "当前工作集:", work_set
```

```
def access_pages(pages, input_work_list):
    global work_set
    global work_list
    work_list = input_work_list
    work_set = set(input_work_list)
    print "页面访问顺序为:", pages
    print "初始工作集:", work_set
    print "初始访问列表:", work_list
    pages = pages.split(',')
    count = 0
    for i in pages:
        count += 1
        print 'Page ', count, i,
        access(i)
        print ''
    # 访问完成之后清空
    work_list = []
    work_set = set(work_list)

if __name__ == '__main__':
    print "这是一个工作集置换算法的简单实现"

    log = '''
    t = 4
    test_pages = "c,c,d,b,c,e,c,e,a,d"
    test_work_str = "e,d,a"
    '''

    print "课件中测试集:", log
    t = input("窗口大小t=")
    assert (int(t) > 0)
    # 访问页面顺序
    test_pages = raw_input('测试页面顺序(以,隔开,如e,d,a,c,c) test_pages=')
    test_work_str = raw_input('此前访问的t个页面顺序为(如e,d,a)test_work_str=')
    test_work_list = test_work_str.split(',')
    print test_work_list
    assert (len(test_work_list) <= t)
    # test_pages = "e,d,a,c,c,d,b,c,e,c,e,a,d"
    access_pages(test_pages, test_work_list)
```

测试用例使用课本中的访问顺序

```
页面访问顺序为: c,c,d,b,c,e,c,e,a,d
初始工作集: set(['a', 'e', 'd'])
初始访问列表: ['e', 'd', 'a']
Page  1 c 页缺失
page c 换入
当前访存链表: ['e', 'd', 'a', 'c']
当前工作集: set(['a', 'c', 'e', 'd'])

Page  2 c 命中页面
page e 换出
当前访存链表: ['d', 'a', 'c', 'c']
当前工作集: set(['a', 'c', 'd'])
```

```
Page  3 d 命中页面
当前访存链表: ['a', 'c', 'c', 'd']
当前工作集: set(['a', 'c', 'd'])

Page  4 b 页缺失
page a 换出
page b 换入
当前访存链表: ['c', 'c', 'd', 'b']
当前工作集: set(['c', 'b', 'd'])

Page  5 c 命中页面
当前访存链表: ['c', 'd', 'b', 'c']
当前工作集: set(['c', 'b', 'd'])

Page  6 e 页缺失
page e 换入
当前访存链表: ['d', 'b', 'c', 'e']
当前工作集: set(['c', 'b', 'e', 'd'])

Page  7 c 命中页面
page d 换出
当前访存链表: ['b', 'c', 'e', 'c']
当前工作集: set(['c', 'b', 'e'])

Page  8 e 命中页面
page b 换出
当前访存链表: ['c', 'e', 'c', 'e']
当前工作集: set(['c', 'e'])

Page  9 a 页缺失
page a 换入
当前访存链表: ['e', 'c', 'e', 'a']
当前工作集: set(['a', 'c', 'e'])

Page  10 d 页缺失
page d 换入
当前访存链表: ['c', 'e', 'a', 'd']
当前工作集: set(['a', 'c', 'e', 'd'])
```

6. 请判断 OPT、LRU、FIFO、Clock 和 LFU 等各页面置换算法是否存在 Belady 现象? 如果存在，给出实例；如果不存在，给出证明。

FIFO: 存在 Belady 现象，例子

物理页面增加，缺失数次数反而升高，出现了 Belady 现象

LRU: 不存在 Belady 现象

证明：对于确定的页面访问序列 PagesList，设其分配的物理页面数为 n，当访问 PagesList[i] 页面时发生了缺页（易知 i>n），即页面 PagesList[i] 不在页面 i-n 到 i-1 这 n 个页面中，进而发生了缺页异常，而加大 n 至 n1，PagesList[i] 有可能在 i-n1 至 i-1 这 n1 个页面中，也可能不在，前者此时不会发生缺页异常，后者仍然会发生，但无论如何，分配的物理页面数增加时，其缺页数可能持平，也可能减少，但不可能增加。

物理页帧数为3，缺失9次												
时间	1	2	3	4	5	6	7	8	9	10	11	12
物理页帧	3	2	1	0	3	2	4	3	2	1	0	4
0	3	2	1	0	3	2	4	4	4	1	0	0
1		3	2	1	0	3	2	2	2	4	1	1
2			3	2	1	0	3	3	3	2	4	4
缺页?	缺页	缺页	缺页	缺页	缺页	缺页	缺页			缺页	缺页	

物理页帧数为4，缺页10次												
时间	1	2	3	4	5	6	7	8	9	10	11	12
物理页帧	3	2	1	0	3	2	4	3	2	1	0	4
0	3	2	1	0	0	0	4	3	2	1	0	4
1		3	2	1	1	1	0	4	3	2	1	0
2			3	2	2	2	1	0	4	3	2	1
3				3	3	3	2	1	0	4	3	2
缺页?	缺页	缺页	缺页	缺页			缺页	缺页	缺页	缺页	缺页	缺页

Figure 5: FIFO 算法

OPT：不存在 Belady 现象

最优置换算法置换出未来最长时间不访问的页面，这就意味着分配物理页数目为 n 时，置换一次页面之后，至少 n 次访问不会出现缺页，而 n 增大时，可以将缺页继续向后推迟，进而 n 增大时缺页数不可能会反而增加

LFU,Clock：不存在 Belady 现象

实际上，堆栈式页面置换算法都不会产生 Belady 现象，LRU,OPT，LFU,Clock 均属于堆栈式页面置换算法，在堆栈式页面置换算法中，页面优先级的分配是独立于页面帧数的，对于堆栈式页面置换算法，分配的物理页面帧数增加时可以使得其缺页数减少，而 FIFO 这类算法，其替换页面的优先级决定于其分配的物理页面帧数，因此在这种情况下可能会出现 Belady 现象，而堆栈式算法则不会出现这样的状况。

参考资料：

- Efficient (stack) algorithms for analysis of write-back and sector memories [http://www.eecs.berkeley.edu/Pubs/Tech
  87-358.pdf section 1.3]
- Principles of Computer Operating Systemm Stack Page Replacement Policies [http://cseweb.ucsd.edu/classes/sp02/cse12