实 验 报 告

课程名称：操作系统试验

实 验 三：内存页面置换算法

班　　级：2020 级计算机科学与技术

学生姓名：胡诚皓

学　　号：20201060330

专　　业：2020 级计算机科学与技术

指导教师：杨旭涛

学　　期：2022－2023 学年秋季学期

成　　绩：

云南大学信息学院

一、实验目的

1、掌握内存的分区、分页和分段管理的基本概念和原理，掌握内存的虚拟空间和物理空间的对应关系；

2、掌握内存分配中的连续和非连续分配、固定分配和动态分配等概念，掌握几种内存分配方法的分配过程和回收过程；

3、掌握内存的页面置换算法，包括先进先出（FIFO），最近最久未使用（LRU），最不经常适用（LFU），最近未使用（NRU），最佳置换（OPT）等方法，以及理解算法间的优劣差别，了解缺页率、belady 现象等内容。

二、知识要点

1、内存的虚拟地址和物理地址映射；

2、内存的分区管理、分页管理、分段管理和段页式管理；

3、页面置换算法，包括先进先出（FIFO），最近最久未使用（LRU），最不经常适用（LFU），最近未使用（NRU），最佳置换（OPT）等方法。

三、实验预习（要求做实验前完成）

1、了解 linux 系统中常用命令的使用方法；

2、掌握内存的虚拟地址和物理地址的描述；

3、掌握内存的分页管理的基本原理和过程；

4、掌握基本的内存页面置换算法，包括先进先出（FIFO），最近最久未使用（LRU）等。

四、实验内容和试验结果

结合课程所讲授内容以及课件中的试验讲解，完成以下试验。请分别描述程序的流程，附上源代码，并将试验结果截图附后。

模拟整体程序设计

下面进行的模拟内存管理都使用同样的程序框架，唯一区别在于在发生缺页中断后，若内存中的空闲页面已经全部被占用，选择的用于换出的页不同。另外，为了方便观察，在下面的测试中，页面大小均设为 1。下面是对统一的部分的算法描述。

① 询问用户要设置的页面数量（即内存中能同时存在的页的个数）和页面的大小，以及要执行的访存序列。

② 初始化页列表和页面的列表，一开始内存中的所有页面都是空闲页面，将初始化的结果直接作为空闲页面链表即可。

③ 遍历获取访存序列中下一个要访问的内存地址。若完成了序列中的所有访存，转到⑦

④ 将内存地址转换为要访问的虚页号。若该页对应的页面已经在内存中，就直接访问，更新该页的相关字段，记录当前内存中页的状态后转到③

⑤ 若该页对应的页面并不在内存中，需要从外存调入，即发起缺页中断。若内存中仍有空闲页面，则取一个空闲页面，并将需要的页面换入到该空闲页面，记录当前内存中页的状态后转到③

⑥ 若该页对应的页面并不在内存中且内存中已经没有空闲页面，调用置换算法完成淘汰和换入换出操作，记录当前内存中页的状态后转到③

⑦ 构建页面分配过程表，计算缺页率，进行显示

主函数代码如下

```
def main(replace_algorithm, visit_seq):
    global page_size
    # 要求用户输入页面数量与页数量
    total_pf = int(input("设置页面数量: "))
    total_vp = max(visit_seq) + 1
    page_size = int(input("设置页面大小: "))

    global cpu_time
    global record
    dismiss = 0
```

```
        visit_cnt = 0

        # 初始化后，页面都是空闲的
        # freef、busyf 分别为空闲页面链表和正在使用的页面链表，下标为 0 视为队头
        page_list, freef = initialize(total_pf, total_vp)
        busyf = []
        # 逐个访问地址序列
        for addr in visit_seq:
            # 计算得到虚页号
            vaddr = addr // page_size
            # 页访问范围的控制
            if vaddr >= total_vp:
                print("地址{}超出页的范围，跳过".format)
                continue
            visit_cnt += 1
            # 要访问的页对应的页面在内存中，直接访问即可
            if page_list[vaddr].pframe_num >= 0:
                # 更新访问时间及访问次数
                page_list[vaddr].timestamp = cpu_time
                page_list[vaddr].counter += 1
            else:
                # 要访问的页对应的页面不在内存中，而在外存中，此时需要换入，即产生缺页中断
                dismiss += 1
                # 内存中仍有空闲页面，直接使用即可
                if len(freef) > 0:
                    # 访问之
                    page_list[vaddr].pframe_num = freef[0].pframe_num
                    page_list[vaddr].timestamp = cpu_time
                    page_list[vaddr].counter = 1
                    freef[0].page_num = vaddr
                    # 将该页面加入 busy
                    busyf.append(freef[0])
                    del freef[0]
                else:
                    # 内存中没有空闲页面了，调用置换算法完成换入换出操作
                    replace_algorithm(vaddr, page_list, busyf)
            cpu_time += 1
            # 记录每次分配完成后的页面情况
            record.append(tuple([i.page_num for i in busyf]))
        display_table(visit_seq, record, total_pf)
```

```
        print("缺页率为: {}%".format(round(dismiss*100 / visit_cnt, 2)))
```

1、模拟内存的页式管理，实现内存的分配和调用，完成虚拟内存地址序列和物理内存的对应。在内存调用出现缺页时，调入程序的内存页。在出现无空闲页面时，使用先进先出（FIFO）算法实现页面置换。

总体上 FIFO 使用队列完成淘汰页面的选择，选择队首的即可。

算法描述

① 获取已使用的页面队列的队首元素

② 将该页面对应的页换出

③ 将需要访问页的页面换入内存，加入已使用页面队列

实际测试

使用访存序列 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4 进行测试，分别设置页面数量为 3 和 4，得到结果如下：

```
>>>
====================================== RESTART: E:\Library\Desktop\tmp.py ===========
请输入访问序列（以空格分隔）
0 1 2 3 0 1 4 0 1 2 3 4
设置页面数量：3
设置页面大小：1
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|       | 0   | 1   |   2 |   3 |   0 |   1 |   4 |   0 |   1 |   2 |   3 |   4 |
+=======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| 页面0 | 0   | 0   |   0 |   1 |   2 |   3 |   0 |   0 |   0 |   1 |   4 |   4 |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面1 |     | 1   |   1 |   2 |   3 |   0 |   1 |   1 |   1 |   4 |   2 |   2 |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面2 |     |     |   2 |   3 |   0 |   1 |   4 |   4 |   4 |   2 |   3 |   3 |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
缺页率为：75.0%
```

```
>>>
====================================== RESTART: E:\Library\Desktop\tmp.py ============
请输入访问序列（以空格分隔）
0 1 2 3 0 1 4 0 1 2 3 4
设置页面数量：4
设置页面大小：1
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|       | 0   | 1   | 2   |   3 |   0 |   1 |   4 |   0 |   1 |   2 |   3 |   4 |
+=======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| 页面0 | 0   | 0   | 0   |   0 |   0 |   0 |   1 |   2 |   3 |   4 |   0 |   1 |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面1 |     | 1   | 1   |   1 |   1 |   1 |   2 |   3 |   4 |   0 |   1 |   2 |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面2 |     |     | 2   |   2 |   2 |   2 |   3 |   4 |   0 |   1 |   2 |   3 |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面3 |     |     |     |   3 |   3 |   3 |   4 |   0 |   1 |   2 |   3 |   4 |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
缺页率为：83.33%
```

很显然，使用 FIFO 算法进行页面置换时，当页面数量增加时，缺页率反而会增加，产生了 Belady 现象。

FIFO 置换策略的代码如下

```
# 先进先出淘汰策略
def FIFO(vaddr, page_list, busyf):
    global record
    # 将 busy 中的第一个页面换出
    tmp = busyf[0]
    page_list[tmp.page_num].pframe_num = -1
    # 将需要的页换入
    page_list[vaddr].pframe_num = tmp.pframe_num
    page_list[vaddr].timestamp = cpu_time
    page_list[vaddr].counter = 1
    # 为了维持队列，将该元素重新插到队尾
    del busyf[0]
    tmp.page_num = vaddr
    busyf.append(tmp)

    return 1
```

2、参考第一题的页式内存管理，在出现无空闲页面时，改使用最近最久未使用（LRU）算法。

算法描述

① 遍历内存中存在的所有页面，通过页的信息找到最后使用时间戳最小的（即最久未使用的）

② 将该页换出内存

③ 将需要的页存到这个页面的位置，并更新相关的访问信息（包括最后访问时间戳，访问计数器等）

实际测试

使用和上面测试 FIFO 淘汰算法同样的序列，在页面数量为 3、4 情况下的淘汰率

```
>>>
========================= RESTART: E:\Library\Desktop\tmp.py =========================
请输入访问序列（以空格分隔）
0 1 2 3 0 1 4 0 1 2 3 4
设置页面数量：3
设置页面大小：1
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|       | 0   | 1   |  2  |  3  |  0  |  1  |  4  |  0  |  1  |  2  |  3  |  4  |
+=======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| 页面0 | 0   | 0   |  0  |  3  |  3  |  3  |  4  |  4  |  4  |  2  |  2  |  2  |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面1 |     | 1   |  1  |  1  |  0  |  0  |  0  |  0  |  0  |  0  |  3  |  3  |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面2 |     |     |  2  |  2  |  2  |  1  |  1  |  1  |  1  |  1  |  1  |  4  |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
缺页率为：83.33%
```

```
>>>
========================= RESTART: E:\Library\Desktop\tmp.py =========================
请输入访问序列（以空格分隔）
0 1 2 3 0 1 4 0 1 2 3 4
设置页面数量：4
设置页面大小：1
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|       | 0   | 1   | 2   |  3  |  0  |  1  |  4  |  0  |  1  |  2  |  3  |  4  |
+=======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| 页面0 | 0   | 0   | 0   |  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |  4  |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面1 |     | 1   | 1   |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面2 |     |     | 2   |  2  |  2  |  2  |  4  |  4  |  4  |  4  |  3  |  3  |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 页面3 |     |     |     |  3  |  3  |  3  |  3  |  3  |  3  |  2  |  2  |  2  |
+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
缺页率为：66.67%
```

Belady 现象不复存在，但对于这个序列，LRU 算法的效果未优于 FIFO。

LRU 置换策略的代码如下：

```
# 最近最久未使用淘汰策略
def LRU(vaddr, page_list, busyf):
    earliest = (0, page_list[busyf[0].page_num].timestamp)
    # 在所有页中遍历找出使用时间戳最早的，将其换出
    for idx, i in enumerate(busyf):
        page = page_list[i.page_num]
        if page.timestamp < earliest[1]:
            earliest = (idx, page.timestamp)
    page_list[busyf[earliest[0]].page_num].pframe_num = -1
    # 将需要的页换入
    page_list[vaddr].pframe_num = busyf[0].pframe_num
    page_list[vaddr].timestamp = cpu_time
    page_list[vaddr].counter = 1
    busyf[earliest[0]].page_num = vaddr
```

```
        return 1
```

3、对比前两题实现的页面置换算法，以相同的内存调用序列数据做实验，输出缺页率，尝试讨论它们的差别。

在上面两题中的实际测试部分,已经完成使用相同内存调用序列数据，可以发现FIFO算法出现了Belady现象，即当可使用的内存页面数量增加时，缺页率反而增加了。

FIFO算法会出现Belady现象的原因，本质上是在于它并没有利用到程序的局部性原理，在内存中存在时间越久，就越容易被换出，这显然是非常不合理的。根据程序的局部性原理，在内存中存在时间久的很有可能是需要经常使用到的页面,这样的页面应该尽可能被保留下来才对。FIFO算法在按线性顺序访问地址空间时是相对理想的，否则效率不高。但是在实现的复杂度上来看,FIFO算法所需要的计算量明显低于LRU，甚至不需要计算，只需要维持一个队列即可。

LRU算法就考虑到了程序的局部性原理，将最久没有使用的页面淘汰。这事实上也是一种简化的思想，有可能需要经常使用的页面在近期刚好用不太到，在更长的时间尺度内是经常用到的，这种情况下LRU算法就会出现抖动现象。LRU常常被认为是效果相当好的一种淘汰算法，但是实现它需要的计算量和硬件支持要求较高。

此处另外实现了LFU与NUR算法,NUR的具体实现使用的是CLOCK算法，以下是使用相同的100组随机序列（访存范围在0-9之间，序列长度为12）对四种算法的测试，计算平均缺页率

	FIFO	LRU	LFU	NUR
页面数量为3	75.67%	75.25%	76.83%	75.25%

页面数量为 4	69.83%	69.75%	70.58%	69.50%

从上表的结果可以得出如下结论：

① LRU 和 NUR 结果相近，在观察分配过程时发现在很多情况下，NUR 算法会退化成 LRU。

② FIFO 算法和其他几种算法的结果相近，也就是说总的来看 Belady 现象对 FIFO 结果的影响并不很大。

③ 总的来看，页面数量的增加对各种算法的效果都有很大的提升。

LFU 与 NUR 的代码

```
    # 最近最少使用淘汰算法
    def LFU(vaddr, page_list, busyf):
        least_use = (0, page_list[busyf[0].page_num].counter)
        # 在所有页中遍历找出使用次数最少的，将其换出
        for idx, i in enumerate(busyf):
            page = page_list[i.page_num]
            if page.counter < least_use[1]:
                least_use = (idx, page.counter)

        page_list[busyf[least_use[0]].page_num].pframe_num = -1
        # 将需要的页换入
        page_list[vaddr].pframe_num =
busyf[least_use[0]].pframe_num
        page_list[vaddr].timestamp = cpu_time
        page_list[vaddr].counter += 1
        busyf[least_use[0]].page_num = vaddr
```

```
    # 最近没有使用的淘汰策略，此处具体实现使用 CLOCK 算法
    def NUR(vaddr, page_list, busyf):
        global NUR_pt
        if "NUR_pt" not in globals():
            NUR_pt = 0
        flag = True
        # 标志位仍利用 counter 字段
        while flag:
```

```
            # 当前检查的页面对应的页
            page = page_list[busyf[NUR_pt].page_num]
            # 若未访问过，就换出
            if page.counter == 0:
                page_list[busyf[NUR_pt].page_num].pframe_num = -
1
                flag = False
                break
            else:
                # 访问过，将访问位置零，继续循环查找
                page.counter = 0
                NUR_pt = (NUR_pt + 1) % len(busyf)
                continue

        # 换入需要的页
        page_list[vaddr].pframe_num = busyf[NUR_pt].pframe_num
        page_list[vaddr].timestamp = cpu_time
        page_list[vaddr].counter = 1
        busyf[NUR_pt].page_num = vaddr
        # 换入后指针后移一个
        NUR_pt = (NUR_pt + 1) % len(busyf)
```

下面是以 word 附件形式附加的 Python 源代码，双击可以打开。若被 Office 阻止访问无法打开，可以选中后进行复制，粘贴到任一文件夹中即可。

exp3.py