

# 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：02 班

学生姓名：白文强

学 号：20191060064

专 业：计算机科学与技术

指导教师：杨旭涛

学 期：2021—2022 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。

数据结构：

```
//页表
typedef struct page_struct {
    int pn;    //页号
    int pfn;   //页面号
    int time;  //时间
} page_type;
page_type page_table[TOTAL_VP];
```

```

//页框，free 和 busy 链表
typedef struct page_node {
    int pn;
    int pfn;
    struct page_node *next;
} pfc;

```

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。

```

void FIFO(int total_pf) {
    //total_pf 表示页面数
    initialize(total_pf);
    diseffect = 0;
    pfc *p;
    busy_head = busy_tail = NULL;
    for (int i = 0; i < total_instructions; i++) {
        //找到需要的页号
        int pn = instructions[i].pn;
        if (page_table[pn].pfn == INVALID) {
            //页面不存在
            diseffect++;

            if (free_head == NULL) {
                //从 busy 链表中取出放入 free 链表
                p = busy_head->next; //保留
                page_table[busy_head->pn].pfn = INVALID;
                free_head = busy_head;
                free_head->next = NULL; ////改的这里
                busy_head = p;
            }
            //从 free 中取出
            p = free_head->next;
            free_head->next = NULL;
            free_head->pn = pn;
            page_table[pn].pfn = free_head->pfn;

            //放入 busy 链表
            if (busy_tail == NULL) {
                busy_head = busy_tail = free_head;
            } else {
                busy_tail->next = free_head;
                busy_tail = free_head;
            }
        }
    }
}

```

```

    }
    free_head = p;
}
}
printf("FIFO:%.2f%%\n", (1 - (float) diseffect / total_instructions) * 100);
}

```

## FIFO 算法流程:

```

DO{
    访问当前页
    IF(当前页在内存中)
        直接访问
    ELSE
        缺页计数+1
        IF(free 链表非空)
            直接将该页换入空闲页面中
            将该页框加入到 busy 链表中
        ELSE
            在 busy 链表中选择最先进入内存的一个页淘汰
            将其加入到 free 链表中
            从 free 链表中取出一个页框
            将该页换入到取出的页框中
        访问
    }
}

```

## 运行结果:

设进程共有 8 个页，程序访问的顺序为：7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1，在内存中分配三个页面和四个页面，计算命中率：

```

bwq@ubuntu:~/桌面/C$ gcc FIFOandLRU.c -o FIFOandLRU.out
bwq@ubuntu:~/桌面/C$ ./FIFOandLRU.out
Query Order:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1
Three Pages FIFO:29.41%
Four Pages FIFO:47.06%

```

三个页面，命中率：5/17 = 29.41%

|   | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 |   | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | i | i | i | i |
| 5 |   |   | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 |

四个页面，命中率：8/17=47.06%

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 |
| 7 | 7 | 7 | 7 | 7 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 |
|   | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
|   |   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|   |   |   | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |

当访问顺序变为 1,2,3,4,1,2,5,1,2,3,4,5 时，出现 Belady 现象：

```
bwq@ubuntu:~/桌面/C$ gcc FIFOandLRU.c -o FIFOandLRU.out
bwq@ubuntu:~/桌面/C$ ./FIFOandLRU.out
Query Order:
1 2 3 4 1 2 5 1 2 3 4 5
Three Pages FIFO:25.00%
Four Pages FIFO:16.67%
```

三个页面，命中率  $3/12 = 25\%$

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 1 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
|   | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 3 |
|   |   | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 4 | 4 |

四个页面，命中率：  $2/12=16.67\%$ ，命中率不增反降

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 4 | 4 |
|   | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 5 |
|   |   | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 |
|   |   |   | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 |

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。

```
void LRU(int total_pf) {
    int min_time;
    int min_j;
    pfc *p = NULL, *prep = NULL;
    diseffect = 0;
    initialize(total_pf);
```

```

int present_time = 0;
for (int i = 0; i < total_instructions; i++) {
    int pn = instructions[i].pn;
    if (page_table[pn].pfn == INVALID) {
        diseffect++;
        if (free_head == NULL) {
            //找到最早进入的页面
            min_time = 65535;
            for (int j = 0; j < TOTAL_VP; j++) {
                if (page_table[j].time < min_time && page_table[j].pfn !=
INVALID) {
                    min_time = page_table[j].time;
                    min_j = j;
                }
            }
            prep = NULL;
            p = busy_head;
            while (p != NULL) {
                if (p->pn == min_j) {
                    break;
                }
                prep = p;
                p = p->next;
            }

            //从 busy 链表中取出
            if (prep == NULL) {
                busy_head = p->next;
            } else {
                prep->next = p->next;
            }
            //放入 free 链表
            free_head = p;
            free_head->next = NULL;

            page_table[min_j].pfn = INVALID;
            page_table[min_j].time = -1;
        }

        //从 free 中取出一个 p
        p = free_head;
        free_head = free_head->next;
        p->pn = pn;
        p->next = NULL;
    }
}

```

```

        //把 p 插入 busy 链表
        if (busy_tail == NULL) {
            busy_head = p;
        } else {
            busy_tail->next = p;
        }

        busy_tail = p;

        page_table[pn].pfn = p->pfn;
        page_table[pn].time = present_time;
    } else {
        page_table[pn].time = present_time;
    }
    present_time++;
}
printf("LRU:%.2f%%\n", (1 - (float) diseffect / total_instructions) * 100);
}

```

## LRU 算法流程

```

DO{
    访问当前页
    IF(当前页在内存中)
        直接访问
        页表中对应的页 time 值更新为 present_time
    ELSE
        缺页计数+1
        IF(free 链表非空)
            直接将该页换入空闲页面中
            将该页框加入到 busy 链表中
        ELSE
            在 busy 链表选择 time 最小的页框淘汰
            将其加入到 free 链表中
            从 free 链表中取出一个页框
            将该页换入到取出的页框中
            页表中对应的页 time 值更新为 present_time
        访问
    }
}

```

## 运行结果：

设进程共有 8 个页，程序访问的顺序为：4, 7, 0, 7, 1, 0, 1, 2, 1, 2, 6，在内存中分配三个页面和五个页面，计算命中率：

```
bwq@ubuntu:~/桌面/C$ ./FIFOandLRU.out
Query Order:
4 7 0 7 1 0 1 2 1 2 6
Five Pages LRU:45.45%
```

五个页面，命中率：5/11=45.45%

| 4 | 7 | 0 | 7 | 1 | 0 | 1 | 2 | 1 | 2 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   | 2 | 1 | 2 | 6 |
|   |   |   |   | 1 | 0 | 1 | 1 | 2 | 1 | 2 |
|   |   | 0 | 7 | 7 | 1 | 0 | 0 | 0 | 0 | 1 |
|   | 7 | 7 | 0 | 0 | 7 | 7 | 7 | 7 | 7 | 0 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 7 |

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。

内存调用序列：7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1

```
bwq@ubuntu:~/桌面/C$ gcc FIFOandLRU.c -o FIFOandLRU.out
bwq@ubuntu:~/桌面/C$ ./FIFOandLRU.out
Query Order:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1
Three Pages FIFO:29.41%
Three Pages LRU:35.29%
Four Pages FIFO:47.06%
Four Pages LRU:58.82%
```

可以看出，在同样的内存调用序列和同样数量的页框数的情况下，LRU 算法的命中率比 FIFO 算法的命中率高，缺页率低。

LRU 算法的思想是：如果某一页被访问了，那么它很可能马上又被访问；反之，如果某一页很长时间没有被访问，那么最近也不太可能会被访问。与 FIFO 算法的先入先出相比，更符合实际的页面置换，增加页面的命中率，提高访问速度。

## 五、问题讨论

1、说明程序中如何描述一个页面。

```
typedef struct page_node {
    int pn;
    int pfn;
    struct page_node *next;
} pfc;
```

页面用上面的数据结构表示，其中 pn 表示该页面在页表中的页号，pfn 表示该页面在内存中的页面号。不同的页面用链表进行串联。



## 2、说明程序中如何管理空闲页框。

空闲页框和在内存中的页面分别用一个链表来管理，`free_head` 表示空闲页框链表的头部，`busy_head` 表示在内存中的页面链表的头部。`free_head` 链表和 `busy_head` 链表共同表示该进程的页面。

当需要访问的页面不在内存中时，则需要从 `free_head` 链表中取出一个空闲页框，将需要的页面换入该页框中。如果 `free_head` 链表为空，则需要从 `busy_head` 链表中淘汰一个页面，将其放入 `free_head` 链表中。