扩展练习 Challenge：实现不考虑实现开销和效率的LRU页替换算法（需要编程）

小组成员：**于成俊　李帅东　楚乾靖**

方案一

设计思想和分析

我们发现，`*(unsigned char *)0x3000 = 0x0c;`会改变相应地址的`page->pra_vaddr`，所以我们想修改测试函数，让每次输入的值递增，即让每次输入的值充当page的访问时间。我们修改测试函数如下：

```
static int
_lru_check_swap(void) {

    int i = 0x10;

    cprintf("---------LRU check begin----------\n");
    cprintf("write Virt Page 3 in lru_check_swap\n");
    *(unsigned char *)0x3000 = i;i+=1;
    assert(pgfault_num==4);
    cprintf("write Virt Page 1 in lru_check_swap\n");
    *(unsigned char *)0x1000 =  i;i+=1;
    assert(pgfault_num==4);
    cprintf("write Virt Page 4 in lru_check_swap\n");
    *(unsigned char *)0x4000 =  i;i+=1;
    assert(pgfault_num==4);
    cprintf("write Virt Page 2 in lru_check_swap\n");
    *(unsigned char *)0x2000 =  i;i+=1;
    assert(pgfault_num==4);
    cprintf("write Virt Page 5 in lru_check_swap\n");
    *(unsigned char *)0x5000 =  i;i+=1;
    assert(pgfault_num==5);
    cprintf("write Virt Page 3 in lru_check_swap\n");
    *(unsigned char *)0x3000 =  i;i+=1;
    assert(pgfault_num==6);
    cprintf("write Virt Page 1 in lru_check_swap\n");
    *(unsigned char *)0x1000 =  i;i+=1;
    assert(pgfault_num==7);
    cprintf("write Virt Page 4 in lru_check_swap\n");
    *(unsigned char *)0x4000 =  i;i+=1;
    assert(pgfault_num==8);
    cprintf("write Virt Page 4 in lru_check_swap\n");
    *(unsigned char *)0x4000 =  i;i+=1;
    assert(pgfault_num==8);
    cprintf("write Virt Page 5 in lru_check_swap\n");
    *(unsigned char *)0x5000 =  i;i+=1;
    assert(pgfault_num==8);
    cprintf("write Virt Page 2 in lru_check_swap\n");
    *(unsigned char *)0x2000 =  i;i+=1;
```

```
    assert(pgfault_num==9);
    cprintf("write Virt Page 3 in lru_check_swap\n");
    *(unsigned char *)0x3000 =  i;i+=1;
    assert(pgfault_num==10);
    cprintf("LRU check succeed!\n");

    return 0;
}
```

并修改`_lru_swap_out_victim()`函数如下：

```
#define True 1
#define False 0
list_entry_t* curr_ptr;
static int
_lru_swap_out_victim(struct mm_struct *mm, struct Page ** ptr_page, int in_tick)
{
    if(mm == NULL || ptr_page == NULL){
        assert(False);
    }
    if(in_tick != 0){
        assert(False);
    }
    list_entry_t* head=(list_entry_t*) mm->sm_priv;
    list_entry_t* current = list_prev(head);
    list_entry_t* vic = NULL;
    struct Page* result = NULL;
    int minimum = 2147483647;
    //寻找pra_vaddr最小的页面，即最早被访问的页面
    while(current != head){
        struct Page* tmp_page = le2page(current, pra_page_link);
        int temp = *(unsigned char*) tmp_page->pra_vaddr;
        if(temp < minimum){
            minimum = temp;
            result = tmp_page;
            vic = current;
        }
        current = list_prev(current);
    }
    if(vic != NULL){
        *ptr_page = le2page(vic, pra_page_link);
        list_del(&result->pra_page_link);
    }else {
        *ptr_page = NULL;
    }
    cprintf("Here!\n");
    return 0;
}
```

测试样例

如下：

		命中	命中	命中	命中	缺失	缺失	缺失	缺失	命中	命中	缺失	缺失
	访问序列:	P_3	P_1	P_4	P_2	P_5	P_3	P_1	P_4	P_4	P_5	P_2	P_3
初始:	P_1	P_1	P_2	P_2	P_3	P_1	P_4	P_2	P_5	P_5	P_3	P_1	P_4
	P_2	P_2	P_4	P_3	P_1	P_4	P_2	P_5	P_3	P_3	P_1	P_4	P_5
	P_3	P_4	P_3	P_1	P_4	P_2	P_5	P_3	P_1	P_1	P_4	P_5	P_2
	P_4	P_3	P_1	P_4	P_2	P_5	P_3	P_1	P_4	P_4	P_5	P_2	P_3

结果:

```
---------LRU check begin----------
write Virt Page 3 in lru_check_swap
write Virt Page 1 in lru_check_swap
write Virt Page 4 in lru_check_swap
write Virt Page 2 in lru_check_swap
write Virt Page 5 in lru_check_swap
Store/AMO page fault
page fault at 0x00005000: K/W
Here!
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
write Virt Page 3 in lru_check_swap
Store/AMO page fault
page fault at 0x00003000: K/W
Here!
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page 1 in lru_check_swap
Store/AMO page fault
page fault at 0x00001000: K/W
Here!
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
```

```
write Virt Page 4 in lru_check_swap
Store/AMO page fault
page fault at 0x00004000: K/W
Here!
swap_out: i 0, store page in vaddr 0x2000 to disk swap entry 3
swap_in: load disk swap entry 5 with swap_page in vadr 0x4000
write Virt Page 4 in lru_check_swap
write Virt Page 5 in lru_check_swap
write Virt Page 2 in lru_check_swap
Store/AMO page fault
page fault at 0x00002000: K/W
Here!
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
swap_in: load disk swap entry 3 with swap_page in vadr 0x2000
write Virt Page 3 in lru_check_swap
Store/AMO page fault
page fault at 0x00003000: K/W
Here!
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
LRU check succeed!
count is 1, total is 8
check_swap() succeeded!
++ setup timer interrupts
```

方案二

设计思想和分析

我们知道，每次访问一个页，产生缺页异常时，会调用`_lru_map_swappable()`函数，这个函数维护一个链表，并以页面的第一次访问时间为顺序排列。所以，实现`LRU`，在缺页异常时，并不需要我们做什么，问题在于，访问一个已经在内存的物理页时，该如何更新这个链表（当没有缺页异常时，OS不会调用`_lru_map_swappable()`函数）。

`*(unsigned char *)0x5000 = 0x0e;`我们了解到，这段代码是向一个虚拟地址写入一个值，若没有产生缺页异常，这由处理器直接完成，OS无法介入。所以，我们想手动调用`_lru_map_swappable()`函数，来更新链表。

所以，我们写了下面这个函数（在测试函数中，每次向地址写入值时，调用这个函数）：

```c
static int
_lru_update(struct mm_struct *mm, uintptr_t addr){
    pte_t *ptep = NULL;
    //获取管理该地址的页表项
    ptep = get_pte(mm->pgdir, addr, 1);
    if(ptep == NULL){

    }
    else{
        //根据该页表项获取页，再调用 _lru_map_swappable（）函数更新链表
        _lru_map_swappable(mm, addr, pte2page(*ptep), 0);
    }
    return 0;
}
```

我们又更改了 `_lru_map_swappable()` 函数：

```c
static int
_lru_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int
swap_in)
{

    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    list_entry_t *entry=&(page->pra_page_link);
    assert(entry != NULL && head != NULL);
    curr_ptr = head->next;
    //检查该页是否已经在链表中
    while(curr_ptr!=head){
       if(curr_ptr==entry){
          //若在则删除它，后续把它再加入到链表头的后面
          list_del(entry);
          break;
       }
       curr_ptr=curr_ptr->next;
    }
    //将其加入到链表头的后面
    list_add(head, entry);
    return 0;
}
```

由于`_lru_swap_out_victim()`函数取出的是链表最后一个，即最早访问的页面，所以无需更改。

测试样例

如下：

		命中	命中	命中	命中	缺失	缺失	缺失	命中	命中	缺失	缺失	缺失
	访问序列：	P_3	P_1	P_4	P_2	P_5	P_3	P_1	P_2	P_3	P_4	P_5	P_1
初始：	P_1	P_1	P_2	P_2	P_3	P_1	P_4	P_2	P_5	P_5	P_1	P_2	P_3
	P_2	P_2	P_4	P_3	P_1	P_4	P_2	P_5	P_3	P_1	P_2	P_3	P_4
	P_3	P_4	P_3	P_1	P_4	P_2	P_5	P_3	P_1	P_2	P_3	P_4	P_3
	P_4	P_3	P_1	P_4	P_2	P_5	P_3	P_1	P_2	P_3	P_4	P_5	P_1

结果：

```
---------LRU check begin----------
write Virt Page 3 in lru_check_swap
write Virt Page 1 in lru_check_swap
write Virt Page 4 in lru_check_swap
write Virt Page 2 in lru_check_swap
write Virt Page 5 in lru_check_swap
Store/AMO page fault
page fault at 0x00005000: K/W
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
write Virt Page 3 in lru_check_swap
Store/AMO page fault
page fault at 0x00003000: K/W
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page 1 in lru_check_swap
Store/AMO page fault
page fault at 0x00001000: K/W
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
```

```
write Virt Page b in fifo_check_swap
write Virt Page c in fifo_check_swap
write Virt Page d in fifo_check_swap
Store/AMO page fault
page fault at 0x00004000: K/W
swap_out: i 0, store page in vaddr 0x5000 to disk swap entry 6
swap_in: load disk swap entry 5 with swap_page in vadr 0x4000
write Virt Page e in fifo_check_swap
Store/AMO page fault
page fault at 0x00005000: K/W
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
swap_in: load disk swap entry 6 with swap_page in vadr 0x5000
write Virt Page a in fifo_check_swap
Store/AMO page fault
page fault at 0x00001000: K/W
swap_out: i 0, store page in vaddr 0x2000 to disk swap entry 3
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
LRU check succeed!
count is 1, total is 8
check_swap() succeeded!
++ setup timer interrupts
100 ticks
```