练习1：给未被映射的地址映射上物理页（需要编程）

完成do_pgfault（mm/vmm.c）函数，给未被映射的地址映射上物理页。设置访问权限 的时候需要参考页面所在 VMA 的权限，同时需要注意映射物理页时需要操作内存控制 结构所指定的页表，而不是内核的页表。

补全代码如下

```
/* do_pgfault - interrupt handler to process the page fault execption
 * @mm         : the control struct for a set of vma using the same PDT
 * @error_code : the error code recorded in trapframe->tf_err which is setted by
x86 hardware
 * @addr       : the addr which causes a memory access exception, (the contents
of the CR2 register)
 *
 * CALL GRAPH: trap--> trap_dispatch-->pgfault_handler-->do_pgfault
 * The processor provides ucore's do_pgfault function with two items of
information to aid in diagnosing
 * the exception and recovering from it.
 *   (1) The contents of the CR2 register. The processor loads the CR2 register
with the
 *       32-bit linear address that generated the exception. The do_pgfault fun
can
 *       use this address to locate the corresponding page directory and page-
table
 *       entries.
 *   (2) An error code on the kernel stack. The error code for a page fault has a
format different from
 *       that for other exceptions. The error code tells the exception handler
three things:
 *         -- The P flag   (bit 0) indicates whether the exception was due to a
not-present page (0)
 *            or to either an access rights violation or the use of a reserved
bit (1).
 *         -- The W/R flag (bit 1) indicates whether the memory access that
caused the exception
 *            was a read (0) or write (1).
 *         -- The U/S flag (bit 2) indicates whether the processor was executing
at user mode (1)
 *            or supervisor mode (0) at the time of the exception.
 */
int
do_pgfault(struct mm_struct *mm, uint32_t error_code, uintptr_t addr) {
    int ret = -E_INVAL;
    //try to find a vma which include addr
    struct vma_struct *vma = find_vma(mm, addr);

    pgfault_num++;
    //If the addr is in the range of a mm's vma?
    if (vma == NULL || vma->vm_start > addr) {
        cprintf("not valid addr %x, and  can not find it in vma\n", addr);
        goto failed;
```

```
    }
    //check the error_code
    switch (error_code & 3) {
    default:
            /* error code flag : default is 3 ( W/R=1, P=1): write, present */
    case 2: /* error code flag : (W/R=1, P=0): write, not present */
        if (!(vma->vm_flags & VM_WRITE)) {
            cprintf("do_pgfault failed: error code flag = write AND not present,
but the addr's vma cannot write\n");
            goto failed;
        }
        break;
    case 1: /* error code flag : (W/R=0, P=1): read, present */
        cprintf("do_pgfault failed: error code flag = read AND present\n");
        goto failed;
    case 0: /* error code flag : (W/R=0, P=0): read, not present */
        if (!(vma->vm_flags & (VM_READ | VM_EXEC))) {
            cprintf("do_pgfault failed: error code flag = read AND not present,
but the addr's vma cannot read or exec\n");
            goto failed;
        }
    }
    /* IF (write an existed addr ) OR
     *    (write an non_existed addr && addr is writable) OR
     *    (read  an non_existed addr && addr is readable)
     * THEN
     *    continue process
     */
    uint32_t perm = PTE_U;
    if (vma->vm_flags & VM_WRITE) {
        perm |= PTE_W;
    }
    addr = ROUNDDOWN(addr, PGSIZE);

    ret = -E_NO_MEM;

    pte_t *ptep=NULL;
    /*LAB3 EXERCISE 1: YOUR CODE
    * Maybe you want help comment, BELOW comments can help you finish the code
    *
    * Some Useful MACROs and DEFINEs, you can use them in below implementation.
    * MACROs or Functions:
    *   get_pte : get an pte and return the kernel virtual address of this pte
for la
    *             if the PT contians this pte didn't exist, alloc a page for PT
(notice the 3th parameter '1')
    *   pgdir_alloc_page : call alloc_page & page_insert functions to allocate a
page size memory & setup
    *             an addr map pa<--->la with linear address la and the PDT pgdir
    * DEFINES:
    *   VM_WRITE  : If vma->vm_flags & VM_WRITE == 1/0, then the vma is
writable/non writable
    *   PTE_W           0x002                   // page table/directory entry
flags bit : Writeable
    *   PTE_U           0x004                   // page table/directory entry
flags bit : User can access
    * VARIABLES:
    *   mm->pgdir : the PDT of these vma
```

```
    *
    */
#if 0
    /*LAB3 EXERCISE 1: YOUR CODE*/
    ptep = ???              //(1) try to find a pte, if pte's PT(Page Table)
isn't existed, then create a PT.
    if (*ptep == 0) {
                            //(2) if the phy addr isn't exist, then alloc a page
& map the phy addr with logical addr

    }
    else {
    /*LAB3 EXERCISE 2: YOUR CODE
    * Now we think this pte is a  swap entry, we should load data from disk to a
page with phy addr,
    * and map the phy addr with logical addr, trigger swap manager to record the
access situation of this page.
    *
    *  Some Useful MACROs and DEFINEs, you can use them in below implementation.
    *  MACROs or Functions:
    *    swap_in(mm, addr, &page) : alloc a memory page, then according to the
swap entry in PTE for addr,
    *                               find the addr of disk page, read the content
of disk page into this memroy page
    *    page_insert  : build the map of phy addr of an Page with the linear addr
la
    *    swap_map_swappable  : set the page swappable
    */
        if(swap_init_ok) {
            struct Page *page=NULL;
                                    //(1）According to the mm AND addr, try to
load the content of right disk page
                                    //    into the memory which page managed.
                                    //(2) According to the mm, addr AND page,
setup the map of phy addr <---> logical addr
                                    //(3) make the page swappable.
        }
        else {
            cprintf("no swap_init_ok but ptep is %x, failed\n",*ptep);
            goto failed;
        }
   }
#endif
    // try to find a pte, if pte's PT(Page Table) isn't existed, then create a
PT.
    // (notice the 3th parameter '1')
    ptep = get_pte(mm->pgdir, addr, 1);
    if (ptep == NULL) {
        cprintf("get_pte in do_pgfault failed\n");
        goto failed;
    }

    if (*ptep == 0) {
        // if the phy addr isn't exist, then alloc a page & map the phy addr with
logical addr
        if (pgdir_alloc_page(mm->pgdir, addr, perm) == NULL) {
            cprintf("pgdir_alloc_page in do_pgfault failed\n");
            goto failed;
```

```
        }
    }
    else { // if this pte is a swap entry, then load data from disk to a page
with phy addr
           // and call page_insert to map the phy addr with logical addr
        if(swap_init_ok) {
            struct Page *page=NULL;
            ret = swap_in(mm, addr, &page);
            if (ret != 0) {
                cprintf("swap_in in do_pgfault failed\n");
                goto failed;
            }
            page_insert(mm->pgdir, page, addr, perm);
            swap_map_swappable(mm, addr, page, 1);
            page->pra_vaddr = addr;
        }
        else {
            cprintf("no swap_init_ok but ptep is %x, failed\n",*ptep);
            goto failed;
        }
   }
   ret = 0;
failed:
    return ret;
}
```

请描述页目录项（Pag Director Entry）和页表（Page Table Entry）中组成部分对ucore实现页替换算法的潜在用处。

每个页表项（PTE）都由一个32位整数来存储数据，其结构如下

```
COPY      31-12      9-11     8    7    6   5   4      3    2   1   0
+--------------+-------+-----+----+---+---+-----+-----+---+---+---+
|     Offset    | Avail | MBZ | PS | D | A | PCD | PWT | U | W | P |
+--------------+-------+-----+----+---+---+-----+-----+---+---+---+
```

- 0 - Present: 表示当前PTE所指向的物理页面是否驻留在内存中
- 1 - Writeable: 表示是否允许读写
- 2 - User: 表示该页的访问所需要的特权级。即User(ring 3)是否允许访问
- 3 - PageWriteThough: 表示是否使用write through缓存写策略
- 4 - PageCacheDisable: 表示是否**不对**该页进行缓存
- 5 - Access: 表示该页是否已被访问过
- 6 - Dirty: 表示该页是否已被修改
- 7 - PageSize: 表示该页的大小
- 8 - MustBeZero: 该位必须保留为0
- 9-11 - Available: 第9-11这三位并没有被内核或中断所使用，可保留给OS使用。
- 12-31 - Offset: 目标地址的后20位。

如果ucore的缺页服务例程在执行过程中访问内存，出现了页访问异常，请问硬件要做哪些事情？

- 将发生错误的线性地址（虚拟地址）保存至CR2寄存器中。

- 压入`EFLAGS`，`CS`，`EIP`，错误码和中断号至当前内核栈中。
- 保存上下文。
- 执行新的缺页中断程序。
- 恢复上下文。
- 继续执行上一级的缺页服务例程。

练习2：补充完成基于FIFO的页面替换算法（需要编程）

完成vmm.c中的do_pgfault函数，并且在实现FIFO算法的swap_fifo.c中完成map_swappable和swap_out_vistim函数。通过对swap的测试。注意：在LAB2 EXERCISE 2处填写代码。

编写代码如下

```
/*
 * (3)_fifo_map_swappable: According FIFO PRA, we should link the most recent arrival page at the back of pra_list_head qeueue
 */
static int
_fifo_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);
    //record the page access situlation
    /*LAB3 EXERCISE 2: YOUR CODE*/
    //(1)link the most recent arrival page at the back of the pra_list_head qeueue.
    list_add(head, entry);
    return 0;
}
/*
 *  (4)_fifo_swap_out_victim: According FIFO PRA, we should unlink the  earliest arrival page in front of pra_list_head qeueue,
 *                              then assign the value of *ptr_page to the addr of this page.
 */
static int
_fifo_swap_out_victim(struct mm_struct *mm, struct Page ** ptr_page, int in_tick)
{
     list_entry_t *head=(list_entry_t*) mm->sm_priv;
         assert(head != NULL);
     assert(in_tick==0);
     /* Select the victim */
     /*LAB3 EXERCISE 2: YOUR CODE*/
     //(1)  unlink the  earliest arrival page in front of pra_list_head qeueue
     //(2)  assign the value of *ptr_page to the addr of this page
     list_entry_t *le = head->prev;
     assert(head!=le);
     struct Page *p = le2page(le, pra_page_link);
     list_del(le);
     assert(p != NULL);
     *ptr_page = p;
```

```
    return 0;
}
```

如果要在ucore上实现"extended clock页替换算法"请给你的设计方案，现有的swap_manager框架是否足以支持在ucore中实现此算法？如果是，请给你的设计方案。如果不是，请给出你的新的扩展和基此扩展的设计方案。并需要回答如下问题

- 需要被换出的页的特征是什么？

PTE_P(Present)和PTE_D(Dirty)位均为0。

- 在ucore中如何判断具有这样特征的页？

通过位运算来判断

- 何时进行换入和换出操作？

缺页时

结果测试：

```
-> % make qemu
WARNING: Image format was not specified for 'bin/ucore.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
WARNING: Image format was not specified for 'bin/swap.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
(THU.CST) os is loading ...

Special kernel symbols:
  entry  0xc0100036 (phys)
  etext  0xc0109253 (phys)
  edata  0xc0127000 (phys)
  end    0xc0128114 (phys)
Kernel executable memory footprint: 161KB
memory management: default_pmm_manager
e820map:
  memory: 0009fc00, [00000000, 0009fbff], type = 1.
  memory: 00000400, [0009fc00, 0009ffff], type = 2.
  memory: 00010000, [000f0000, 000fffff], type = 2.
  memory: 07ee0000, [00100000, 07fdffff], type = 1.
  memory: 00020000, [07fe0000, 07ffffff], type = 2.
  memory: 00040000, [fffc0000, ffffffff], type = 2.
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
```

```
-------------------- BEGIN --------------------
PDE(0e0) c0000000-f8000000 38000000 urw
  |-- PTE(38000) c0000000-f8000000 38000000 -rw
PDE(001) fac00000-fb000000 00400000 -rw
  |-- PTE(000e0) faf00000-fafe0000 000e0000 urw
  |-- PTE(00001) fafeb000-fafec000 00001000 -rw
--------------------- END ---------------------
check_vma_struct() succeeded!
page fault at 0x00000100: K/W [no page found].
check_pgfault() succeeded!
check_vmm() succeeded.
ide 0:      10000(sectors), 'QEMU HARDDISK'.
ide 1:     262144(sectors), 'QEMU HARDDISK'.
SWAP: manager = fifo swap manager
BEGIN check_swap: count 1, total 31960
setup Page Table for vaddr 0X1000, so alloc a page
setup Page Table vaddr 0~4MB OVER!
set up init env for check_swap begin!
page fault at 0x00001000: K/W [no page found].
page fault at 0x00002000: K/W [no page found].
page fault at 0x00003000: K/W [no page found].
page fault at 0x00004000: K/W [no page found].
set up init env for check_swap over!
write Virt Page c in fifo_check_swap
write Virt Page a in fifo_check_swap
write Virt Page d in fifo_check_swap
write Virt Page b in fifo_check_swap
write Virt Page e in fifo_check_swap
page fault at 0x00005000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
write Virt Page b in fifo_check_swap
write Virt Page a in fifo_check_swap
page fault at 0x00001000: K/W [no page found].
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
write Virt Page b in fifo_check_swap
page fault at 0x00002000: K/W [no page found].
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 c in fifo_check_swap
page fault at 0x00003000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page d in fifo_check_swap
page fault at 0x00004000: K/W [no page found].
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
page fault at 0x00005000: K/W [no page found].
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
page fault at 0x00001000: K/R [no page found].
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
count is 0, total is 7
check_swap() succeeded!
++ setup timer interrupts
```

```
100 ticks
End of Test.
kernel panic at kern/trap/trap.c:20:
    EOT: kernel seems ok.
stack trackback:
Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K>
-> % make grade
Check SWAP:              (s)
  -check pmm:                        OK
  -check page table:                 OK
  -check vmm:                        OK
  -check swap page fault:            OK
  -check ticks:                      OK
Total Score: 45/45
```

可知结果正确 实验成功

扩展练习 Challenge：实现识别dirty bit的extended clock页替换算法（需要编程）

查找资料了解clock页替换算法：

时钟页替换算法把各个页面组织成环形链表的形式，类似于一个钟的表面。然后把一个指针（简称当前指针）指向最老的那个页面，即最先进来的那个页面。另外，时钟算法需要在页表项（PTE）中设置了一位访问位来表示此页表项对应的页当前是否被访问过。当该页被访问时，CPU中的MMU硬件将把访问位置"1"。当操作系统需要淘汰页时，对当前指针指向的页所对应的页表项进行查询，如果访问位为"0"，则淘汰该页，如果该页被写过，则还要把它换出到硬盘上；如果访问位为"1"，则将该页表项的此位置"0"，继续访问下一个页。

首先创建两个新文件 swap_clock.h swap_clock.c

仿照swap_fifo 编写函数 _clock_init_mm 完全相同 都是循环链表的初始化

```
static int
_clock_init_mm(struct mm_struct *mm)
{
     list_init(&pra_list_head);
     mm->sm_priv = &pra_list_head;
     return 0;
}
```

编写函数 _clock_map_swappable

这里和fifo有一点点 不同 新插入的页需要将dirty位置0

由于是通过dirty位来进行置换 所以插入顺序没有影响 这里直接插入了链表的最后一个位置

```
static int
_clock_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);

    struct Page *ptr = le2page(entry, pra_page_link);
    pte_t *pte = get_pte(mm -> pgdir, ptr -> pra_vaddr, 0);
    *pte &= ~PTE_D;// 将dirty位 置0
    list_add(head -> prev, entry);// 插入链表最后
    return 0;
}
```

编写函数 _clock_swap_out_victim

这里实现了clock页置换的换出算法

遍历整个循环链表

如果dirty位为1 则置0

如果dirty位为0 则换出 并退出遍历

```
static int
_clock_swap_out_victim(struct mm_struct *mm, struct Page ** ptr_page, int
in_tick)
{
     list_entry_t *head=(list_entry_t*) mm->sm_priv;
     assert(head != NULL);
     assert(in_tick==0);

     list_entry_t *p = head;
     while (1) {
         // 遍历循环链表
         p = list_next(p);
         if (p == head) {
             p = list_next(p);
         }
         struct Page *ptr = le2page(p, pra_page_link);
         pte_t *pte = get_pte(mm -> pgdir, ptr -> pra_vaddr, 0);
         //获取页表项
         if ((*pte & PTE_D) == 1) {// 如果dirty bit为1，改为0
             *pte &= ~PTE_D;
         }
         else
         {// 如果dirty bit为0，则标记为换出页
             *ptr_page = ptr;
             list_del(p);
             break;
         }
     }
     return 0;
}
```

仿照 swap_fifo.c 的检查函数

```
static int
_clock_check_swap(void) {
    cprintf("write Virt Page c in clock_check_swap\n");
    *(unsigned char *)0x3000 = 0x0c;
    assert(pgfault_num==4);
    cprintf("write Virt Page a in clock_check_swap\n");
    *(unsigned char *)0x1000 = 0x0a;
    assert(pgfault_num==4);
    cprintf("write Virt Page d in clock_check_swap\n");
    *(unsigned char *)0x4000 = 0x0d;
    assert(pgfault_num==4);
    cprintf("write Virt Page b in clock_check_swap\n");
    *(unsigned char *)0x2000 = 0x0b;
    assert(pgfault_num==4);
    cprintf("write Virt Page e in clock_check_swap\n");
    *(unsigned char *)0x5000 = 0x0e;
    assert(pgfault_num==5);
    cprintf("write Virt Page b in clock_check_swap\n");
    *(unsigned char *)0x2000 = 0x0b;
    assert(pgfault_num==5);
    cprintf("write Virt Page a in clock_check_swap\n");
    *(unsigned char *)0x1000 = 0x0a;
    assert(pgfault_num==6);
    cprintf("write Virt Page b in clock_check_swap\n");
    *(unsigned char *)0x2000 = 0x0b;
    assert(pgfault_num==7);
    cprintf("write Virt Page c in clock_check_swap\n");
    *(unsigned char *)0x3000 = 0x0c;
    assert(pgfault_num==8);
    cprintf("write Virt Page d in clock_check_swap\n");
    *(unsigned char *)0x4000 = 0x0d;
    assert(pgfault_num==9);
    cprintf("write Virt Page e in clock_check_swap\n");
    *(unsigned char *)0x5000 = 0x0e;
    assert(pgfault_num==10);
    cprintf("write Virt Page a in clock_check_swap\n");
    assert(*(unsigned char *)0x1000 == 0x0a);
    *(unsigned char *)0x1000 = 0x0a;
    assert(pgfault_num==11);
    return 0;
}
```

编写其他函数和结构体

```
static int
_clock_init(void)
{
    return 0;
}

static int
_clock_set_unswappable(struct mm_struct *mm, uintptr_t addr)
{
    return 0;
```

```
}

static int
_clock_tick_event(struct mm_struct *mm)
{
    return 0;
}
struct swap_manager swap_manager_clock =
{
     .name            = "extend_clock swap manager",
     .init            = &_clock_init,
     .init_mm         = &_clock_init_mm,
     .tick_event      = &_clock_tick_event,
     .map_swappable   = &_clock_map_swappable,
     .set_unswappable = &_clock_set_unswappable,
     .swap_out_victim = &_clock_swap_out_victim,
     .check_swap      = &_clock_check_swap,
};
```

修改swap.c 将sm变为 swap_manager_clock

测试结果如下

```
-> % make qemu
+ cc kern/mm/swap_fifo.c
+ ld bin/kernel
+ ld bin/bootblock
'obj/bootblock.out' size: 442 bytes
build 512 bytes boot sector: 'bin/bootblock' success!
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0261785 s, 196 MB/s
1+0 records in
1+0 records out
512 bytes copied, 2.7445e-05 s, 18.7 MB/s
349+1 records in
349+1 records out
178700 bytes (179 kB, 175 KiB) copied, 0.000659711 s, 271 MB/s
WARNING: Image format was not specified for 'bin/ucore.img' and probing guessed
raw.
         Automatically detecting the format is dangerous for raw images, write
operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
WARNING: Image format was not specified for 'bin/swap.img' and probing guessed
raw.
         Automatically detecting the format is dangerous for raw images, write
operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
(THU.CST) os is loading ...

Special kernel symbols:
  entry  0xc0100036 (phys)
  etext  0xc0109253 (phys)
  edata  0xc0127000 (phys)
```

```
  end    0xc0128114 (phys)
Kernel executable memory footprint: 161KB
memory management: default_pmm_manager
e820map:
  memory: 0009fc00, [00000000, 0009fbff], type = 1.
  memory: 00000400, [0009fc00, 0009ffff], type = 2.
  memory: 00010000, [000f0000, 000fffff], type = 2.
  memory: 07ee0000, [00100000, 07fdffff], type = 1.
  memory: 00020000, [07fe0000, 07ffffff], type = 2.
  memory: 00040000, [fffc0000, ffffffff], type = 2.
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
-------------------- BEGIN --------------------
PDE(0e0) c0000000-f8000000 38000000 urw
  |-- PTE(38000) c0000000-f8000000 38000000 -rw
PDE(001) fac00000-fb000000 00400000 -rw
  |-- PTE(000e0) faf00000-fafe0000 000e0000 urw
  |-- PTE(00001) fafeb000-fafec000 00001000 -rw
--------------------- END ---------------------
check_vma_struct() succeeded!
page fault at 0x00000100: K/W [no page found].
check_pgfault() succeeded!
check_vmm() succeeded.
ide 0:      10000(sectors), 'QEMU HARDDISK'.
ide 1:     262144(sectors), 'QEMU HARDDISK'.
SWAP: manager = extend_clock swap manager
BEGIN check_swap: count 1, total 31960
setup Page Table for vaddr 0X1000, so alloc a page
setup Page Table vaddr 0~4MB OVER!
set up init env for check_swap begin!
page fault at 0x00001000: K/W [no page found].
page fault at 0x00002000: K/W [no page found].
page fault at 0x00003000: K/W [no page found].
page fault at 0x00004000: K/W [no page found].
set up init env for check_swap over!
write Virt Page c in clock_check_swap
write Virt Page a in clock_check_swap
write Virt Page d in clock_check_swap
write Virt Page b in clock_check_swap
write Virt Page e in clock_check_swap
page fault at 0x00005000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
write Virt Page b in clock_check_swap
write Virt Page a in clock_check_swap
page fault at 0x00001000: K/W [no page found].
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
write Virt Page b in clock_check_swap
page fault at 0x00002000: K/W [no page found].
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 c in clock_check_swap
page fault at 0x00003000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page d in clock_check_swap
page fault at 0x00004000: K/W [no page found].
```

```
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 clock_check_swap
page fault at 0x00005000: K/W [no page found].
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 clock_check_swap
page fault at 0x00001000: K/R [no page found].
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
count is 0, total is 7
check_swap() succeeded!
++ setup timer interrupts
100 ticks
End of Test.
kernel panic at kern/trap/trap.c:20:
    EOT: kernel seems ok.
stack trackback:
Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K>
```

可知结果正确 实验成功