

实验三 虚拟内存管理

计31 张正 2013011418

一、实验目的

- 了解虚拟内存的Page Fault异常处理实现
- 了解页替换算法在操作系统中的实现

二、实验内容

本次实验是在实验二的基础上，借助于页表机制和实验一中涉及的中断异常处理机制，完成Page Fault异常处理和FIFO页替换算法的实现，结合磁盘提供的缓存空间，从而能够支持虚存管理，提供一个比实际物理内存空间“更大”的虚拟内存空间给系统使用。

练习0：填写已有实验

本实验依赖实验1/2。请把你做的实验1/2的代码填入本实验中代码中有“LAB1”,“LAB2”的注释相应部分。

make之后

File Edit View Search Terminal Help

page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 page fault at 0x00000100: K/W [no page found].
 p[~/moococ/ucore_lab/labcodes/lab3]
 moococ->

[no page found]说明正在读取不存在的页

练习1 给未被映射的地址映射上物理页

完成do_pgfault (mm/vmm.c) 函数，给未被映射的地址映射上物理页。

修改LAB3 EXERCISE 1

```
/*LAB3 EXERCISE 1: 2013011418*/
if ((ptep = get_pte(mm->pgdir, addr, 1)) == NULL) {
    cprintf("get_pte in do_pgfault failed\n");
    goto failed;
}          //(1) try to find a pte, if pte's PT(Page Table) isn't existed, then
create a PT.
if (*ptep == 0) {
if (pgdir_alloc_page(mm->pgdir, addr, perm) == NULL) {
    cprintf("pgdir_alloc_page in do_pgfault failed\n");
    goto failed;
}
}
```

练习2：补充完成基于FIFO的页面替换算法

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_victim函数。

在LAB3 EXERCISE 2出增加代码

```
if(swap_init_ok) {
    struct Page *page=NULL;
    if ((ret = swap_in(mm, addr, &page)) != 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;
}
}
```

并且完成了

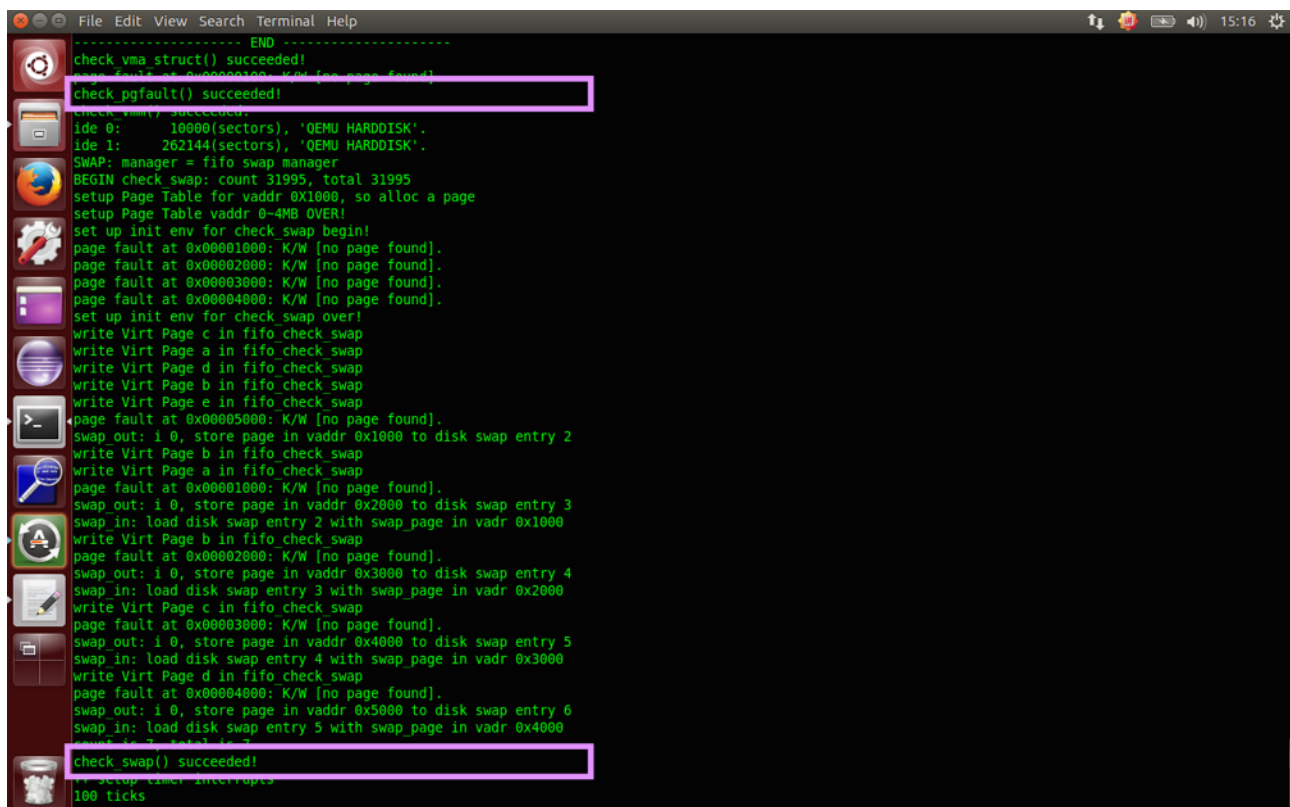
```
static int _fifo_map_swappable(struct mm_struct *mm, uintptr_t addr, struct
Page *page, int swap_in)
{
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    list_entry_t *entry=&(page->pra_page_link);
```

```

    assert(entry != NULL && head != NULL);
    //record the page access situation
    /*LAB3 EXERCISE 2: YOUR CODE*/
    //(1)link the most recent arrival page at the back of the pra_list_head
queueue.
    list_add(head, entry);
    return 0;
}
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 queueue
    //(2) set the addr of addr of this page to ptr_page
    /* Select the tail */
    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;
}

```

完成后make得如下结果



```
File Edit View Search Terminal Help
----- END -----
check vma struct() succeeded!
page fault at 0x00001000: K/W [no page found]
check pgfault() succeeded!
check_swap() succeeded!
ide 0: 10000(sectors), 'OEMU_HARDDISK'.
ide 1: 262144(sectors), 'OEMU_HARDDISK'.
SWAP: manager = fifo swap manager
BEGIN check swap: count 31995, total 31995
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
check_swap() succeeded!
100 ticks
```

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？

换出的页的特征是这些也访问次数少，并且有映射到用户空间且被用户程序直接访问的页面才能被交换，而被内核直接使用的内核空间的页面不能被换出

○在ucore中如何判断具有这样特征的页？

通过swap_map_swappable函数来查询这些页的访问情况并间接调用相关函数，换出“不常用”的页到磁盘上。

○何时进行换入和换出操作？

当ucore或应用程序访问地址所在的页不在内存时，就会产生page fault异常，引起调用do_pgfault函数，此函数会判断产生访问异常的地址属于check_mm_struct某个vma表示的合法虚拟地址空间，且保存在硬盘swap文件中（即对应的PTE的高24位不为0，而最低位为0），则是执行页换入的时机，将调用swap_in函数完成页面换入。

ucore目前大致有两种策略，即积极换出策略和消极换出策略。积极换出策略是指操作系统周期性地（或在系统不忙的时候）主动把某些认为“不常用”的页换出到硬盘上，从而确保系统中总有一定数量的空闲页存在，这样当需要空闲页时，基本上能够及时满足需求；消极换出策略是指，只是当试图得到空闲页时，发现当前没有空闲的物理页可供分配，这时才开始查找“不常用”页面，并把一个或多个这样的页换出到硬盘上。