# 实验项目：虚拟内存管理

姓名：张伟焜　学号：17343155　邮箱：zhangwk8@mail2.sysu.edu.cn
院系：数据科学与计算机学院 专业：17 级软件工程 指导教师：张永东

**【实验题目】**

虚拟内存管理

**【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

**【实验要求】**

根据指导，完成练习 0~2。

**【实验方案】**

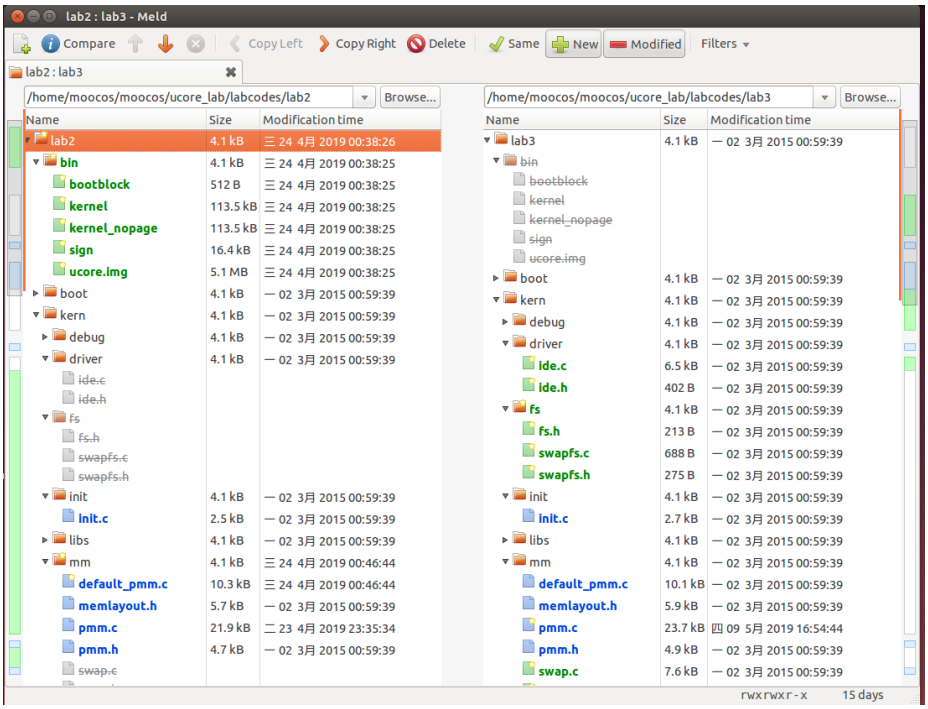
实验环境：老师提供的虚拟机（Virtual box），无特殊硬件要求

实验思路：根据实验指导，先了解理论知识，再进行实验

**【实验过程】**

**练习 0：填写已有实验。**

使用 meld 软件将 ucore 启动实验的代码导入。




注意要点击标星文件进行对比，将上次实验完成的函数复制过来，不要将整个文件进行覆盖。之前实验修改的内容主要在 kdebug.c trap.c pmm.c default_pmm.c。

**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。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。注意：在 LAB3 EXERCISE 1 处填写代码。执行 make qemu 后，如果通过 check_pgfault 函数的测试后，会有“check_pgfault() succeeded!”的输出，表示练习 1 基本正确。*

**分析：**ucore 的初始化过程。在调用 vmm_int 前先调用 pmm_init 实现物理内存的管理（lab2 中已经完成）；接着，执行中断、异常相关初始化函数，即 pic_int、idt_int 等（lab1）；

在 idt_int 函数之后，调用 vmm_int、ide_int、swap_int 函数。

在 lab 3 中 do_pgfault 函数会申请一个空闲物理页，并建立好映射关系，让“合法”的虚拟页与实际的物理页帧进行对应。

为了区分什么是“合法”的虚拟页，我们建立了 mm_struct 和 vma_struct 来描述 ucore 模拟应用程序运行所需的合法内存空间。

vma_struct 述应用程序对虚拟内存“需求”，以及针对 vma_struct 的函数操作。

```
// the virtual continuous memory area(vma), [vm_start, vm_end),
// addr belong to a vma means  vma.vm_start<= addr <vma.vm_end
struct vma_struct {
    struct mm_struct *vm_mm; // the set of vma using the same PDT
    uintptr_t vm_start;      // start addr of vma
    uintptr_t vm_end;        // end addr of vma, not include the vm_end itself
    uint32_t vm_flags;       // flags of vma
    list_entry_t list_link;  // linear list link which sorted by start addr of vma
};
```

vm_mm 是一个指针，指向一个比 vma_struct 更高的抽象层次的数据结构 mm_struct

```
// the control struct for a set of vma using the same PDT
struct mm_struct {
    list_entry_t mmap_list;        // linear list link which sorted by start addr of vma
    struct vma_struct *mmap_cache; // current accessed vma, used for speed purpose
    pde_t *pgdir;                  // the PDT of these vma
    int map_count;                 // the count of these vma
    void *sm_priv;                   // the private data for swap manager
};
```

页面异常（pgfault）的情况有

1.目标页面不存在（页表项全为 0，即该线性地址与物理地址尚未建立映射或者已经撤销）；

2.相应的物理页面不在内存中（页表项非空，但 Present 标志位=0，比如在 swap 分区或磁盘文件上）

3.访问权限不符合（此时页表项 P 标志=1，比如企图写只读页面）.

结合注释，我们给出以下代码：

```
if ((ptep = get_pte(mm->pgdir, addr, 1)) == NULL) {  //get_pte失败,非法访问
    cprintf("get_pte in do_pgfault failed\n");
    goto failed;
}
if (*ptep == 0) {
    if (pgdir_alloc_page(mm->pgdir, addr, perm) == NULL) {  //pgdir_alloc_page失败,内存不足
        cprintf("pgdir_alloc_page in do_pgfault failed\n");
        goto failed;
    }
   }
```

运行结果：

```
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'.
```

观察到 check_pgfault() succeeded! 实验结果正确。

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

（1）页目录项：使用双向链表记录在物理内存中的所有页的物理地址和逻辑地址的映射关系。实现页的替换算法时，被换出的页从 pgdir（页目录项）中选出。

（2）页表（Page Table Entry）页表项中的 dirty bit 和访问位可以帮助实现一些页替换算法。此外，页表存储了替换算法中被换入的页的信息，替换后会将其映射到物理地址。

**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？**

CPU 将产生异常的地址保存在 CR2 寄存器中，同时还要将 EFLAGS，CS，EIP，ErrorCode 等保存在内核栈上。将 ErrorCode 设为 0xE（页访问异常编码）。将页访问异常的中断服务例程的地址加载到 CS 和 EIP 中，并开始执行中断服务程序。

**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。通过对 swap 的测试。注意：在 LAB3 EXERCISE 2 处填写代码。执行 make qemu 后，如果通过 check_swap 函数的测试后，会有“check_swap() succeeded!”的输出，表示练习 2 基本正确。*

**分析：**

当应用程序访问数据或代码时，如果这些数据或代码不在内存中，则会产生页访问异常。这时，操作系统必须处理这种页访问异常，应当尽快把应用程序当前需要的数据或代码放到内存中来，然后重新执行应用程序产生异常的访存指令。(换入)

如果在换入操作前，操作系统发现物理内存空间不够，这时它必须选择某页换出到磁盘上，以腾出空间给要换入的数据或代码。(换出)

在换入时，需要先检查产生访问异常的地址是否属于某个 vma 表示的合法虚拟地址，并且保存在硬盘的 swap 文件中（对应的 PTE 的高 24 位不为 0）。如果满足以上两点，则执行 swap_in()函数换入页面。

```
else {  //页表项非空，尝试换入页面
    if (swap_init_ok) {
        struct Page *page = NULL;
```

```
            //硬盘中的内容换入至page中
            if ((ret = swap_in(mm, addr, &page)) != 0) { //failed
                cprintf("swap_in in do_pgfault failed\n");
                goto failed;
            }
            page_insert(mm->pgdir, page, addr, perm); //建立虚拟地址和物理地址之间映射关系
            swap_map_swappable(mm, addr, page, 1); //设为可交换的
            page->pra_vaddr = addr;
        }
        else {  //失败
            cprintf("no swap_init_ok but ptep is %x, failed\n", *ptep);
            goto failed;
        }
    }
```

换出，当申请空闲页面时，alloc_pages()函数不能获得空闲页，则需要调用 swap_out()函数换出不常用的页面。

_fifo_map_swappable()的作用是将最近被用到的页面添加到算法所维护的次序队列。

```
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;
}
```

_fifo_swap_out_victim()函数主要作用是用来查询哪个页面需要被换出。

```
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)  set the addr of addr of this page to ptr_page
    list_entry_t *le = head->prev;  //用le指示需要被换出的页，最先进入队列的元素head的
另一端
```

```
    assert(head != le);
    struct Page *p = le2page(le, pra_page_link);//le2page宏可以根据链表元素获得对应的
Page指针p
    list_del(le);       //将进来最早的页面从队列中删除
    assert(p != NULL);
    *ptr_page = p; //将这一页的地址存储在ptr_page中

    return 0;
}
```

运行结果:

```
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
count is 7, total is 7
check_swap() succeeded!
++ setup timer interrupts
100 ticks
```

观察到 check_swap() succeeded! 实验结果正确

**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：**

现有的 swap_manager 框架是否足以支持在 ucore 中实现 extended clock 页替换算法。*ptep &PTE_A 用于表明该页是否被访问过，由此实现 extended clock 算法。

**a.需要被换出的页的特征是什么?**

在 FIFO 中，需要被换出的页是目前所有页中最早被调入的那一页。

**b.在 ucore 中如何判断具有这样特征的页?**

需要被换出的页位于队列的前端，即 mm->sm_priv->next 指示的那一页。

**c.何时进行换入和换出操作?**

当需要调用的页不在页表中时，并且页表已满，这时候需要进行换入和换出操作。

**【实验总结】**

**完成实验后，请分析 ucore_lab 中提供的参考答案，并请在实验报告中说明你的实现与参考答案的区别**

结合注释打代码，注释中每一步都很详细，按照注释打下来几乎和答案一模一样，有些语句的顺序不一样，但最终运行效果是一样的。

最后在写实验报告的时候给代码加上了注释。

**列出你认为本实验中重要的知识点，以及与对应的 OS 原理中的知识点，并简要说明你对二者的含义，关系，差异等方面的理解（也可能出现实验中的知识点没有对应的原理知识点）**

1.虚拟内存的概念。虚拟内存是计算机系统内存管理的一种技术。它使得应用程序认为它拥有连续的可用的内存（一个连续完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。

2.Page Fault 异常处理。页错误处理的时机就是求调页/页换入换出/处理的执行时机。相关异常处理完后，会返回到产生异常的指令处重新执行。因此，本实验涉及到页错误处理。

3.页面置换机制。本实验采用 FIFO 的置换策略。此外还有时钟页替换和改进的时钟页替换等算法。要完成页面置换，我们要掌握置换的时机以及置换页的选取，同时还要处理好虚存中的页较和扇区之间的映射关系。

**列出你认为 OS 原理中很重要，但在实验中没有对应上的知识点**

1.共享页。实验没有实现共享公共代码。

2.写时复制技术。提供快速进程创建，且最小化新创建进程必须分配的新页面的数量。

3.其他的置换算法如最优质换、LRU 置换近似 LRU 置换等

4.页缓冲算法。

5.预调页。

**心得体会**

本次实验从代码量来看，并不多。但是，涉及到的知识是综合的。这需要我们对之前做的实验也要有一定的印象。

虚拟内存的理论知识容易理解，而实验的过程就是将理论转化为实践的过程。

本次实验是操作系统的虚拟内存管理，目的是了解虚拟内存的 Page Fault 异常处理实现，并且了解页替换算法在操作系统中的实现。

总之，通过这次实验，我对操作系统内存管理相关知识掌握更加熟练了，我也意识到自己的操作系统编程能力有待提高。

**【参考文献】**

*《操作系统实验指导(清华大学)陈渝、向勇编著》*