Lab3 实验

练习 0：

1、使用 meld 的时候一定要小心仔细，千万不能合并出错。如果两份文件同一个函数的位置行数差别比较大的话,meld 可能无法识别。

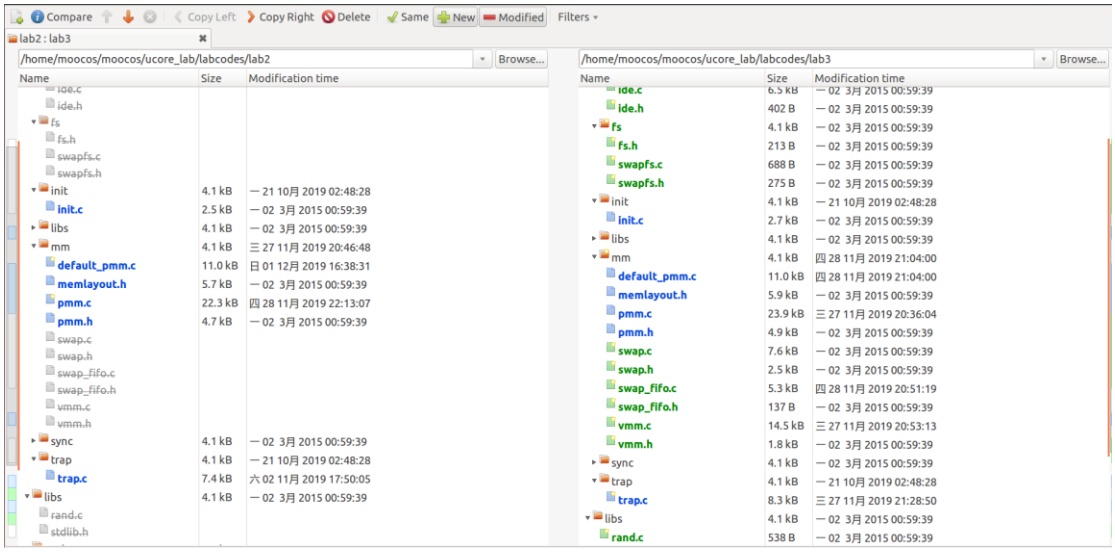

图 1 meld 比较 Lab2 和 Lab3 目录树的区别

2、Lab3 是在 Lab2 的基础上实现的，Lab3 中可以实现虚拟内存的管理，即虚拟地址空间可以比实际物理地址空间大。为了实现这个目标，需要在原有基础上引入虚拟内存的 Page Fault 异常处理以及页替换算法。同样，在做 Lab3 之前，复习了 Lab2 的部分内容。

练习 1：

1、练习 1 的问题相对来说比较简单，即给未被映射的地址映射上物理页。这一部分主要了解 do_pgfault 函数的内容，该函数是进行 Page Fault 异常处理的主要函数。在操作系统中，产生页访问异常的原因主要有三个：一是目标页帧不存在，即该线性地址和物理地址尚未建立映射或者已经撤销；二是相应的物理页帧不在内存中，页表项为空，但标志位 Present 为 0，这个时候就要通过页替换算法将物理页帧换到内存中；三是不满足访问权限，比如用户程序试图访问操作系统的内核空间等。而练习一要处理的就是第一种页访问异常，相关函数实现如下图所示。

```
if((ptep=get_pte(mm->pgdir,addr,1)) == NULL)//get_pte(pde_t *pgdir, uintptr_t la, bool create)
{
    cprintf("get_pte in do_pgfault failed\n");
    goto failed;
}
if(*ptep == 0)
{
    if(pgdir_alloc_page(mm->pgdir,addr,perm) == NULL)
    {
        cprintf("pgdir_alloc_page in do_pgfault failed\n");//print mistakes
        goto failed;
    }
}
```

图 2 练习 1 的代码截图

2、页目录表（PDE）作为一个双向链表储存目前所有页的物理地址和逻辑地址的对应，从 PDE 中选出被换出的页；页表（PTE）存储被换入的页的信息时需要将其映射到物理地址上，存储被换出的页的信息时映射此页在硬盘上的起始扇区位置。

3、页访问异常，CPU 在当前内核栈保存当前被打断的程序现场，即依次压入当前被打断程序使用的 EFLAGS，CS，EIP，ErrorCode，并将引起页访问异常的线性地址装到 CR2 寄存器中；由于页访问异常的中断号是 0xE，CPU 把异常中断号 0xE 对应的中断服务例程的地址(vectors.S 中的标号 vector14 处)加载到 CS 和 EIP 寄存器中，开始执行中断服务例程。

练习 2：

1、该练习主要实现 FIFO 的页面替换算法，FIFO 页替换算法的原理较为简单，即每次物理内存不够的时候将最先进入内存的页换出。FIFO 算法有一种异常现象，即在增加放置页的页帧的情况下反而使 Page Fault 次数增多。

2、在实现 FIFO 算法之前，需要补充 do_pgfault 函数，即处理由第二种情况导致的 Page Fault 相应的物理页帧不在内存中，页表项为空，但标志位 Present 为 0，这个时候就要通过页替换算法将物理页帧换到内存中，代码如下。

```
    else
    {
        if(swap_init_ok)//if swap_init_ok is 1
        {
            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,0); //Also right,why?
        }
        else
        {
            cprintf("no swap_init_ok but ptep is %x, failed\n",*ptep);
            goto failed;
        }
    }
```

图 3 补充 do_pgfault 函数

4、事实上，该题已经帮我们完成了 fifo 算法的框架，需要补充的地方的注释也很清楚，唯一要注意的是双向循环链表数据结构的特点，前后须保持一致，相关代码如下。

```
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;//sm_priv指向用来链接记录页访问情况的链表头
    list_entry_t *entry=&(page->pra_page_link);//visiting time

    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_after(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 set the addr of addr of this page to ptr_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)  set the addr of addr of this page to ptr_page
     list_entry_t *le = head->prev;
     struct Page *p = le2page(le, pra_page_link);
     list_del(le);
     *ptr_page = p;
     return 0;
}
```

图 4 实现 fifo 算法

3、页面替换算法有很多种，除了 fifo 算法之外，extended clock 页算法也是其中一种。认真阅读代码和实验报告可知，swap_manager 框架实际上里面是一

系列的函数指针，这些函数指针可以指向任意的页替换算法，因此该框架支持 extended clock 页替换算法。在 ucore 中页表项中的 PTE_A 访问位和 PTE_D 修改位会帮助完成需要换出的页的特征识别，详细操作在讲解代码时已经说明过，这里便不再赘述。