ucore实验三：虚拟内存管理

笔记本：	My Notebook		
创建时间：	2022/7/28 10:05	**更新时间：**	2022/7/29 23:56
作者：	赵凌珂		
URL：	https://chyyuu.gitbooks.io/ucore_os_docs/content/lab3/lab3_2_1_exercises.html		

作者：赵凌珂

练习一：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。
请在实验报告中简要说明你的设计实现过程。请回答如下问题：

- 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。
- 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？

完成do_pgfault函数：

```
int do_pgfault(struct mm_struct *mm, uint32_t error_code, uintptr_t addr) {
    int ret = -E_INVAL;

// 获取pgfault的虚拟地址所在虚拟页
    struct vma_struct *vma = find_vma(mm, addr);
    pgfault_num++;

// 如果当前访问的虚拟地址不在已经分配的虚拟页中
    if (vma == NULL || vma->vm_start > addr) {
        cprintf("not valid addr %x, and  can not find it in vma\n", addr);
        goto failed;
    }

// 检测错误代码
    switch (error_code & 3) {
```

```
    default:
        // 写，同时存在物理页，则写时复制

    case 2:
        // 读，同时不存在物理页
        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:
        // 读，同时存在物理页
        cprintf("do_pgfault failed: error code flag = read AND present\n");
        goto failed;

    case 0:
        // 写，同时不存在物理页面
        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;
        }
    }

// 设置页表条目的权限
    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;

// 查找当前虚拟地址的页表项
    if ((ptep = get_pte(mm->pgdir, addr, 1)) == NULL) {
        cprintf("get_pte in do_pgfault failed\n");
        goto failed;
    }

// 如果这个页表项所对应的物理页不存在
    if (*ptep == 0) {
        if (pgdir_alloc_page(mm->pgdir, addr, perm) == NULL) {
            cprintf("pgdir_alloc_page in do_pgfault failed\n");
            goto failed;
        }
    }

// 如果这个页表项所对应的物理页存在，但不在内存中
    else {
        // 如果swap已经初始化
        if(swap_init_ok) {
            struct Page *page=NULL;

            // 将目标数据加载到某块新的物理页中
            if ((ret = swap_in(mm, addr, &page)) != 0) {
                cprintf("swap_in in do_pgfault failed\n");
                goto failed;
            }
            // 将该物理页与对应的虚拟地址关联，同时设置页表
            page_insert(mm->pgdir, page, addr, perm);
            // 当前缺失的页已经加载回内存中，所以设置当前页为可swap
            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;
}
```

执行make qemu进行测试：

```
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'.
```

- 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。

分页机制的实现，确保了虚拟地址和物理地址之间的对应关系，一方面，通过查找虚拟地址是否存在于一二级页表中，可以容易发现该地址是否是合法的，同时可以通过修改映射关系即可实现页替换操作。另一方面，在实现页替换时涉及到换入换出：换入时需要将某个虚拟地址对应的磁盘的一页内容读入到内存中，换出时需要将某个虚拟页的内容写到磁盘中的某个位置。

- 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会把产生异常的线性地址存储在CR2寄存器中，并且把表示页访问异常类型的值（简称页访问异常错误码，errorCode）保存在中断栈中。之后通过上述分析的trap–> trap_dispatch–>pgfault_handler–>do_pgfault调用关系，一步步做出处理。

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。
请在实验报告中简要说明你的设计实现过程。
请在实验报告中回答如下问题：

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？
- 在ucore中如何判断具有这样特征的页？
- 何时进行换入和换出操作？

完成map_swappable函数：

```
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);
    list_add(head, entry);
    return 0;
}
```

完成swap_out_victim函数：

```
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);

     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;
}
```

执行make qemu进行测试：

```
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
100 ticks
```

1.需要被换出的页的特征是什么？
最早被换入，且最近没有再被访问的页

2.在ucore中如何判断具有这样特征的页？
通过判断是否访问过，对未访问过的物理页FIFO即可

3.何时进行换入和换出操作？
当需要调用的页不在页表中时，并且在页表已满的情况下，会引发PageFault，此时需要进行换入和换出操作