lab3 实验报告

练习1：给未被映射的地址映射上物理页

题目

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须要参考页面所在 VMA 的权限，同时须要注意映射物理页时须要操做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，若是经过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。git

请在实验报告中简要说明你的设计实现过程。请回答以下问题：github

1. 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。
2. 若是ucore的缺页服务例程在执行过程当中访问内存，出现了页访问异常，请问硬件要作哪些事情?

实现过程

do_pgfault函数已经完成了参数检查及错误检查等流程，根据注释不难完成剩下的流程。算法

1. 检查页面异常发生时的错误码的最低两位，即存在位和读/写位，若是发现错误则打印相关提示信息并返回。致使错误的缘由有：读没有读权限的内存、写没有写权限的内存、所读内容在内存中却读失败等。（原代码中已实现）编程
2. 用虚拟地址addr索引页目录表和页表，获得对应的页表项。这时要分两种状况讨论。app
3. 若是页表项为0，说明系统还没有为虚拟地址addr分配物理页，所以首先须要申请分配一个物理页；而后设置页目录表和页表，以创建虚拟地址addr到物理页的映射；最后，设置该物理页为swappable，而且把它插入到可置换物理页链表的末尾。框架
4. 若是页表项不为0，而又出现缺页异常，说明系统已创建虚拟地址addr到物理页的映射，但对应物理页已经被换出到磁盘中。这时一样须要申请分配一个物理页，把换出到磁盘中的那个页面的内容写到该物理页中；接下来和步骤3相似，一样须要创建虚拟地址addr到物理页的映射，一样须要把该物理页插入到可置换页链表的末尾。函数

问题1：页目录项和页表项对页替换算法的用处

答：页替换涉及到换入换出，换入时须要将某个虚拟地址vaddr对应于磁盘的一页内容读入到内存中，换出时须要将某个虚拟页的内容写到磁盘中的某个位置。而页表项能够记录该虚拟页在磁盘中的位置，为换入换出提供磁盘位置信息。页目录项则是用来索引对应的页表。测试

问题2：缺页服务例程出现页访问异常时，硬件须要作哪些事情

答：优化

1. 关中断
2. 保护现场。包括：将页访问异常的错误码压入内核栈的栈顶、将致使页访问异常的虚拟地址记录在cr2寄存器中、保存状态寄存器PSW及断点等。
3. 根据中断源，跳转到缺页服务例程

代码优化

对照答案对代码进行优化。this

1. do_pgfault调用get_pte时没有检查返回值。 个人代码：

```
pte_t *ptep = get_pte(mm->pgdir, addr, 1);
```

答案的代码：

```
pte_t *ptep=NULL;
// try to find a pte, if pte's PT(Page Table) isn't existed, then create a PT.
// (notice the 3th parameter '1')
if ((ptep = get_pte(mm->pgdir, addr, 1)) == NULL) {
    cprintf("get_pte in do_pgfault failed\n");
    goto failed;
}
```

1. do_pgfault调用pgdir_alloc_page和swap_in失败后没打印错误信息以方便定位。 个人代码：

```
if (*ptep == 0) {
        if (page = pgdir_alloc_page(mm->pgdir, addr, perm)) {
            ret = 0;
        }
    }
    else if (swap_init_ok) {
        swap_in(mm, addr, &page);

        if (0 == page_insert(mm->pgdir, page, addr, perm)) {
            swap_map_swappable(mm, addr, page, 0);
            ret = 0;
        }
    }
```

答案的代码：

```
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;
            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;
        }
        else {
            cprintf("no swap_init_ok but ptep is %x, failed\n",*ptep);
            goto failed;
        }
    }
```

练习2：补充完成基于FIFO的页面替换算法（须要编程）

题目

完成vmm.c中的do_pgfault函数，而且在实现FIFO算法的swap_fifo.c中完成map_swappable和swap_out_victim函数。经过对swap的测试。注意：在LAB3 EXERCISE 2处填写代码。执行make qemu后，若是经过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？

设计实现

- 根据练习1中的分析，可以知道最终page fault的处理会转交给do_pgfault函数进行处理，并且在该函数最后，如果通过page table entry获知对应的物理页被换出在外存中，则需要将其换入内存，之后中断返回之后便可以进行正常的访问处理。这里便涉及到了以下若干个关于换入的问题：
    - 应当在何处获取物理页在外存中的位置？
        - 物理页在外存中的位置保存在了PTE中；
    - 如果当前没有了空闲的内存页，应当将哪一个物理页换出到外存中去？

        - 这里涉及到了将物理页换出的问题，对于不同的算法会有不同的实现，在本练习中所实现的FIFO算法则选择将在内存中驻留时间最长的物理页换出；
- 在进行了上述分析之后，便可以进行具体的实现了，实现的流程如下：
    - 判断当前是否对交换机制进行了正确的初始化；
    - 将虚拟页对应的物理页从外存中换入内存；
    - 给换入的物理页和虚拟页建立映射关系；
    - 将换入的物理页设置为允许被换出；
- 而具体的代码实现如下所示：

```
ptep = get_pte(mm->pgdir, addr, 1); // 获取当前发生缺页的虚拟页对应的PTE
if (*ptep == 0) { // 如果需要的物理页是没有分配而不是被换出到外存中
    struct Page* page = pgdir_alloc_page(mm->pgdir, addr, perm); // 分配物理页，并且与对应的虚拟页建立映射关系
} else {
        if (swap_init_ok) { // 判断是否当前交换机制正确被初始化
            struct Page *page = NULL;
            swap_in(mm, addr, &page); // 将物理页换入到内存中
            page_insert(mm->pgdir, page, addr, perm); // 将物理页与虚拟页建立映射关系
            swap_map_swappable(mm, addr, page, 1); // 设置当前的物理页为可交换的
            page->pra_vaddr = addr; // 同时在物理页中维护其对应到的虚拟页的信息，这个语句本人觉得最好应当放置在page_insert函数中进行维护，在该建立映射关系的函数外对物理page对应的虚拟地址进行维护显得有些不太合适
        } else {
            cprintf("no swap_init_ok but ptep is %x, failed\n",*ptep);
            goto failed;
        }
}
```

- 对上述代码进行分析，发现当调用swap_in函数的时候，会进一步调用alloc_page函数进行分配物理页，一旦没有足够的物理页，则会使用swap_out函数将当前物理空间的某一页换出到外存，该函数会进一步调用sm（swap manager）中封装的swap_out_victim函数来选择需要换出的物理页，该函数是一个函数指针进行调用的，具体对应到了

```
_fifo_swap_out_victim
```

  函数（因为在本练习中使用了FIFO替换算法），在FIFO算法中，按照物理页面换入到内存中的顺序建立了一个链表，因此链表头处便指向了最早进入的物理页面，也就在在本算法中需要被换出的页面，因此只需要将链表头的物理页面取出，然后删掉对应的链表项即可；具体的代码实现如下所示：

```
list_entry_t *head=(list_entry_t*) mm->sm_priv; // 找到链表的入口
assert(head != NULL); // 进行一系列检查
assert(in_tick==0);
list_entry_t *le = list_next(head); // 取出链表头，即最早进入的物理页面
assert(le != head); // 确保链表非空
struct Page *page = le2page(le, pra_page_link); // 找到对应的物理页面的Page结构
list_del(le); // 从链表上删除取出的即将被换出的物理页面
*ptr_page = page;
```

- 在进行page fault处理中还有另外一个与交换相关的函数，swap_map_swappable，用于将指定的物理页面设置为可被换出，分析代码可以发现，该函数是sm中的swap_map_swappable函数的一个简单封装，对应到FIFO算法中实现的

  ```
  _fifo_swap_map_swappable
  ```

  函数，这也是在本次练习中需要进行实现的另外一个函数，这个函数的功能比较简单，就是将当前的物理页面插入到FIFO算法中维护的可被交换出去的物理页面链表中的末尾，从而保证该链表中越接近链表头的物理页面在内存中的驻留时间越长；该函数的一个具体实现如下所示：

  ```
  list_entry_t *head=(list_entry_t*) mm->sm_priv; // 找到链表入口
  list_entry_t *entry=&(page->pra_page_link); // 找到当前物理页用于组织成链表的
  list_entry_t
  assert(entry != NULL && head != NULL);
  list_add_before(head, entry); // 将当前指定的物理页插入到链表的末尾
  ```

问题回答

1.需要被换出的页的特征是什么?

- 该物理页在当前指针上一次扫过之前没有被访问过;
- 该物理页的内容与其在外存中保存的数据是一致的, 即没有被修改过;

2.在ucore中如何判断具有这样特征的页?

- 在ucore中判断具有这种特征的页的方式已经在上文设计方案中提及过了，具体为:
  - 假如某物理页对应的所有虚拟页中存在一个dirty的页，则认为这个物理页为dirty，否则不这么认为;
  - 假如某物理页对应的所有虚拟页中存在一个被访问过的页，则认为这个物理页为被访问过的，否则不这么认为;

3.何时进行换入和换出操作?

- 在产生page fault的时候进行换入操作;
- 换出操作源于在算法中将物理页的dirty从1修改成0的时候，因此这个时候如果不进行写出到外存，就会造成数据的不一致，具体写出内存的时机是比较细节的问题，可以在修改dirty的时候写入外存，或者是在这个物理页面上打一个需要写出的标记，到了最终删除这个物理页面的时候，如果发现了这个写出的标记，则在这个时候再写入外存；后者使用一个写延迟标记，有利于多个写操作的合并，从而降低缺页的代价;