Week8 Report

姓名：Yitong WANG(王奕童) 11910104@mail.sustech.edu.cn

学号：11910104

实验课时段：周五5-6节

实验课教师：Yun SHEN(沈昀) sheny@mail.sustech.edu.cn

实验课SA:

- Yining TANG(汤怡宁) 11811237@mail.sustech.edu.cn
- Yushan WANG(王宇杉) 11813002@mail.sustech.edu.cn

Q1. mm_struct

一个进程有一个mm_struct。

mm_struct是内存描述符，描述一个进程的虚拟地址空间，包含装入的可执行映像信息以及进程的页目录指针pgd。

> reference: https://blog.csdn.net/lf_2016/article/details/54346121?ref=myread

Q2. vma_struct

vma_struct结构体描述一段连续的虚拟地址，从 vm_start 到 vm_end，描述了虚拟地址空间的一个区间(简称虚拟区)。

> reference: https://blog.csdn.net/lf_2016/article/details/54346121?ref=myread

Q3. Page Fault

在进程CPU访问虚拟地址时，找不到对应的物理内存的时候出发缺页中断。主要有两种可能：

- 页表中不存在虚拟地址对应的PTE（Page Table Entry），属于下列两种之一
  - 虚拟地址无效
  - 虚拟地址有效，但没有分配物理内存页与建立映射关系

- 现有的权限不能操作对应的PTE

Q4. Major Page Fault

major page fault也称为hard page fault, 指需要访问的内存不在虚拟地址空间，也不在物理内存中，需要从慢速设备载入。处理方法是需要重新从外部的慢速设备中载入页的相关信息。

主要是在 `kern/mm/vmm.c` 文件中：

```
313 int
314 do_pgfault(struct mm_struct *mm, uint_t error_code, uintptr_t addr) {
315     int ret = -E_INVAL;
316     //try to find a vma which include addr
317     struct vma_struct *vma = find_vma(mm, addr);
318
319     pgfault_num++;
320     //If the addr is in the range of a mm's vma?
321     if (vma == NULL || vma->vm_start > addr) {
322         cprintf("not valid addr %x, and  can not find it in vma\n", addr);
323         goto failed;
324     }
325
326     uint32_t perm = PTE_U;
327     if (vma->vm_flags & VM_WRITE) {
328         perm |= (PTE_R | PTE_W);
329     }
330     addr = ROUNDDOWN(addr, PGSIZE);
331
332     ret = -E_NO_MEM;
333
334     pte_t *ptep=NULL;
335
336     ptep = get_pte(mm->pgdir, addr, 1);  //(1) try to find a pte, if pte's
337                                          //PT(Page Table) isn't existed, then
338                                          //create a PT.
339     if (*ptep == 0) {
340         if (pgdir_alloc_page(mm->pgdir, addr, perm) == NULL) {
341             cprintf("pgdir_alloc_page in do_pgfault failed\n");
342             goto failed;
343         }
344     } else {
345         if (swap_init_ok) {
346             struct Page *page = NULL;
347             swap_in(mm, addr, &page);  //According to the mm AND addr, try
348                                        //to load the content of right disk page
349                                        //    into the memory which page managed.
350             page_insert(mm->pgdir, page, addr, perm);  //According to the mm,
351                                                        //addr AND page, setup the
352                                                        //map of phy addr <--->
353                                                        //logical addr
354             swap_map_swappable(mm, addr, page,
355                                1);  //make the page swappable.
356             page->pra_vaddr = addr;
357         } else {
358             cprintf("no swap_init_ok but ptep is %x, failed\n", *ptep);
359             goto failed;
360         }
361    }
362
363    ret = 0;
364 failed:
365     return ret;
366 }
367
```

Q5. swap_in & swap_out

Swap只会发生在数据不在RAM（Random Access Memory）中的情况。

- Swap in：将硬盘中的数据放入主存或者RAM。时机是只要内存仍有空闲空间的时候，就会发生Swap in；或者在空间满了，Swap out清理出一片空间后，会再通过Swap in换进来
- Swap out：将RAM中数据放入硬盘，主要应用于内存空间满了的情况，需要选择一些不太可能经常访问的数据（这取决于具体算法）清理出新空间给新数据高速访问。

> reference: https://t4tutorials.com/swapping-swap-in-swap-out-in-operating-systems-os/