Week11 Report in Class (Fri56)

11911839 聂雨荷

Q1

一个进程有多少个 mm_struct? mm_struct 的作用是什么?

```
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 Page Table of these vma
    int map_count; // the count of these vma
    //...
}
```

- 一个进程只有一个 mm_struct，因为一个进程只有一个 VMA
- mm_struct 的作用是作为一个控制结构体来管理使用同一个页表的一组 VMA，它把一个页表对应的信息组合起来，包括 vma_struct 链表的首指针，对应的页表在内存里的指针，vma_struct 链表的元素个数

Q2

vma_struct 的作用是什么?

```
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
    uint_t vm_flags; // flags of vma
    list_entry_t list_link; // linear list link which
sorted by start addr of vma
};
```

vma_struct 结构体描述一段连续的虚拟地址，从 vm_start 到 vm_end

Q3

什么情况下会出触发缺页中断?

当 CPU 访问虚拟地址，而该虚拟地址找不到对应的物理内存时触发该异常

1. 页表中没有虚拟地址对应的PTE（虚拟地址无效或虚拟地址有效但没有分配物理内存页）
2. 现有权限无法操作对应的PTE

在 Linux 中 page fault 主要分为三种类型

1. major page fault (hard page fault): 访问的虚拟地址内容不在内存中，需要从外设载入。常见于内容页被置换到外设交换区中，需要 将交换区中的页面重新载入内存。
2. minor page fault (soft page fault): 虚拟地址在页表中没有建立映射，常见于进程申请虚拟内存后初次操作内存，及多个进程访问共 享内存尚未建立虚拟地址映射的情况。
3. invalid fault: 访问的虚拟地址不合法

Q4

major page fault 是如何处理的？在实验代码中对应哪一段?

major page fault (hard page fault): 访问的虚拟地址内容不在内存中，需要从外设载入。常见于内容页被置换到外设交换区中，需要 将交换区中的页面重新载入内存，他的 处理方法是

1. 通过 `get_pte` 函数获取三级页表项
2. 如果页表项不为 0，则说明对应的页面被换出了
3. 根据 PTE 在硬盘上找到对应的页面
4. 将页面换入，建立映射
5. 最后标记这个页面时可以换出的
6. 记录该页的虚拟地址

对应实验代码中 kern/mm/vmm.c 的 `do_pgfault` 的一段代码

```
int do_pgfault(struct mm_struct *mm, uint_t error_code,
uintptr_t addr) {
    // ... unrelated
    ptep = get_pte(mm->pgdir, addr, 1); //(1) try to find a
pte, if pte's
    //PT(Page Table) isn't existed, then
    //create a PT.
    if (*ptep == 0) {
        // ... unrelated
    }else { // major page fault
```

```
        if (swap_init_ok) {
            struct Page *page = NULL;
            swap_in(mm, addr, &page);  //According to the
mm AND addr, try to load the content of right disk page
into the memory which page managed.
            page_insert(mm->pgdir, page, addr, perm);
 //According to the mm, addr AND page, setup the map of phy
addr <---> logical addr
            swap_map_swappable(mm, addr, page, 1);  //make
the page swappable.
            page->pra_vaddr = addr;
        } else {
            cprintf("no swap_init_ok but ptep is %x,
failed\n", *ptep);
            goto failed;
        }
    }
}
```