浙江大学

本科实验报告

课程名称：	操作系统
实验名称：	RV64 缺页异常处理
姓　　名：	蔡雨谦
学　　院：	计算机学院
系：	计算机系
专　　业：	计算机科学与技术
学　　号：	3210102466
指导教师：	寿黎但

2023 年　　12 月　　21 日

浙江大学实验报告

一、 实验目的

- 通过 vm_area_struct 数据结构实现对 task 多区域虚拟内存的管理。
- 在 Lab4 实现用户态程序的基础上，添加缺页异常处理 Page Fault Handler。

二、 实验环境

- Ubuntu 22.04.2 LTS Windows Subsystem for Linux 2

三、 操作方法与实验步骤

1、实现 VMA

1.1 修改 proc.h

```
#define VM_X_MASK         0x0000000000000008
#define VM_W_MASK         0x0000000000000004
#define VM_R_MASK         0x0000000000000002
#define VM_ANONYM         0x0000000000000001


struct vm_area_struct {
    uint64_t vm_start;          /* VMA 对应的用户态虚拟地址的开始    */
    uint64_t vm_end;            /* VMA 对应的用户态虚拟地址的结束    */
    uint64_t vm_flags;          /* VMA 对应的 flags */
    /* 如果对应了一个文件，
    那么这块 VMA 起始地址对应的文件内容相对文件起始位置的偏移量，
    也就是 ELF 中各段的 p_offset 值 */
    uint64_t vm_content_offset_in_file;
    /* 对应的文件内容的长度。
    思考为什么还需要这个域?
    和 (vm_end-vm_start)一比，
    不是冗余了吗? */
    uint64_t vm_content_size_in_file;
};
```

```
struct task_struct {
    uint64_t state;
    uint64_t counter;
    uint64_t priority;
    uint64_t pid;

    struct thread_struct thread;
    pagetable_t pgd;

    uint64_t vma_cnt;
    struct vm_area_struct vmas[0];
                                这个定
};
```

1.2 do_mmap

虽然 vmas 在结构体中定义为长度为零的数组，但实际上可以向后扩展。

```
void do_mmap(struct task_struct *task, uint64_t addr, uint64_t length, uint64_t flags,
             uint64_t vm_content_offset_in_file, uint64_t vm_content_size_in_file)
{
    task->vma_cnt++;
    struct vm_area_struct *vma = &(task->vmas[task->vma_cnt]);
    vma->vm_start = addr;
    vma->vm_end = addr + length;
    vma->vm_flags = flags;
    vma->vm_content_offset_in_file = vm_content_offset_in_file;
    vma->vm_content_size_in_file = vm_content_size_in_file;
}
```

1.3 find_vma

返回 addr 所在的 vma 的地址。

```
struct vm_area_struct *find_vma(struct task_struct *task, uint64_t addr)
{
    for (int i = 0; i < task->vma_cnt; i++)
    {
        if (addr <= task->vmas[i].vm_end)
        {
            return &(task->vmas[i]);
        }
    }
    return NULL;
}
```

2、Page Fault Handler

2.1 task_init

取消之前实验中对 U-MODE 代码以及栈进行的映射，再用 do_mmap 替换 create_mapping。

```
    Elf64_Ehdr *ehdr = (Elf64_Ehdr *)_sramdisk;

    uint64_t phdr_start = (uint64_t)ehdr + ehdr->e_phoff;
    int phdr_cnt = ehdr->e_phnum;

    Elf64_Phdr *phdr;
    int load_phdr_cnt = 0;
    for (int i = 0; i < phdr_cnt; i++)
    {
        phdr = (Elf64_Phdr *)(phdr_start + sizeof(Elf64_Phdr) * i);
        if (phdr->p_type == PT_LOAD)
        {
            //代码和数据区域：该区域从 ELF 给出的 Segment 起始用户态虚拟地址 phdr->p_vaddr 开始，对应文件中偏移量为 phdr->p_
            do_mmap(task, phdr->p_vaddr, phdr->p_memsz, ((phdr->p_flags) << 1), phdr->p_offset, phdr->p_filesz);
        }
    }
    // 用户栈：范围为 [USER_END - PGSIZE, USER_END) ，权限为 VM_READ | VM_WRITE，并且是匿名的区域。
    do_mmap(task, USER_END - PGSIZE, PGSIZE, VM_R_MASK | VM_W_MASK | VM_ANONYM, 0, 0);
    // 初始化我们刚刚在 thread_struct 中添加的三个变量
    task[i]->thread.sepc = task[i]->thread.sepc = ehdr->e_entry;
    uint64 sstatus = csr_read(sstatus);
    task[i]->thread.sstatus = sstatus & ~(0x100) | 0x40020; // spp[8]=0   spie[5]=1  sum[18]=1
    task[i]->thread.sscratch = USER_END;
    task[i]->pgd = (unsigned long)task[i]->pgd - PA2VA_OFFSET;
}
```

2.2 trap_handler

增添一个 else 分支即可。

```
else if(scause==12||scause==13||scause==15)
{
    do_page_fault(regs);
}
```

2.3 do_page_fault

找到访问出错的虚拟内存地址所在的 VMA,判断该 VMA 是否匿名。若匿名则清零，否则将 uapp 的内容拷贝到新分配的页面中。

```
//1. 通过 stval 获得访问出错的虚拟内存地址（Bad Address）
uint64 bad_address = regs->stval;
//2. 通过 find_vma() 查找 Bad Address 是否在某个 vma 中
struct vm_area_struct *vma = find_vma(current, bad_address);
if (vma != NULL)
{
    // 3. 分配一个页，将这个页映射到对应的用户地址空间
    uint64 new_page = (uint64)kalloc();
    create_mapping((uint64 *)((uint64)current->pgd + PA2VA_OFFSET),
                   bad_address >> 12 << 12, new_page - PA2VA_OFFSET, PGSIZE, (vma->vm_flags) | 0x11);
    // 4. 通过 (vma->vm_flags & VM_ANONYM) 获得当前的 VMA 是否是匿名空间
    // 5. 根据 VMA 匿名与否决定将新的页清零或是拷贝 uapp 中的内容
    // 匿名，将新的页清零
    if (vma->vm_flags & VM_ANONYM)
    {
        new_page = 0;
    }
    // 非匿名，拷贝 uapp 中的内容
    else
    {
        char *uapp = (char *)(((uint64)_sramdisk + (bad_address - vma->vm_start)) >> 12 << 12);
        for (int i = 0; i < PGSIZE; i++)
        {
            (&new_page)[i] = uapp[i];
        }
    }
}
else
{
    return;
}
```

四、思考题

1.uint64_t vm_content_size_in_file; 对应的文件内容的长度。为什么还需要这个域?

本实验只有一个文件 uapp，如果有多个文件就要分别保存各自的文件长度。而且文件可能相对 VMA 起始地址有偏移。

2. struct vm_area_struct vmas[0]; 为什么可以开大小为 0 的数组? 这个定义可以和前面的 vma_cnt 换个位置吗?

因为 vmas 后没有定义新的变量，有空闲的地址空间，可以在后期进行扩展。如果和前面的 vma_cnt 换位置，由于数组的地址空间要求是连续的，数组的大小

就固定了。

五、讨论心得

课程中讲到的 page fault 相对来说要更 high-level 也更容易理解，而进入实验里就涉及较底层的设计，包括地址的具体转换方法，页面的创建与管理等，增添了许多细节。

我也查询过实际的 linux 中使用的 VMA，是用链表来储存所有的 VMA，为了方便搜索和维护还使用了红黑树的结构，这次实验助教替大家着想简化了数据结构。