

# 浙江大学

## 本科实验报告

课程名称：操作系统

姓 名：董佳鑫

学 院：计算机学院

系：计算机系

专 业：计算机科学与技术

学 号：3210102181

指导教师：寿黎但

## Lab 5: RV64 缺页异常处理

### 实验目的

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

### 实验过程和操作步骤

#### 1. 实现 VMA

阅读文档以后，实现 `find_vma` 和 `do_mmap` 两个函数，这两个函数的实现较为直观。

```
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 *new_vma = &_task->vmas[_task->vma_cnt - 1];
    new_vma->vm_start = addr;
    new_vma->vm_end = addr + length;
    new_vma->vm_flags = flags;
    new_vma->vm_content_offset_in_file = vm_content_offset_in_file;
    new_vma->vm_content_size_in_file = vm_content_size_in_file;
}

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

#### 2. 修改 PAGE FAULT HANDLER

本次实验需要添加新的异常处理：`page fault`。对应的 `scause` 有三个值。因此在 `trap_handler` 中要添加对这三种异常的捕获，并在捕获后调用 `do_page_fault` 函数，处理缺页异常。

具体缺页异常处理如图。

```
uint64_t bad_addr = regs->stval;
struct vm_area_struct *vma = find_vma(current, bad_addr);
if(vma != NULL){
    uint64_t new_addr = kalloc();
    unsigned long* pgtbl = (((unsigned long)current->pgd & 0xffffffff) << 12) + PA2VA_OFFSET;
    create_mapping_sub(pgtbl, bad_addr, new_addr-PA2VA_OFFSET, PGSIZE, (vma->vm_flags & ~(uint64_t)VM_ANONYM) | 0x11);
    if( !(vma->vm_flags & VM_ANONYM) ){
        uint64_t src_addr = (uint64_t)(sramdisk) + vma->vm_content_offset_in_file;
        uint64_t offset = (uint64_t)(bad_addr) - PGROUNDDOWN(bad_addr);
        uint64_t sz = (vma->vm_start + vma->vm_content_size_in_file > bad_addr) ? PGSIZE - offset : 0;
        memcpy(new_addr+offset, src_addr, sz);
    }
}
```

首先查找出现缺页异常的地址有没有被记录到 VMA 中，如果有，就要对 `bad_addr` 所在页建立映射。并且如果该区域不是匿名区域，还应该拷贝文件的内容。这里要注意在拷贝的时候，由于 `bad_addr` 可能会落在未初始化的区域，这块区域并没有占用文件大小，也就是 `vm_content_offset_in_file`。因此如果 `bad_addr` 超过了 `vm_content_offset_in_file` 的范围，则不应该拷贝那些未初始化的值，而应该直接默认使用 0 值。

完成上述代码后，运行程序，可以看到，在处理了缺页异常后，程序最终找到了指令所在地址，程序正确运行。

```
...setup_vm final done!
...proc_init done!
[S-MODE] 2022 Hello RISC-V
[S-MODE] switch to [PID = 1 COUNTER = 4]
[S-MODE] Page Fault Interrupt, scause: 000000000000000c, stval: 0000000000100e8, sepc: 0000000000100e8
[S-MODE] Page Fault Interrupt, scause: 000000000000000f, stval: 0000003fffffff8, sepc: 000000000010124
[S-MODE] Page Fault Interrupt, scause: 000000000000000d, stval: 000000000011880, sepc: 000000000010140
[PID = 1] is running, variable: 0
[PID = 1] is running, variable: 1
[PID = 1] is running, variable: 2
[PID = 1] is running, variable: 3
[PID = 1] is running, variable: 4
[PID = 1] is running, variable: 5
[PID = 1] is running, variable: 6
[S-MODE] switch to [PID = 4 COUNTER = 5]
[S-MODE] Page Fault Interrupt, scause: 000000000000000c, stval: 0000000000100e8, sepc: 0000000000100e8
[S-MODE] Page Fault Interrupt, scause: 000000000000000f, stval: 0000003fffffff8, sepc: 000000000010124
[S-MODE] Page Fault Interrupt, scause: 000000000000000d, stval: 000000000011880, sepc: 000000000010140
[PID = 4] is running, variable: 0
[PID = 4] is running, variable: 1
[PID = 4] is running, variable: 2
[PID = 4] is running, variable: 3
[PID = 4] is running, variable: 4
[PID = 4] is running, variable: 5
[PID = 4] is running, variable: 6
[PID = 4] is running, variable: 7
[PID = 4] is running, variable: 8
[PID = 4] is running, variable: 9
[PID = 4] is running, variable: 10
[S-MODE] switch to [PID = 3 COUNTER = 8]
[S-MODE] Page Fault Interrupt, scause: 000000000000000c, stval: 0000000000100e8, sepc: 0000000000100e8
[S-MODE] Page Fault Interrupt, scause: 000000000000000f, stval: 0000003fffffff8, sepc: 000000000010124
[S-MODE] Page Fault Interrupt, scause: 000000000000000d, stval: 000000000011880, sepc: 000000000010140
[PID = 3] is running, variable: 0
[PID = 3] is running, variable: 1
[PID = 3] is running, variable: 2
[PID = 3] is running, variable: 3
[PID = 3] is running, variable: 4
[PID = 3] is running, variable: 5
[PID = 3] is running, variable: 6
```

```
[PID = 4] is running, variable: 8
[PID = 4] is running, variable: 9
[PID = 4] is running, variable: 10
[S-MODE] switch to [PID = 3 COUNTER = 8]
[S-MODE] Page Fault Interrupt, scause: 000000000000000c, stval: 0000000000100e8, sepc: 0000000000100e8
[S-MODE] Page Fault Interrupt, scause: 000000000000000f, stval: 0000003fffffff8, sepc: 000000000010124
[S-MODE] Page Fault Interrupt, scause: 000000000000000d, stval: 000000000011880, sepc: 000000000010140
[PID = 3] is running, variable: 0
[PID = 3] is running, variable: 1
[PID = 3] is running, variable: 2
[PID = 3] is running, variable: 3
[PID = 3] is running, variable: 4
[PID = 3] is running, variable: 5
[PID = 3] is running, variable: 6
[PID = 3] is running, variable: 7
[PID = 3] is running, variable: 8
[PID = 3] is running, variable: 9
[PID = 3] is running, variable: 10
[PID = 3] is running, variable: 11
[PID = 3] is running, variable: 12
[PID = 3] is running, variable: 13
[PID = 3] is running, variable: 14
[S-MODE] switch to [PID = 2 COUNTER = 9]
[S-MODE] Page Fault Interrupt, scause: 000000000000000c, stval: 0000000000100e8, sepc: 0000000000100e8
[S-MODE] Page Fault Interrupt, scause: 000000000000000f, stval: 0000003fffffff8, sepc: 000000000010124
[S-MODE] Page Fault Interrupt, scause: 000000000000000d, stval: 000000000011880, sepc: 000000000010140
[PID = 2] is running, variable: 0
[PID = 2] is running, variable: 1
[PID = 2] is running, variable: 2
[PID = 2] is running, variable: 3
[PID = 2] is running, variable: 4
[PID = 2] is running, variable: 5
[PID = 2] is running, variable: 6
[PID = 2] is running, variable: 7
[PID = 2] is running, variable: 8
[PID = 2] is running, variable: 9
[PID = 2] is running, variable: 10
[PID = 2] is running, variable: 11
[PID = 2] is running, variable: 12
[PID = 2] is running, variable: 13
```

```

[PID = 2] is running, variable: 0
[PID = 2] is running, variable: 1
[PID = 2] is running, variable: 2
[PID = 2] is running, variable: 3
[PID = 2] is running, variable: 4
[PID = 2] is running, variable: 5
[PID = 2] is running, variable: 6
[PID = 2] is running, variable: 7
[PID = 2] is running, variable: 8
[PID = 2] is running, variable: 9
[PID = 2] is running, variable: 10
[PID = 2] is running, variable: 11
[PID = 2] is running, variable: 12
[PID = 2] is running, variable: 13
[S-MODE] switch to [PID = 1 COUNTER = 2]
[PID = 1] is running, variable: 7
[PID = 1] is running, variable: 8
[PID = 1] is running, variable: 9
[PID = 1] is running, variable: 10
[S-MODE] switch to [PID = 2 COUNTER = 5]
[PID = 2] is running, variable: 14
[PID = 2] is running, variable: 15
[PID = 2] is running, variable: 16
[PID = 2] is running, variable: 17
[PID = 2] is running, variable: 18
[PID = 2] is running, variable: 19
[PID = 2] is running, variable: 20
[PID = 2] is running, variable: 21
[PID = 2] is running, variable: 22
[S-MODE] switch to [PID = 4 COUNTER = 5]
[PID = 4] is running, variable: 11
[PID = 4] is running, variable: 12
[PID = 4] is running, variable: 13
[PID = 4] is running, variable: 14
[PID = 4] is running, variable: 15
[PID = 4] is running, variable: 16

```

## 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？

由于在内存中的长度往往会大于在文件中的长度，因为存在一些未初始化的变量不占用文件空间。我们的 `[vma_start, vma_end)` 的范围可能会大于 `vm_content_size_in_file`，这就导致我们的 `bad_addr` 可能会落在未初始化区域，而这部分区域在 `demand paging` 中是不需要拷贝的，因此我们需要使用 `vm_content_size_in_file` 来决定是否对刚映射的一个 `page` 进行拷贝文件内容。

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？

大小为 0 的数组是可变数组。这样处理的好处是不会占用更多的内存空间，只有当需要的时候才动态申请内存空间（使用 `kalloc()` 函数），并且比链表实现更为方便。不可以和 `vma_cnt` 换位置，它必须处在结构体的末尾，才能动态决定结构体的 `size` 以节省空间。

## 讨论心得

本次实验的难度相比前两个有了很大的降低，基本没有遇到太多问题。唯一需要注意的就是 `do_page_fault` 函数的实现，需要理清各种细节。关键点就是在新建的页建立映射后是否要进行内容拷贝，`bad_addr` 如果没有超过

`vm_content_size_in_file` 的范围则正常拷贝，否则不进行拷贝。对于文件中的大小和内存中的大小的区别一定要掌握清楚。另外一个值得关注的地方就是在建立映射或者 `do_mmap` 的时候，权限设置要谨慎，由于权限不正确而导致缺页异常处理失败的错误也非常致命。