Lab 5: RV64 缺页异常处理

张志心 3210106357

1 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。

2 实验环境

- Same as previous labs.

3 实验步骤

3.1 实现 VMA

修改 `proc.h`，增加如下相关结构。

```
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 */

    /* uint64_t file_offset_on_disk */
    uint64_t vm_content_offset_in_file;

    uint64_t vm_content_size_in_file;
};


/* 线程数据结构 */
struct task_struct {
    struct thread_info thread_info;
    uint64 state;    // 线程状态
    uint64 counter;  // 运行剩余时间
    uint64 priority; // 运行优先级 1最低 10最高
    uint64 pid;      // 线程id

    struct thread_struct thread;

    uint64 satp;
    pagetable_t pgd;

    uint64_t vma_cnt;
    struct vm_area_struct vmas[0];
};

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

```
#define VM_ANONYM         0x0000000000000001

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);

struct vm_area_struct *find_vma(struct task_struct *task, uint64_t addr);
```

其中：

- `find_vma` 查找包含某个 addr 的 vma。
- `do_mmap` 创建一个新的 vma。

```
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) {

    #define nowvma task->vmas[task->vma_cnt]
    if (task->vma_cnt >= 49) {
        printk("Warning: too many vm areas!");
    }

    nowvma.vm_start = addr;
    nowvma.vm_end = addr + length;
    nowvma.vm_content_offset_in_file = vm_content_offset_in_file;
    nowvma.vm_content_size_in_file = vm_content_size_in_file;
    nowvma.vm_flags = flags;

    #undef nowvma
    task->vma_cnt++;
}

struct vm_area_struct *find_vma(struct task_struct *task, uint64_t addr) {
    int vma_idx;
    #define nowvma task->vmas[vma_idx]
    // more semantic information can help compile-time checks
    #define in(addr, begin, end) (((addr) >= (begin)) && ((addr) < (end)))

    for (vma_idx = 0; vma_idx < task->vma_cnt; vma_idx++)
        if (in(addr, nowvma.vm_start, nowvma.vm_end))
            break;

    if (vma_idx == task->vma_cnt) return NULL;
    else return (task->vmas) + vma_idx;

    #undef nowvma
    #undef in
}
```

3.2 Page Fault Handler

修改 `task_init` 函数代码，更改为 `Demand Paging`。

- 取消之前实验中对 `U-MODE` 代码以及栈进行的映射
- 调用 `do_mmap` 函数，建立用户 task 的虚拟地址空间信息，在本次实验中仅包括两个区域：

  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_offset` 开始的部分。权限参考 `phdr->p_flags` 进行设置。

  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。

```c
void task_init() {
    printk("Entering task init\n");
    test_init(NR_TASKS);
    idle = (struct task_struct *)kalloc();
    if(!idle) return;
    current = task[0] = idle;
    idle->state = TASK_RUNNING;
    idle->counter = 0;
    idle->priority = 0;
    idle->pid = 0;

    for(int i = 1; i < NR_TASKS; ++i) {
        task[i] = (struct task_struct *)kalloc();
        if(!task[i]) return;
        task[i]->vma_cnt = 0;
        task[i]->thread.sepc = USER_START;
        task[i]->thread.sstatus &= ~(1 << 8);
        task[i]->thread.sstatus |= (1 << 5);
        task[i]->thread.sstatus |= (1 << 18);
        task[i]->thread.sscratch = USER_END;
        task[i]->pgd = (unsigned long*)kalloc();
        memcpy(task[i]->pgd, swapper_pg_dir, PGSIZE);
        task[i]->thread.sp = (uint64_t)task[i] + PGSIZE;
        task[i]->thread_info.kernel_sp = (uint64_t)task[i] + PGSIZE;
        task[i]->thread_info.user_sp = (uint64_t)kalloc();

        #define STACK_SIZE (PGSIZE << 4)
        #define UAPP_SIZE (1 << 24) // 16 MiB

        uint64_t va = USER_END - STACK_SIZE;
        do_mmap(task[i], va, STACK_SIZE, VM_R_MASK | VM_W_MASK | VM_ANONYM, 0, 0);

        Elf_Ehdr *header = (void *) _sramdisk;
        Elf_Half phnum = header->e_phnum;
        Elf_Off phoff = header->e_phoff;
        printk("get %u segments\n", phnum);
        while (phnum--) {
            Elf_Phdr *segment = (void *) _sramdisk + phoff;
            phoff += header->e_phentsize;
            printk("Segment at 0x%08x with memory size 0x%06x\n", segment->p_vaddr,
segment->p_memsz);
```

```c
            if (!(segment->p_type == PT_LOAD)) {
                printk("Not a PT_LOAD segment, ignoring...\n");
                continue;
            }

            void *file_pt = _sramdisk + segment->p_offset;
            #define MIN(a, b) ((a) < (b) ? (a) : (b))
            #define CEIL(sza, szb) ((((sza) + ((szb) - 1)) & (~((szb) - 1))) /
PGSIZE)
            #define ROUNDUP(a, sz)      ((((uint64_t)a) + (sz) - 1) & ~((sz) - 1))
            #define ROUNDDOWN(a, sz)    ((((uint64_t)a)) & ~((sz) - 1))

            uint64_t seg_flag_vma = 0;
            if (segment->p_flags & PF_R)
                seg_flag_vma |= VM_R_MASK;
            if (segment->p_flags & PF_W)
                seg_flag_vma |= VM_W_MASK;
            if (segment->p_flags & PF_X)
                seg_flag_vma |= VM_X_MASK;

            do_mmap(task[i], segment->p_vaddr, segment->p_memsz, seg_flag_vma,
segment->p_offset, segment->p_filesz);

            printk("Load finished\n");
        }

        uint64 satp = csr_read(satp);
        satp = (satp >> 44) << 44;
        satp |= ((uint64)(task[i]->pgd) - PA2VA_OFFSET) >> 12;
        task[i]->satp = satp;

        task[i]->state = TASK_RUNNING;
        task[i]->counter = task_test_counter[i];
        task[i]->priority = task_test_priority[i];
        task[i]->pid = i;
        task[i]->thread.ra = (uint64)__dummy;
        task[i]->thread.sp = PGSIZE + (long)task[i];
    }

    printk("...proc_init done!\n");
}
```

实现 **Page Fault** 的检测与处理:

- 修改 `trap.c` 增加捕获 **Page Fault** 的逻辑。

```c
void trap_handler(uint64 scause, uint64 sepc, struct pt_regs* regs) {
    uint64 stval = csr_read(stval);
    if ((long) scause < 0 && (scause & ((1ul << 63) - 1)) == 5) {
        // printk("%s", "Get STI!\n");
        clock_set_next_event();
        do_timer();
        regs->sepc += 4;
        return ;
```

```
        // printk("[S] Supervisor Mode Timer Interrupt!\n");
    } else if (scause== 8) {
        syscall(regs);
        return;
    } else if (scause == 15) { // store
        do_page_fault(stval, PGF_W);
        return ;
    } else if (scause == 13) { // load
        do_page_fault(stval, PGF_R);
        return ;
    } else if (scause == 12) { // inst
        do_page_fault(stval, PGF_X);
        return ;
    }
    printk("[S] Unhandled trap: scause = %lx, sepc = %llx, stval = %llx\n", scause,
sepc, stval);
    while(1);
}
```

- 实现缺页异常的处理函数 `do_page_fault`。

```
void do_page_fault(uint64 addr, int type) {
    struct vm_area_struct *find_ret = find_vma(current, addr);
    if (!find_ret) {
        printk("SEGMENT FAULT at %llx no segment type %d\n", addr, type);
        return ;
    }
    uint64 pg_start = ROUNDDOWN(addr, PGSIZE);
    uint64 vm_flags = 0x10;
    if (find_ret->vm_flags & VM_R_MASK)
        vm_flags |= 0x2;
    else if (type == PGF_R) {
        printk("SEGMENT FAULT at %llx at read\n", addr);
        return ;
    }
    if (find_ret->vm_flags & VM_W_MASK)
        vm_flags |= 0x4;
    else if (type == PGF_W) {
        printk("SEGMENT FAULT at %llx at write\n", addr);
        return ;
    }
    if (find_ret->vm_flags & VM_X_MASK)
        vm_flags |= 0x8;
    else if (type == PGF_X) {
        printk("SEGMENT FAULT at %llx at inst\n", addr);
        return ;
    }
    uint64_t sa = kalloc();
    create_mapping(current->pgd, pg_start, sa - PA2VA_OFFSET, PGSIZE, vm_flags);
    memset((void *) sa, 0, PGSIZE);
    if (find_ret->vm_flags & VM_ANONYM)
        return ;
    if (pg_start <= find_ret->vm_start) {
        uint64_t realstart = find_ret->vm_start - pg_start + sa;
```

```
        uint64_t bytes_to_copy = MIN(find_ret->vm_content_size_in_file, PGSIZE -
(find_ret->vm_start - pg_start));
        // realstart + bytes_to_copy <= PGSIZE + sa
        memcpy((void *) realstart, _sramdisk + find_ret->vm_content_offset_in_file,
bytes_to_copy);
    } else {
        uint64_t bytes_rest = find_ret->vm_content_size_in_file - (pg_start -
find_ret->vm_start);
        uint64_t real_offset = find_ret->vm_content_offset_in_file + (pg_start -
find_ret->vm_start);
        uint64_t bytes_to_copy = MIN(bytes_rest, PGSIZE);

        memcpy((void *) pg_start, _sramdisk + real_offset, bytes_to_copy);
    }
}
```

4 测试

每个 uapp 一共会发生 2 次 Page Fault。

```
Not a PT_LOAD segment, ignoring...
...proc_init done!
2022 Hello RISC-V
switch to [PID = 1 COUNTER = 4]
[S] PAGE FAULT stval 0000000000000000 scause 000000000000000c sepc 0000000000000
[S] PAGE FAULT stval 0000003ffffffff8 scause 000000000000000f sepc 0000000000000
[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
[PID = 1] is running, variable: 7
switch to [PID = 4 COUNTER = 5]
[S] PAGE FAULT stval 0000000000000000 scause 000000000000000c sepc 0000000000000
[S] PAGE FAULT stval 0000003ffffffff8 scause 000000000000000f sepc 0000000000000
[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
switch to [PID = 3 COUNTER = 8]
[S] PAGE FAULT stval 0000000000000000 scause 000000000000000c sepc 0000000000000
[S] PAGE FAULT stval 0000003ffffffff8 scause 000000000000000f sepc 0000000000000
[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
```

```
[PID = 3] is running, variable: 12
[PID = 3] is running, variable: 13
[PID = 3] is running, variable: 14
[PID = 3] is running, variable: 15
switch to [PID = 2 COUNTER = 9]
[S] PAGE FAULT stval 0000000000000000 scause 000000000000000c sepc 0000000000000000
[S] PAGE FAULT stval 0000003ffffffff8 scause 000000000000000f sepc 000000000000003c
[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: 14
[PID = 2] is running, variable: 15
[PID = 2] is running, variable: 16
[PID = 2] is running, variable: 17
SET [PID = 1 COUNTER = 1]
SET [PID = 2 COUNTER = 4]
SET [PID = 3 COUNTER = 10]
SET [PID = 4 COUNTER = 4]
switch to [PID = 1 COUNTER = 1]
[PID = 1] is running, variable: 8
[PID = 1] is running, variable: 9
switch to [PID = 2 COUNTER = 4]
[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
[PID = 2] is running, variable: 23
[PID = 2] is running, variable: 24
[PID = 2] is running, variable: 25
switch to [PID = 4 COUNTER = 4]
[PID = 4] is running, variable: 10
[PID = 4] is running, variable: 11
```

```
[PID = 2] is running, variable: 24
[PID = 2] is running, variable: 25
switch to [PID = 4 COUNTER = 4]
[PID = 4] is running, variable: 10
[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
[PID = 4] is running, variable: 17
switch to [PID = 3 COUNTER = 10]
[PID = 3] is running, variable: 16
[PID = 3] is running, variable: 17
[PID = 3] is running, variable: 18
[PID = 3] is running, variable: 19
[PID = 3] is running, variable: 20
[PID = 3] is running, variable: 21
[PID = 3] is running, variable: 22
[PID = 3] is running, variable: 23
[PID = 3] is running, variable: 24
[PID = 3] is running, variable: 25
[PID = 3] is running, variable: 26
[PID = 3] is running, variable: 27
[PID = 3] is running, variable: 28
[PID = 3] is running, variable: 29
[PID = 3] is running, variable: 30
[PID = 3] is running, variable: 31
[PID = 3] is running, variable: 32
[PID = 3] is running, variable: 33
[PID = 3] is running, variable: 34
[PID = 3] is running, variable: 35
SET [PID = 1 COUNTER = 10]
SET [PID = 2 COUNTER = 10]
SET [PID = 3 COUNTER = 5]
SET [PID = 4 COUNTER = 2]
switch to [PID = 4 COUNTER = 2]
[PID = 4] is running, variable: 18
[PID = 4] is running, variable: 19
[PID = 4] is running, variable: 20
[PID = 4] is running, variable: 21
switch to [PID = 3 COUNTER = 5]
[PID = 3] is running, variable: 36
[PID = 3] is running, variable: 37
```

5 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?

   一段非匿名虚拟内存区域不一定全部内容都来自文件，`vm_content_size_in_file` 记录的是该段内存区域来自文件的内容长度。具体的，该段内存的前 `vm_content_size_in_file` 字节来自文件，而后面的区域则要被清零。这个字段对应的实际上是 **elf** 中的 `p_filesz`。在对于非匿名区域处理缺页异常时，我们需要特别计算新映射的页的哪些部分要从文件读取，而哪些内容需要清零。

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?

   一段非匿名虚拟内存区域不一定全部内容都来自文件，`vm_content_size_in_file` 记录的是该段内存区域来自文件的内容长度。具体的，该段内存的前 `vm_content_size_in_file` 字节来自文件，而后面的区域则要被清零。这个字段对应的实际上是 **elf** 中的 `p_filesz`。在对于非匿名区域处理缺页异常时，我们需要特别计算新映射的页的哪些部分要从文件读取，而哪些内容需要清零。