操作系统lab5：RV64 缺页异常处理与fork机制

教师：李环、柳晴

学号：3220104519

姓名：蔡佳伟

一、实验目的

- 通过 **vm_area_struct** 数据结构实现对进程**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**
- 为进程加入 **fork** 机制，能够支持通过 **fork** 创建新的用户态进程

二、实验过程

2.1 准备工程

从仓库同步 `user/main.c` 文件并删除原来的 `getpid.c`

修改 `user/Makefile`：

```
user > Makefile
ASM_SRC      = $(filter-out uapp.S, $(sort $(wildcard *.S)))
C_SRC        = $(sort $(wildcard *.c))
OBJ          = $(patsubst %.S,%.o,$(ASM_SRC)) $(patsubst %.c,%.o,$(C_SRC))
TEST         = PFH1
CFLAG        = -march=$(ISA) -mabi=$(ABI) -mcmodel=medany -fno-builtin -ffunction-sections -fdat

```

补充SYS_CLONE的宏，不然会一直显示报错的，虽然在fork阶段才用到

```
arch > riscv > include > C syscall.h > ...
#ifndef __SYSCALL_H__
#define __SYSCALL_H__
#include "stdint.h"
#include "stddef.h"
#include "defs.h"

#define SYS_WRITE 64
#define SYS_GETPID 172
#define SYS_CLONE 220

int sys_write(unsigned int fd, const char *buf, size_t count);
int sys_getpid();
uint64_t do_fork(struct pt_regs *regs);
#endif
```

2.2 缺页异常处理

2.2.1 实现虚拟内存管理功能

先添加vma的定义与数据结构

arch > riscv > include > defs.h > VM_READ

```c
#ifndef __DEFS_H__
#define __DEFS_H__

#include "stdint.h"

// lab5 vma
#define VM_ANON 0x1
#define VM_READ 0x2
#define VM_WRITE 0x4
#define VM_EXEC 0x8
```

arch > riscv > include > proc.h > ...

```c
#ifndef __PROC_H__
#define PRIORITY_MAX 10

#define task_struct_ptr struct task_struct *

// lab 5
struct vm_area_struct
{
    struct mm_struct *vm_mm;                     // 所属的 mm_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
    struct vm_area_struct *vm_next, *vm_prev;    // 链表指针
    uint64_t vm_flags;                           // VMA 对应的 flags
    // struct file *vm_file;    // 对应的文件（目前还没实现，而且我们只有一个 uapp 所以暂不需要）
    uint64_t vm_pgoff;  // 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
    uint64_t vm_filesz; // 对应的文件内容的长度
};

struct mm_struct
{
    struct vm_area_struct *mmap;
};
```

```c
/* 线程状态段数据结构 */
struct thread_struct
{
    uint64_t ra;                        // 32
    uint64_t sp;                        // 40
    uint64_t s[12];                     // 48
    uint64_t sepc, sstatus, sscratch;   // 144 152 160
};

/* 线程数据结构 */
struct task_struct
{
    uint64_t state;    // 线程状态 0
    uint64_t counter;  // 运行剩余时间 8
    uint64_t priority; // 运行优先级 1 最低 10 最高，16
    uint64_t pid;      // 线程 id 24

    struct thread_struct thread;
    uint64_t *pgd;         // 用户态页表 168
    struct mm_struct mm; // 176
};
```

每一个 `vm_area_struct` 都对应于 task 地址空间的唯一**连续**区间。

为了支持 demand paging，我们需要支持对 `vm_area_struct` 的添加和查找：

find_vma 函数：实现对vm_area_struct的查找

- 根据传入的地址 `addr`，遍历链表 `mm` 包含的 VMA 链表，找到该地址所在的 `vm_area_struct`
- 如果链表中所有的 `vm_area_struct` 都不包含该地址，则返回 `NULL`

```
// lab 5 function
struct vm_area_struct *find_vma(struct mm_struct *mm, uint64_t addr)
{
    // 遍历查找
    struct vm_area_struct *vma = mm->mmap;
    while (vma)
    {
        if (vma->vm_start <= addr && addr < vma->vm_end)
            return vma;
        vma = vma->vm_next;
    }
    return NULL;
}
```

`do_mmap` 函数：实现 `vm_area_struct` 的添加

- 新建 `vm_area_struct` 结构体，根据传入的参数对结构体赋值，并添加到 `mm` 指向的 VMA 链表中

```
uint64_t do_mmap(struct mm_struct *mm, uint64_t addr, uint64_t len, uint64_t vm_pgoff, uint64_t vm_filesz, uint64_t flags)
{
    // 插入到链表头
    struct vm_area_struct *mmap = (struct vm_area_struct *)kalloc();
    mmap->vm_mm = mm;
    mmap->vm_start = addr;
    mmap->vm_end = addr + len;
    mmap->vm_flags = flags;
    mmap->vm_pgoff = vm_pgoff;
    mmap->vm_filesz = vm_filesz;
    mmap->vm_prev = NULL;
    if (mm->mmap != NULL)
        mm->mmap->vm_prev = mmap;
    mmap->vm_next = mm->mmap;
    mm->mmap = mmap;
    return addr;
}
```

2.2.2 修改task_init

本次注释掉之前实验对用户栈，代码load segment的映射操作（alloc和create_mapping).

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
- 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（`VM_ANON`）

```
void load_program(struct task_struct *task)
{
    INFO("\ntask init - load program");
    Elf64_Ehdr *ehdr = (Elf64_Ehdr *)_sramdisk;
    Elf64_Phdr *phdrs = (Elf64_Phdr *)(_sramdisk + ehdr->e_phoff);

    // uint64_t uapp_size = _eramdisk - _sramdisk;
    // int page_num = ((uapp_size + PGSIZE + 1) / PGSIZE);
    // DEBUG("uapp page size: %llx page_num: %llx", uapp_size, page_num);
    // char *new_uapp = (char *)alloc_pages(page_num);
    printk("enter load_program, e_phnum = %d\n", ehdr->e_phnum);
    for (int i = 0; i < ehdr->e_phnum; ++i)
```

```c
    {
        Elf64_Phdr *phdr = phdrs + i;
        printk("i = %d, type = %d\n",i, phdr->p_type);
        if (phdr->p_type == PT_LOAD)
        {
            // alloc space and copy content
            uint64_t offset = PGOFFSET(phdr->p_vaddr);
            uint64_t size = offset + phdr->p_memsz;
            uint64_t perm = 0x0; // 初始化
            perm = (phdr->p_flags & PF_X) ? perm | PTE_X : perm;
            perm = (phdr->p_flags & PF_W) ? perm | PTE_W : perm;
            perm = (phdr->p_flags & PF_R) ? perm | PTE_R : perm;
            DEBUG("perm: %llx", perm);
            // char *va = (char *)alloc_pages((size + PGSIZE - 1) / PGSIZE);
            DEBUG("size : %llx", phdr->p_memsz + offset);
            DEBUG("page number : %llx , offset number : %llx", (size + PGSIZE -
1) / PGSIZE, offset);
            // segment 使用的内存就是 [p_vaddr, p_vaddr + p_memsz) 这一连续区间，然后
将 segment 的内容从 ELF 文件中读入到这一内存区间
            // 为了实现对齐，实际上 0 - offset之间的为无效内容
            // for (int i = 0; i < phdr->p_filesz; i++)
            // va[offset + i] = ((char *)ehdr)[phdr->p_offset + i];
            // DEBUG("va : %llx", va);
            // 将 [p_vaddr + p_filesz, p_vaddr + p_memsz) 对应的 *物理区间* 清零
            // memset(va + offset + phdr->p_filesz, 0, phdr->p_memsz - phdr-
>p_filesz);
            // DEBUG("va: %llx, pa: %llx, size: %llx, perm: %llx", va,
(uint64_t)va - PA2VA_OFFSET, phdr->p_memsz + offset, perm | PTE_U | PTE_V);
            // create_mapping(task->pgd, phdr->p_vaddr, (uint64_t)va -
PA2VA_OFFSET, size, perm | PTE_U | PTE_V);

            // lab5
            do_mmap(&task->mm, phdr->p_vaddr, phdr->p_memsz, phdr->p_offset,
phdr->p_filesz, VM_READ | VM_WRITE | VM_EXEC);
        }
    }
    // DEBUG("user_start: %llx", USER_START);
    // uint64_t user_stack = (uint64_t)alloc_page();
    // DEBUG("user_stack: %llx", user_stack);
    // create_mapping(task->pgd, USER_END - PGSIZE, user_stack - PA2VA_OFFSET,
PGSIZE, PTE_U | PTE_V | PTE_R | PTE_W);
    do_mmap(&task->mm, USER_END - PGSIZE, PGSIZE, 0, 0, VM_ANON | VM_READ |
VM_WRITE);
    task->thread.sepc = ehdr->e_entry;
}
```

task_init函数也需要修改，修改如下：

```c
void task_init()
{
    INFO("task init\n");
    srand(2024);
    // 1. 调用 kalloc() 为 idle 分配一个物理页
    // 2. 设置 state 为 TASK_RUNNING;
    // 3. 由于 idle 不参与调度，可以将其 counter / priority 设置为 0
    // 4. 设置 idle 的 pid 为 0
    // 5. 将 current 和 task[0] 指向 idle
    idle = (struct task_struct *)kalloc();
    if (idle == NULL)
        printk(RED "Error : kalloc() failed in task_init()\n" CLEAR);

    idle->state = TASK_RUNNING;
    idle->counter = 0;
    idle->priority = 0;
    idle->pid = 0;

    current = idle;
    task[0] = idle;

    // 1. 参考 idle 的设置，为 task[1] ~ task[NR_TASKS - 1] 进行初始化
    // 2. 其中每个线程的 state 为 TASK_RUNNING，此外，counter 和 priority 进行如下赋值：
    //     - counter  = 0;
    //     - priority = rand() 产生的随机数（控制范围在 [PRIORITY_MIN, PRIORITY_MAX] 之间）
    // 3. 为 task[1] ~ task[NR_TASKS - 1] 设置 thread_struct 中的 ra 和 sp
    //     - ra 设置为 __dummy（见 4.2.2）的地址
    //     - sp 设置为该线程申请的物理页的高地址

    printk("_sramdisk: %llx, _eramdisk: %llx, _sbss: %llx\n",(uint64_t)_sramdisk,(uint64_t)_eramdisk,(uint64_t)_sbss);
    for (int i = 1; i < nr_tasks; i++)
    {
        task[i] = (struct task_struct *)kalloc();
        task[i]->state = TASK_RUNNING;
        task[i]->counter = 0;
        task[i]->priority = PRIORITY_MIN + rand() % (PRIORITY_MAX - PRIORITY_MIN + 1);
        task[i]->pid = i;
        task[i]->thread.ra = (uint64_t)__dummy;
        task[i]->thread.sp = (uint64_t)task[i] + PGSIZE; // 内核态的栈

        // task[i]->thread.sepc = (USER_START);  // in loadprogram
        task[i]->pgd = (uint64_t *)alloc_page();
        // 将内核页表 swapper_pg_dir 复制到每个进程的页表中
        for (int j = 0; j < 512; ++j)
            task[i]->pgd[j] = swapper_pg_dir[j];

        // set_task_pgd(task[i]);
        load_program(task[i]);
        task[i]->thread.sstatus = ((1 << 18) | (1 << 5)); // SUM SPIE
        task[i]->thread.sscratch = (USER_END);

        printk("task[%d] pid = %d priority = %d\n", i, task[i]->pid, task[i]->priority);
    }

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

2.3 实现page fault handler

修改 `trap.c`，为 `trap_handler` 添加捕获 page fault 的逻辑，分别需要捕获 12, 13, 15 号异常。

```c
// 中断和异常处理函数
void trap_handler(uint64_t scause, uint64_t sepc, struct pt_regs *regs)
{
    uint64_t stval = regs->stval;  // 错误发生时的值
    if (scause & 0x8000000000000000)
    {
        // 处理中断
        uint64_t interrupt_code = scause & 0x7FFFFFFFFFFFFFFF;

        switch (interrupt_code)
        {
        case 5:
            clock_set_next_event();  // 设置下一次时钟中断
            do_timer();  // 处理定时器中断
            break;
        default:
            Err("Unknown interrupt\n");  // 未知中断
            break;
        }
    }
    else
    {
        // 处理异常
        uint64_t exception_code = scause & 0x7FFFFFFFFFFFFFFF;
        switch (exception_code)
        {
        case 8:
            // 处理用户态系统调用
            int sys_call_num = regs->x[17];  // 获取系统调用编号
            switch (sys_call_num)
            {
            case SYS_WRITE:
                sys_write((unsigned int)regs->x[10], (const char *)regs->x[11], (size_t)regs->x[12]);  // 处理write系统调用
                break;
            case SYS_GETPID:
                regs->x[10] = sys_getpid();  // 处理getpid系统调用
                break;
```

```c
            case SYS_CLONE:
                regs->x[10] = do_fork(regs);  // 处理fork系统调用
                break;
            default:
                break;
            }
            regs->sepc += 4;  // 更新程序计数器，跳过系统调用指令
            break;
        case 12:  // 处理加载页面错误
            do_page_fault(regs, 12);
            break;
        case 13:  // 处理读取页面错误
            do_page_fault(regs, 13);
            break;
        case 15:  // 处理写入页面错误
            do_page_fault(regs, 15);
            break;
        default:
            break;
        }
    }
}
```

实现缺页异常的函数 `do_page_fault`，具体逻辑如下：

1. 通过 `stval` 获得访问出错的虚拟内存地址（Bad Address）
2. 通过 find_vma() 查找 bad address 是否在某个vma 中
   - 如果不在，则出现非预期错误，可以通过 `Err` 宏输出错误信息
   - 如果在，则根据vma 的 flags 权限判断当前page fault 是否合法
     - 如果非法（比如触发的是 instruction page fault 但 vma 权限不允许执行），则 `Err` 输出错误信息
     - 其他情况合法，需要我们按接下来的流程创建映射
3. 分配一个页，接下来要将这个页映射到对应的用户地址空间
4. 通过(vma->vm_flags & VM_ANON) 获得当前的 VMA 是否是匿名空间
   - 如果是匿名空间，则直接映射即可
   - 如果不是，则需要根据 `vma->vm_pgoff` 等信息从 ELF 中读取数据，填充后映射到用户空间

```c
// 处理页错误（页异常）
void do_page_fault(struct pt_regs *regs, uint64_t exception_code)
{
    Log("[PID = %d PC = %lx] valid page fault at `%lx` with cause %d", current->pid, regs->sepc, regs->stval, exception_code);

    uint64_t bad_addr = regs->stval;  // 错误地址
    if (current->mm.mmap == NULL)
        Err("current task`s mmap is NULL");  // 如果当前任务没有内存映射，报错

    struct vm_area_struct *vma = find_vma(&current->mm, bad_addr);  // 查找错误地址所在的虚拟内存区域（VMA）
    if (vma == NULL)
        Err("No vma found for va: %lx with exception: %d , sepc: %lx\n", bad_addr, exception_code, regs->sepc);  // 如果没有找到相应的VMA，报错

    // 如果访问的内存区域没有读权限，则报错
    if (vma->vm_flags & VM_READ == 0)
        Err("vma perm err without VM_READ: va: %lx\n", bad_addr);

    // 根据异常代码处理不同的情况
    switch (exception_code)
    {
    case 12:  // 执行错误
        if (vma->vm_flags & VM_EXEC == 0)  // 如果VMA没有执行权限，报错
            Err("vma perm err without VM_EXEC: va: %lx\n", bad_addr);
        break;
    case 13: // 加载错误（读取内存）
        break;
    case 15: // 存储错误（写入内存）
        if (vma->vm_flags & VM_WRITE == 0)  // 如果VMA没有写权限，报错
            Err("vma perm err without VM_WRITE: va: %lx\n", bad_addr);
        break;
    }

    // 创建新的内存页映射
    uint64_t *page = (uint64_t *)alloc_page();  // 分配一页新的内存
    memset((void *)page, 0, PGSIZE);  // 将新分配的内存页清零
```

```c
    uint64_t perm = (vma->vm_flags & 0b01110) | 0b10001;  // 设置内存权限

    if (vma->vm_flags & VM_ANON)  // 如果是匿名映射，直接创建映射
        create_mapping(current->pgd, PGROUNDDOWN(bad_addr), (uint64_t)page - PA2VA_OFFSET, PGSIZE, perm);
    else
    {
        // 如果是文件映射，填充数据
        uint64_t mem_size = vma->vm_end - vma->vm_start;  // 计算内存区域大小

        uint64_t global_part_start = vma->vm_start + vma->vm_filesz;  // 文件数据起始位置
        uint64_t global_part_end = vma->vm_end;  // 文件数据结束位置
        uint64_t bad_seg_start = vma->vm_pgoff + (uint64_t)_sramdisk;  // 文件段起始位置
        uint64_t bad_seg_now = bad_seg_start + PGROUNDDOWN(bad_addr) - vma->vm_start;  // 当前映射段的起始位置
        uint64_t size = PGSIZE; // 要映射的大小
        uint64_t offset;    // 页内的偏移

        if (bad_addr > vma->vm_start + vma->vm_filesz)
        {
            offset = 0;
            size = PGSIZE;
        }

        // 如果地址小于文件映射区域的起始地址，需要处理并拷贝数据
        if (PGROUNDUP(vma->vm_start) > bad_addr)
        {
            offset = PGOFFSET(bad_seg_start);  // 计算偏移量
            size = PGSIZE;
            memcpy((void *)(page), (void *)(bad_seg_now), size);  // 将文件数据拷贝到新分配的页中
        }
        // 处理其他边界情况
        else if (PGROUNDDOWN(vma->vm_start + vma->vm_filesz) == PGROUNDDOWN(bad_addr))
        {
            offset = 0;
            memcpy((void *)(page + offset / 8), (void *)(bad_seg_now), PGOFFSET(vma->vm_start + vma->vm_filesz));
        }
        else if (PGROUNDDOWN(vma->vm_end) == PGROUNDDOWN(bad_addr))
```

```c
        {
            size = PGOFFSET(vma->vm_end);
        }

        // 映射内存页面
        create_mapping(current->pgd, PGROUNDDOWN(bad_addr), (uint64_t)page - PA2VA_OFFSET, size, PTE_U | PTE_V | perm);
    }
}
```

2.4 测试缺页处理

使用`make run TEST=PFH1`在task_init中并未进行映射，直到page fault触发用户态进程的拷贝和映射，且只有第一次触发

```
[INFO] setup vm starting
[INFO] setup vm done
...buddy_init done!
...mm_init done!
[INFO] setup_vm_final starting
[vm.c,125,create_mapping] root : ffffffe00020c000, [80200000, 80204000) -> [ffffffe000200000, ffffffe000204000), perm: b
[vm.c,125,create_mapping] root : ffffffe00020c000, [80204000, 80206000) -> [ffffffe000204000, ffffffe000206000), perm: 3
[vm.c,125,create_mapping] root : ffffffe00020c000, [80206000, 88000000) -> [ffffffe000206000, ffffffe008000000), perm: 7
[DEBUG] satp_val: 800000000008020C
[INFO] setup_vm_final done
[INFO] task init

_sramdisk: ffffffe000207000, _eramdisk: ffffffe000208c50, _sbss: ffffffe000209000
[INFO]
task init - load program
enter load_program, e_phnum = 3
i = 0, type = 1879048195
i = 1, type = 1
[DEBUG] perm: e
[DEBUG] size : 23f0
[DEBUG] page number : 3 , offset number : e8
i = 2, type = 1685382481
task[1] pid = 1 priority = 7
...task_init done!
2024 ZJU Operating System
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
switch from 0 to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,82,do_page_fault] [PID = 1 PC = 100e8] valid page fault at `100e8` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d2000, 802d3000) -> [10000, 11000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 10178] valid page fault at `3ffffffff8` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d5000, 802d6000) -> [3ffffff000, 4000000000), perm: 17
[trap.c,82,do_page_fault] [PID = 1 PC = 10198] valid page fault at `12000` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d8000, 802d83f0) -> [12000, 123f0), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 110ac] valid page fault at `110ac` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d9000, 802da000) -> [11000, 12000), perm: 1f
[U-MODE] pid: 1, sp is 0x3fffffffe0, this is print No.1
[U-MODE] pid: 1, sp is 0x3fffffffe0, this is print No.2
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
```

使用`make run TEST=PFH2` 类似

```
...buddy_init done!
...mm_init done!
[INFO] setup_vm_final starting
[vm.c,125,create_mapping] root : ffffffe00020c000, [80200000, 80204000) -> [ffffffe000200000, ffffffe000204000), perm: b
[vm.c,125,create_mapping] root : ffffffe00020c000, [80204000, 80206000) -> [ffffffe000204000, ffffffe000206000), perm: 3
[vm.c,125,create_mapping] root : ffffffe00020c000, [80206000, 88000000) -> [ffffffe000206000, ffffffe008000000), perm: 7
[DEBUG] satp_val: 800000000008020C
[INFO] setup_vm_final done
[INFO] task init

_sramdisk: ffffffe000207000, _eramdisk: ffffffe000208c48, _sbss: ffffffe000209000
[INFO]
task init - load program
enter load_program, e_phnum = 3
i = 0, type = 1879048195
i = 1, type = 1
[DEBUG] perm: e
[DEBUG] size : 33f8
[DEBUG] page number : 4 , offset number : e8
i = 2, type = 1685382481
task[1] pid = 1 priority = 7
...task_init done!
2024 ZJU Operating System
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
switch from 0 to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,82,do_page_fault] [PID = 1 PC = 100e8] valid page fault at `100e8` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d2000, 802d3000) -> [10000, 11000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 10178] valid page fault at `3ffffffff8` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d5000, 802d6000) -> [3ffffff000, 4000000000), perm: 17
[trap.c,82,do_page_fault] [PID = 1 PC = 10194] valid page fault at `13000` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d8000, 802d83f8) -> [13000, 133f8), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 11080] valid page fault at `11080` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d9000, 802da000) -> [11000, 12000), perm: 1f
[U-MODE] pid: 1, increment: 0
[U-MODE] pid: 1, increment: 1
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
[U-MODE] pid: 1, increment: 2
[U-MODE] pid: 1, increment: 3
```

3.1 实现fork系统调用 准备工作

修改proc.c相关代码，使得其只初始化一个进程，其他进程保留为 NULL 等待 fork 创建

- 定义nr_tasks记录当前进程数并作为栈顶指针
- task_init：只初始化一个进程,将nr_tasks置为2指向栈顶
- Schedule:将NR_TASKS替换为nr_tasks

```c
    for (int i = 1; i < nr_tasks; i++)
    {
        task[i] = (struct task_struct *)kalloc();
        task[i]->state = TASK_RUNNING;
        task[i]->counter = 0;
        task[i]->priority = PRIORITY_MIN + rand() % (PRIORITY_MAX - PRIORITY_MIN + 1);
```

```
// lab 5 fork
// 记录当前进程数，并作为 tasks 的栈顶指针来使用
uint64_t nr_tasks = 2;
```

添加系统调用处理

在syscall.h加入#define SYS_CLONE 220(之前已经加入了)

在trap_handler调用do_fork，处理regs->a7 == SYS_CLONE的情况

```
case 8:
    // 处理用户态系统调用
    int sys_call_num = regs->x[17];  // 获取系统调用编号
    switch (sys_call_num)
    {
    case SYS_WRITE:
        sys_write((unsigned int)regs->x[10], (const char *)regs->x[11], (size_t)regs->x[12]);  // 处理write系统调用
        break;
    case SYS_GETPID:
        regs->x[10] = sys_getpid();  // 处理getpid系统调用
        break;
    case SYS_CLONE:
        regs->x[10] = do_fork(regs);  // 处理fork系统调用
        break;
    default:
        break;
    }
    regs->sepc += 4;  // 更新程序计数器，跳过系统调用指令
    break;
```

3.2 do_fork实现

fork的实现：

- 创建一个新进程：
  - 拷贝内核栈（包括了 `task_struct` 等信息）
  - 创建一个新的页表
    - 拷贝内核页表 `swapper_pg_dir`
    - 遍历父进程 vma，并遍历父进程页表
      - 将这个 vma 也添加到新进程的 vma 链表中
      - 如果该 vma 项有对应的页表项存在（说明已经创建了映射），则需要深拷贝一整页的内容并映射到新页表中

```
uint64_t do_fork(struct pt_regs *regs)
{
    Log("current: thread sp : %lx, thread sscratch : %lx", current->thread.sp, current->thread.sscratch);
    Log("sp : %lx", (uint64_t)regs);

    struct task_struct * _task = (struct task_struct *)alloc_page();  // 为新进程分配一页内存
    uint64_t cur_sscratch = csr_read(sscratch);  // 获取当前的sscratch寄存器值
    Log("cur_sscratch : %lx", cur_sscratch);

    // 复制父进程的内存映射结构
    memcpy((void *)_task, (void *)current, PGSIZE);
    _task->pid = nr_tasks;  // 设置新进程的PID
    nr_tasks++;  // 增加全局任务计数

    INFO("[PID = %d] forked from [PID = %d]", _task->pid, current->pid);  // 记录新进程创建信息

    if (_task->pid >= NR_TASKS)
        Err("task number exceed");  // 如果任务数量超过上限，报错

    task[_task->pid] = _task;  // 将新进程添加到任务数组中
    _task->mm.mmap = NULL;  // 清空新进程的内存映射结构

    // 设置新进程的返回地址和栈指针
    _task->thread.ra = (uint64_t)__ret_from_fork;
    _task->thread.sp = (uint64_t)_task + PGSIZE - (35 * 8);  // 设置新进程的内核栈位置
    _task->thread.sscratch = cur_sscratch;  // 设置新进程的sscratch寄存器

    // 创建新进程的页表
    _task->pgd = (uint64_t *)alloc_page();
    memcpy((void *)_task->pgd, (void *)swapper_pg_dir, PGSIZE);  // 将内核页表拷贝到新进程的页表中
```

- 将新进程加入调度队列
- 处理父子进程的返回值
  - 父进程通过 `do_fork` 函数直接返回子进程的 pid，并回到自身运行
  - 子进程通过被调度器调度后（跳到 `thread.ra`），开始执行并返回 0

```c

    // 遍历父进程的虚拟内存区域（VMA）
    struct vm_area_struct *pvma = current->mm.mmap;
    for (; pvma != NULL; pvma = pvma->vm_next)
    {
        // 根据父进程的VMA信息，映射相应的内存区域到新进程
        do_mmap(&_task->mm, pvma->vm_start, pvma->vm_end - pvma->vm_start, pvma->vm_pgoff, pvma->vm_filesz, pvma->vm_flags);

        // 遍历VMA中的每一页，复制父进程的物理页到新进程
        for (uint64_t va = PGROUNDDOWN(pvma->vm_start); va < pvma->vm_end; va += PGSIZE)
        {
            uint64_t pte_res = findall_pgt(current->pgd, va);  // 获取父进程页表项
            uint64_t pa = PTE2PA(pte_res);  // 获取物理地址
            if (pte_res == 0)
                continue;
            uint64_t *page = (uint64_t *)alloc_page();  // 为新页分配内存
            memcpy((void *)page, (void *)(pa + PA2VA_OFFSET), PGSIZE);  // 复制父进程的页到新进程

            Log("copy page : %lx -> %lx", (uint64_t)(pa), (uint64_t)page);
            create_mapping(_task->pgd, va, (uint64_t)page - PA2VA_OFFSET, PGSIZE, PTE_U | PTE_V | pvma->vm_flags);  // 创建新进程的页表映射
        }
    }

    // 更新新进程的寄存器状态，设置正确的返回值和SEPC
    struct pt_regs *new_regs = (struct pt_regs *)((PGOFFSET((uint64_t)regs) + (uint64_t)_task));
    memcpy((void *)new_regs, (void *)regs, sizeof(struct pt_regs));
    new_regs->x[10] = 0;  // 设置a0寄存器为0
    new_regs->sepc = new_regs->sepc + 4;  // 更新SEPC，跳过fork指令

    return _task->pid;  // 返回新进程的PID
}
```

3.3 处理进程返回逻辑

修改entry.S如下：

```asm
    sd t0, 272(sp)

    # 2. call trap_handler
    csrr a0, scause
    csrr a1, sepc
    addi a2, sp, 0
    call trap_handler

# lab5 fork 后的进程返回地址
    .globl __ret_from_fork

__ret_from_fork:
    # 3. restore sepc and 32 registers (x2(sp)) should be restore last) from stack
    ld t0, 272(sp)
```

此部分还有设置do_fork的父进程、子进程的用户栈和内核栈指针部分，在前面已经展示

3.4 测试fork

make run TEST=FORK1

fork时为已映射页拷贝并映射,fork出的子进程与父进程global_variable相互独立

fork在PID=2 forked from PID=1后调用完cause = 1f的 create_mapping 后完成

```
...buddy_init done!
...mm_init done!
[INFO] setup_vm_final starting
[vm.c,125,create_mapping] root : ffffffe00020c000, [80200000, 80204000) -> [ffffffe000200000, ffffffe000204000), perm: b
[vm.c,125,create_mapping] root : ffffffe00020c000, [80204000, 80206000) -> [ffffffe000204000, ffffffe000206000), perm: 3
[vm.c,125,create_mapping] root : ffffffe00020c000, [80206000, 88000000) -> [ffffffe000206000, ffffffe008000000), perm: 7
[DEBUG] satp_val: 800000000008020C
[INFO] setup_vm_final done
[INFO] task init

_sramdisk: ffffffe000207000, _eramdisk: ffffffe000208d50, _sbss: ffffffe000209000
[INFO]
task init - load program
enter load_program, e_phnum = 3
i = 0, type = 1879048195
i = 1, type = 1
[DEBUG] perm: e
[DEBUG] size : 23f0
[DEBUG] page number : 3 , offset number : e8
i = 2, type = 1685382481
task[1] pid = 1 priority = 7
...task_init done!
2024 ZJU Operating System
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
switch from 0 to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,82,do_page_fault] [PID = 1 PC = 100e8] valid page fault at `100e8` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d2000, 802d3000) -> [10000, 11000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 101ac] valid page fault at `3ffffffff8` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d5000, 802d6000) -> [3fffffff000, 4000000000), perm: 17
[syscall.c,41,do_fork] current: thread sp : ffffffe0002cf000, thread sscratch : 4000000000
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffc0
[INFO] [PID = 2] forked from [PID = 1]
[syscall.c,87,do_fork] copy page : 802d5000 -> ffffffe0002db000
[vm.c,125,create_mapping] root : ffffffe0002d9000, [802db000, 802dc000) -> [3fffffff000, 4000000000), perm: 17
[syscall.c,87,do_fork] copy page : 802d2000 -> ffffffe0002df000
[vm.c,125,create_mapping] root : ffffffe0002d9000, [802df000, 802e0000) -> [10000, 11000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 1022c] valid page fault at `12000` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802e2000, 802e23f0) -> [12000, 123f0), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 11118] valid page fault at `11118` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802e3000, 802e4000) -> [11000, 12000), perm: 1f
[U-PARENT] pid: 1 is running! global_variable: 0
```

make run TEST=FORK2

fork出PID=2的子进程实现深拷贝,正确输出父进程改变的字符串及变量值,且后续与父进程独立.

```
...buddy_init done!
...mm_init done!
[INFO] setup_vm_final starting
[vm.c,125,create_mapping] root : ffffffe00020c000, [80200000, 80204000) -> [ffffffe000200000, ffffffe000204000), perm: b
[vm.c,125,create_mapping] root : ffffffe00020c000, [80204000, 80206000) -> [ffffffe000204000, ffffffe000206000), perm: 3
[vm.c,125,create_mapping] root : ffffffe00020c000, [80206000, 88000000) -> [ffffffe000206000, ffffffe008000000), perm: 7
[DEBUG] satp_val: 800000000008020C
[INFO] setup_vm_final done
[INFO] task init

_sramdisk: ffffffe000207000, _eramdisk: ffffffe000208fc0, _sbss: ffffffe000209000
[INFO]
task init - load program
enter load_program, e_phnum = 3
i = 0, type = 1879048195
i = 1, type = 1
[DEBUG] perm: e
[DEBUG] size : 43f8
[DEBUG] page number : 5 , offset number : e8
i = 2, type = 1685382481
task[1] pid = 1 priority = 7
...task_init done!
2024 ZJU Operating System
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
switch from 0 to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,82,do_page_fault] [PID = 1 PC = 100e8] valid page fault at `100e8` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d2000, 802d3000) -> [10000, 11000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 101ac] valid page fault at `3ffffffff8` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d5000, 802d6000) -> [3fffffff000, 4000000000), perm: 17
[trap.c,82,do_page_fault] [PID = 1 PC = 101d0] valid page fault at `12000` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d8000, 802d9000) -> [12000, 13000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 112d4] valid page fault at `112d4` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d9000, 802da000) -> [11000, 12000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 1043c] valid page fault at `14008` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802da000, 802da3f8) -> [14000, 143f8), perm: 1f
[U] pid: 1 is running! global_variable: 0
[U] pid: 1 is running! global_variable: 1
[U] pid: 1 is running! global_variable: 2
[trap.c,82,do_page_fault] [PID = 1 PC = 10228] valid page fault at `13008` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802db000, 802dc000) -> [13000, 14000), perm: 1f
```

```
[syscall.c,41,do_fork] current: thread sp : ffffffe0002cf000, thread sscratch : 4000000000
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffc0
[INFO] [PID = 2] forked from [PID = 1]
[syscall.c,87,do_fork] copy page : 802d5000 -> ffffffe0002df000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802df000, 802e0000) -> [3ffffff000, 4000000000), perm: 17
[syscall.c,87,do_fork] copy page : 802d2000 -> ffffffe0002e3000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e3000, 802e4000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d9000 -> ffffffe0002e6000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e6000, 802e7000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d8000 -> ffffffe0002e7000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e7000, 802e8000) -> [12000, 13000), perm: 1f
[syscall.c,87,do_fork] copy page : 802db000 -> ffffffe0002e8000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e8000, 802e9000) -> [13000, 14000), perm: 1f
[syscall.c,87,do_fork] copy page : 802da000 -> ffffffe0002e9000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e9000, 802ea000) -> [14000, 15000), perm: 1f
[U-PARENT] pid: 1 is running! Message: ZJU OS Lab5
[U-PARENT] pid: 1 is running! global_variable: 3
[U-PARENT] pid: 1 is running! global_variable: 4
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 6, priority = 7
switch from 1 to [PID = 2 PRIORITY = 7 COUNTER = 6]
[U-CHILD] pid: 2 is running! Message: ZJU OS Lab5
[U-CHILD] pid: 2 is running! global_variable: 3
[U-CHILD] pid: 2 is running! global_variable: 4
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
pid = 2, counter = 7, priority = 7
switch from 2 to [PID = 1 PRIORITY = 7 COUNTER = 7]
[U-CHILD] pid: 1 is running! global_variable: 5
[U-CHILD] pid: 1 is running! global_variable: 6
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 7, priority = 7
switch from 1 to [PID = 2 PRIORITY = 7 COUNTER = 7]
```

make run TEST=FORK3

进程能够正确实现多层fork(),并且子进程正确复制父进程global_variable值,后续父子进程相互独立

```
...buddy_init done!
...mm_init done!
[INFO] setup_vm_final starting
[vm.c,125,create_mapping] root : ffffffe00020c000, [80200000, 80204000) -> [ffffffe000200000, ffffffe000204000), perm: b
[vm.c,125,create_mapping] root : ffffffe00020c000, [80204000, 80206000) -> [ffffffe000204000, ffffffe000206000), perm: 3
[vm.c,125,create_mapping] root : ffffffe00020c000, [80206000, 88000000) -> [ffffffe000206000, ffffffe008000000), perm: 7
[DEBUG] satp_val: 800000000008020C
[INFO] setup_vm_final done
[INFO] task init

_sramdisk: ffffffe000207000, _eramdisk: ffffffe000208d28, _sbss: ffffffe000209000
[INFO]
task init - load program
enter load_program, e_phnum = 3
i = 0, type = 1879048195
i = 1, type = 1
[DEBUG] perm: e
[DEBUG] size : 23f0
[DEBUG] page number : 3 , offset number : e8
i = 2, type = 1685382481
task[1] pid = 1 priority = 7
...task_init done!
2024 ZJU Operating System
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
switch from 0 to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,82,do_page_fault] [PID = 1 PC = 100e8] valid page fault at `100e8` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d2000, 802d3000) -> [10000, 11000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 101ac] valid page fault at `3ffffffff8` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d5000, 802d6000) -> [3ffffff000, 4000000000), perm: 17
[trap.c,82,do_page_fault] [PID = 1 PC = 101c8] valid page fault at `12000` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d8000, 802d83f0) -> [12000, 123f0), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 11134] valid page fault at `11134` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d9000, 802da000) -> [11000, 12000), perm: 1f
[U] pid: 1 is running! global_variable: 0
[syscall.c,41,do_fork] current: thread sp : ffffffe0002cf000, thread sscratch : 4000000000
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 2] forked from [PID = 1]
[syscall.c,87,do_fork] copy page : 802d5000 -> ffffffe0002dd000
[vm.c,125,create_mapping] root : ffffffe0002db000, [802dd000, 802de000) -> [3ffffff000, 4000000000), perm: 17
[syscall.c,87,do_fork] copy page : 802d2000 -> ffffffe0002e1000
```

```
[vm.c,125,create_mapping] root : ffffffe0002db000, [802e1000, 802e2000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d9000 -> ffffffe0002e4000
[vm.c,125,create_mapping] root : ffffffe0002db000, [802e4000, 802e5000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d8000 -> ffffffe0002e5000
[vm.c,125,create_mapping] root : ffffffe0002db000, [802e5000, 802e6000) -> [12000, 13000), perm: 1f
[syscall.c,41,do_fork] current: thread sp : ffffffe0002cf000, thread sscratch : 4000000000
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 3] forked from [PID = 1]
[syscall.c,87,do_fork] copy page : 802d5000 -> ffffffe0002e9000
[vm.c,125,create_mapping] root : ffffffe0002e7000, [802e9000, 802ea000) -> [3ffffff000, 4000000000), perm: 17
[syscall.c,87,do_fork] copy page : 802d2000 -> ffffffe0002ed000
[vm.c,125,create_mapping] root : ffffffe0002e7000, [802ed000, 802ee000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d9000 -> ffffffe0002f0000
[vm.c,125,create_mapping] root : ffffffe0002e7000, [802f0000, 802f1000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d8000 -> ffffffe0002f1000
[vm.c,125,create_mapping] root : ffffffe0002e7000, [802f1000, 802f2000) -> [12000, 13000), perm: 1f
[U] pid: 1 is running! global_variable: 1
[syscall.c,41,do_fork] current: thread sp : ffffffe0002cf000, thread sscratch : 4000000000
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 4] forked from [PID = 1]
[syscall.c,87,do_fork] copy page : 802d5000 -> ffffffe0002f5000
[vm.c,125,create_mapping] root : ffffffe0002f3000, [802f5000, 802f6000) -> [3ffffff000, 4000000000), perm: 17
[syscall.c,87,do_fork] copy page : 802d2000 -> ffffffe0002f9000
[vm.c,125,create_mapping] root : ffffffe0002f3000, [802f9000, 802fa000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d9000 -> ffffffe0002fc000
[vm.c,125,create_mapping] root : ffffffe0002f3000, [802fc000, 802fd000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d8000 -> ffffffe0002fd000
[vm.c,125,create_mapping] root : ffffffe0002f3000, [802fd000, 802fe000) -> [12000, 13000), perm: 1f
[U] pid: 1 is running! global_variable: 2
[U] pid: 1 is running! global_variable: 3
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 6, priority = 7
pid = 3, counter = 6, priority = 7
pid = 4, counter = 6, priority = 7
switch from 1 to [PID = 2 PRIORITY = 7 COUNTER = 6]
[syscall.c,41,do_fork] current: thread sp : ffffffe0002daee8, thread sscratch : 3fffffffd0
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 5] forked from [PID = 2]
[syscall.c,87,do_fork] copy page : 802e1000 -> ffffffe000301000
```

```
[vm.c,125,create_mapping] root : ffffffe0002ff000, [80301000, 80302000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e4000 -> ffffffe000304000
[vm.c,125,create_mapping] root : ffffffe0002ff000, [80304000, 80305000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e5000 -> ffffffe000305000
[vm.c,125,create_mapping] root : ffffffe0002ff000, [80305000, 80306000) -> [12000, 13000), perm: 1f
[syscall.c,87,do_fork] copy page : 802dd000 -> ffffffe000307000
[vm.c,125,create_mapping] root : ffffffe0002ff000, [80307000, 80308000) -> [3ffffff000, 4000000000), perm: 17
[U] pid: 2 is running! global_variable: 1
[syscall.c,41,do_fork] current: thread sp : ffffffe0002daee8, thread sscratch : 3fffffffd0
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 6] forked from [PID = 2]
[syscall.c,87,do_fork] copy page : 802e1000 -> ffffffe00030d000
[vm.c,125,create_mapping] root : ffffffe00030b000, [8030d000, 8030e000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e4000 -> ffffffe000310000
[vm.c,125,create_mapping] root : ffffffe00030b000, [80310000, 80311000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e5000 -> ffffffe000311000
[vm.c,125,create_mapping] root : ffffffe00030b000, [80311000, 80312000) -> [12000, 13000), perm: 1f
[syscall.c,87,do_fork] copy page : 802dd000 -> ffffffe000313000
[vm.c,125,create_mapping] root : ffffffe00030b000, [80313000, 80314000) -> [3ffffff000, 4000000000), perm: 17
[U] pid: 2 is running! global_variable: 2
[U] pid: 2 is running! global_variable: 3
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 0, priority = 7
pid = 3, counter = 6, priority = 7
pid = 4, counter = 6, priority = 7
pid = 5, counter = 6, priority = 7
pid = 6, counter = 6, priority = 7
switch from 2 to [PID = 3 PRIORITY = 7 COUNTER = 6]
[U] pid: 3 is running! global_variable: 1
[syscall.c,41,do_fork] current: thread sp : ffffffe0002e6ee8, thread sscratch : 3fffffffd0
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 7] forked from [PID = 3]
[syscall.c,87,do_fork] copy page : 802ed000 -> ffffffe000319000
[vm.c,125,create_mapping] root : ffffffe000317000, [80319000, 8031a000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802f0000 -> ffffffe00031c000
[vm.c,125,create_mapping] root : ffffffe000317000, [8031c000, 8031d000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802f1000 -> ffffffe00031d000
[vm.c,125,create_mapping] root : ffffffe000317000, [8031d000, 8031e000) -> [12000, 13000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e9000 -> ffffffe00031f000
[vm.c,125,create_mapping] root : ffffffe000317000, [8031f000, 80320000) -> [3ffffff000, 4000000000), perm: 17
```

三、讨论和心得

本次写代码的过程还是比较困难的，战线拉得比较长，细节问题也比较多，很难处理，之前的代码也有很多要注释掉的部分，我对注释掉哪个部分也不算特别清晰，因而踩了一些坑。

还有会有莫名其妙的问题，解决起来更是痛苦，只能通过大改代码来解决。

我在这里设置新进程的内核栈位置错误，没有-35*8，会在切换进程的时候卡死，因为栈指针的位置不对。子线程跳回的地址在_traps的call trap_handler之后，所以要手动调整-35 * 8.

```c
uint64_t do_fork(struct pt_regs *regs)
    uint64_t cur_sscratch = csr_read(sscratch);  // 获取当前的sscratch寄存器值
    Log("cur_sscratch : %lx", cur_sscratch);

    // 复制父进程的内存映射结构
    memcpy((void *)_task, (void *)current, PGSIZE);
    _task->pid = nr_tasks;  // 设置新进程的PID
    nr_tasks++;  // 增加全局任务计数

    INFO("[PID = %d] forked from [PID = %d]", _task->pid, current->pid);  // 记录新进程创建信息

    if (_task->pid >= NR_TASKS)
        Err("task number exceed");  // 如果任务数量超过上限，报错

    task[_task->pid] = _task;  // 将新进程添加到任务数组中
    _task->mm.mmap = NULL;  // 清空新进程的内存映射结构

    // 设置新进程的返回地址和栈指针
    _task->thread.ra = (uint64_t)__ret_from_fork;
    _task->thread.sp = (uint64_t)_task + PGSIZE;    // 设置新进程的内核栈位置
    _task->thread.sscratch = cur_sscratch;  // 设置新进程的sscratch寄存器
```

```
[INFO] [PID = 4] forked from [PID = 1]
[syscall.c,87,do_fork] copy page : 802d5000 -> ffffffe0002f5000
[vm.c,124,create_mapping] root : ffffffe0002f3000, [802f5000, 802f6000) -> [3ffffff000, 4000000000), perm: 17
[syscall.c,87,do_fork] copy page : 802d2000 -> ffffffe0002f9000
[vm.c,124,create_mapping] root : ffffffe0002f3000, [802f9000, 802fa000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d9000 -> ffffffe0002fc000
[vm.c,124,create_mapping] root : ffffffe0002f3000, [802fc000, 802fd000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d8000 -> ffffffe0002fd000
[vm.c,124,create_mapping] root : ffffffe0002f3000, [802fd000, 802fe000) -> [12000, 13000), perm: 1f
[U] pid: 1 is running! global_variable: 2
[U] pid: 1 is running! global_variable: 3
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 6, priority = 7
pid = 3, counter = 6, priority = 7
pid = 4, counter = 6, priority = 7
switch from 1 to [PID = 2 PRIORITY = 7 COUNTER = 6]
```

我在fork的最后一开始没有给new regs的sepc+4，这样会一直进行fork指令，直到超过nr_tasks数目。

```
[U] pid: 4 is running! global_variable: 2
[U] pid: 4 is running! global_variable: 3
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 0, priority = 7
pid = 3, counter = 0, priority = 7
pid = 4, counter = 0, priority = 7
pid = 5, counter = 6, priority = 7
pid = 6, counter = 6, priority = 7
pid = 7, counter = 6, priority = 7
switch from 4 to [PID = 5 PRIORITY = 7 COUNTER = 6]
[syscall.c,41,do_fork] current: thread sp : ffffffe0002feee8, thread sscratch : 3ffffffd0
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3ffffffd0
[INFO] [PID = 8] forked from [PID = 5]
[syscall.c,87,do_fork] copy page : 80307000 -> ffffffe000325000
[vm.c,124,create_mapping] root : ffffffe000323000, [80325000, 80326000) -> [3ffffff000, 4000000000), perm: 17
[syscall.c,87,do_fork] copy page : 80301000 -> ffffffe000329000
[vm.c,124,create_mapping] root : ffffffe000323000, [80329000, 8032a000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 80304000 -> ffffffe00032c000
[vm.c,124,create_mapping] root : ffffffe000323000, [8032c000, 8032d000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 80305000 -> ffffffe00032d000
[vm.c,124,create_mapping] root : ffffffe000323000, [8032d000, 8032e000) -> [12000, 13000), perm: 1f
[U] pid: 5 is running! global_variable: 1
[U] pid: 5 is running! global_variable: 2
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 0, priority = 7
pid = 3, counter = 0, priority = 7
pid = 4, counter = 0, priority = 7
pid = 5, counter = 0, priority = 7
pid = 6, counter = 6, priority = 7
pid = 7, counter = 6, priority = 7
pid = 8, counter = 6, priority = 7
switch from 5 to [PID = 6 PRIORITY = 7 COUNTER = 6]
[syscall.c,41,do_fork] current: thread sp : ffffffe00030aee8, thread sscratch : 3ffffffd0
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3ffffffd0
[INFO] [PID = 9] forked from [PID = 6]
[syscall.c,56,do_fork] task number exceed
```

四、思考题

1. 呈现出你在 page fault 的时候拷贝 ELF 程序内容的逻辑。

通过vma获取bad_addr所在segment和page的起始地址

```
uint64_t bad_seg_start = vma->vm_pgoff + (uint64_t)_sramdisk;  // 文件段起始位置
```

```
uint64_t bad_seg_now = bad_seg_start + PGROUNDDOWN(bad_addr) - vma->vm_start;  // 当前映射段的起始位置
```

拷贝起始地址:

- 如果bad_seg_now和bad_addr在同一页，从seg_start开始拷贝
- bad_seg_now处于一个整页page，直接拷贝所在页的内容
- 如果bad_seg_now和和file size的末尾在同一个page，只对在file size范围内部分进行拷贝，但建立映射的范围是整个page
- bad_seg_now和vm_end在同一个page，从vm_end所在页的起始部分开始拷贝，size=PGOFFSET(vm_end)

2. 回答4.3.5 中的问题:

- ### 在 do_fork 中，父进程的内核栈和用户栈指针分别是什么?

内核栈指针为thread_struct.sp, 用户栈指针为thread_struct.sscratch

- ### 在 do_fork 中，子进程的内核栈和用户栈指针的值应该是什么?

内核栈指针的值为子进程的pt_regs地址，用户栈指针的值为当前sscratch寄存器的值，即父进程用户态栈指针

- ### 在 do_fork 中，子进程的内核栈和用户栈指针分别应该赋值给谁?

用户态进程的thread_struct.sp和thread_struct.sscratch

3. 为什么要为子进程 `pt_regs` 的 `sepc` 手动加四?

子进程应该和父进程一样回到用户程序的对应位置。父进程在do_fork返回用户程序经过trap_handler时，因为发生的是系统调用异常，需要手动+4，避免重复系统调用语句从而死循环。而子进程返回用户程序不经过trap_handler，为达到相同效果需要在此时为其手动+4，使得_traps通过sret返回时，回到下一条语句。

4. 对于 `Fork main #2` （即 `FORK2`），在运行时，`ZJU OS Lab5` 位于内存的什么位置？是否在读取的时候产生了 page fault？请给出必要的截图以说明。

位于[p_vaddr + p_filesz, p_vaddr + p_memsz)

没有发生page fault，在fork进程时父进程已经映射其所在的内存段，子进程也完成复制以及映射

在[12000, 13000)这段里面

```
switch from 0 to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,82,do_page_fault] [PID = 1 PC = 100e8] valid page fault at `100e8` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d2000, 802d3000) -> [10000, 11000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 101ac] valid page fault at `3ffffffff8` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d5000, 802d6000) -> [3ffffff000, 4000000000), perm: 17
[trap.c,82,do_page_fault] [PID = 1 PC = 101d0] valid page fault at `12000` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d8000, 802d9000) -> [12000, 13000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 112d4] valid page fault at `112d4` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d9000, 802da000) -> [11000, 12000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 1043c] valid page fault at `14008` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802da000, 802da3f8) -> [14000, 143f8), perm: 1f
[U] pid: 1 is running! global_variable: 0
[U] pid: 1 is running! global_variable: 1
[U] pid: 1 is running! global_variable: 2
[trap.c,82,do_page_fault] [PID = 1 PC = 10228] valid page fault at `13008` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802db000, 802dc000) -> [13000, 14000), perm: 1f
[syscall.c,41,do_fork] current: thread sp : ffffffe0002cf000, thread sscratch : 4000000000
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3ffffffc0
[INFO] [PID = 2] forked from [PID = 1]
[syscall.c,87,do_fork] copy page : 802d5000 -> ffffffe0002df000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802df000, 802e0000) -> [3ffffff000, 4000000000), perm: 17
[syscall.c,87,do_fork] copy page : 802d2000 -> ffffffe0002e3000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e3000, 802e4000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d9000 -> ffffffe0002e6000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e6000, 802e7000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802d8000 -> ffffffe0002e7000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e7000, 802e8000) -> [12000, 13000), perm: 1f
[syscall.c,87,do_fork] copy page : 802db000 -> ffffffe0002e8000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e8000, 802e9000) -> [13000, 14000), perm: 1f
[syscall.c,87,do_fork] copy page : 802da000 -> ffffffe0002e9000
[vm.c,125,create_mapping] root : ffffffe0002dd000, [802e9000, 802ea000) -> [14000, 15000), perm: 1f
[U-PARENT] pid: 1 is running! Message: ZJU OS Lab5
[U-PARENT] pid: 1 is running! global_variable: 3
[U-PARENT] pid: 1 is running! global_variable: 4
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 6, priority = 7
switch from 1 to [PID = 2 PRIORITY = 7 COUNTER = 6]
[U-CHILD] pid: 2 is running! Message: ZJU OS Lab5
[U-CHILD] pid: 2 is running! global_variable: 3
```

5. 画图分析 `make run TEST=FORK3` 的进程 fork 过程，并呈现出各个进程的 `global_variable` 应该从几开始输出，再与你的输出进行对比验证

开始的全局变量值如下：

pid 1 - 0

- pid 2 - 1
  - pid 5 - 1
    - pid 8 - 2
  - pid 6 - 2
- pid 3 - 1
  - pid 7 - 2
- pid 4 - 2

和输出的结果相同：

```
[trap.c,82,do_page_fault] [PID = 1 PC = 100e8] valid page fault at `100e8` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d2000, 802d3000) -> [10000, 11000), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 101ac] valid page fault at `3ffffffff8` with cause 15
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d5000, 802d6000) -> [3ffffff000, 4000000000), perm: 17
[trap.c,82,do_page_fault] [PID = 1 PC = 101c8] valid page fault at `12000` with cause 13
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d8000, 802d83f0) -> [12000, 123f0), perm: 1f
[trap.c,82,do_page_fault] [PID = 1 PC = 11134] valid page fault at `11134` with cause 12
[vm.c,125,create_mapping] root : ffffffe0002cf000, [802d9000, 802da000) -> [11000, 12000), perm: 1f
[U] pid: 1 is running! global_variable: 0
[syscall.c,41,do_fork] current: thread sp : ffffffe0002cf000, thread sscratch : 4000000000
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 2] forked from [PID = 1]
```

```
[INFO] [PID = 5] forked from [PID = 2]
[syscall.c,87,do_fork] copy page : 802e1000 -> ffffffe000301000
[vm.c,125,create_mapping] root : ffffffe0002ff000, [80301000, 80302000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e4000 -> ffffffe000304000
[vm.c,125,create_mapping] root : ffffffe0002ff000, [80304000, 80305000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e5000 -> ffffffe000305000
[vm.c,125,create_mapping] root : ffffffe0002ff000, [80305000, 80306000) -> [12000, 13000), perm: 1f
[syscall.c,87,do_fork] copy page : 802dd000 -> ffffffe000307000
[vm.c,125,create_mapping] root : ffffffe0002ff000, [80307000, 80308000) -> [3ffffff000, 4000000000), perm: 17
[U] pid: 2 is running! global_variable: 1
[syscall.c,41,do_fork] current: thread sp : ffffffe0002daee8, thread sscratch : 3fffffffd0
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 6] forked from [PID = 2]
[syscall.c,87,do_fork] copy page : 802e1000 -> ffffffe00030d000
[vm.c,125,create_mapping] root : ffffffe00030b000, [8030d000, 8030e000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e4000 -> ffffffe000310000
[vm.c,125,create_mapping] root : ffffffe00030b000, [80310000, 80311000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e5000 -> ffffffe000311000
[vm.c,125,create_mapping] root : ffffffe00030b000, [80311000, 80312000) -> [12000, 13000), perm: 1f
[syscall.c,87,do_fork] copy page : 802dd000 -> ffffffe000313000
[vm.c,125,create_mapping] root : ffffffe00030b000, [80313000, 80314000) -> [3ffffff000, 4000000000), perm: 17
```

```
[U] pid: 3 is running! global_variable: 1
[syscall.c,41,do_fork] current: thread sp : ffffffe0002e6ee8, thread sscratch : 3fffffffd0
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 7] forked from [PID = 3]
[syscall.c,87,do_fork] copy page : 802ed000 -> ffffffe000319000
[vm.c,125,create_mapping] root : ffffffe000317000, [80319000, 8031a000) -> [10000, 11000), perm: 1f
[syscall.c,87,do_fork] copy page : 802f0000 -> ffffffe00031c000
[vm.c,125,create_mapping] root : ffffffe000317000, [8031c000, 8031d000) -> [11000, 12000), perm: 1f
[syscall.c,87,do_fork] copy page : 802f1000 -> ffffffe00031d000
[vm.c,125,create_mapping] root : ffffffe000317000, [8031d000, 8031e000) -> [12000, 13000), perm: 1f
[syscall.c,87,do_fork] copy page : 802e9000 -> ffffffe00031f000
[vm.c,125,create_mapping] root : ffffffe000317000, [8031f000, 80320000) -> [3ffffff000, 4000000000), perm: 17
```

```
switch from 3 to [PID = 4 PRIORITY = 7 COUNTER = 6]
[U] pid: 4 is running! global_variable: 2
[U] pid: 4 is running! global_variable: 3
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 0, priority = 7
pid = 3, counter = 0, priority = 7
pid = 4, counter = 0, priority = 7
pid = 5, counter = 6, priority = 7
pid = 6, counter = 6, priority = 7
pid = 7, counter = 6, priority = 7
switch from 4 to [PID = 5 PRIORITY = 7 COUNTER = 6]
[U] pid: 5 is running! global_variable: 1
[syscall.c,41,do_fork] current: thread sp : ffffffe0002feee8, thread sscratch : 3fffffffd0
[syscall.c,42,do_fork] sp : ffffffe0002ceee8
[syscall.c,46,do_fork] cur_sscratch : 3fffffffd0
[INFO] [PID = 8] forked from [PID = 5]
```

```
switch from 5 to [PID = 6 PRIORITY = 7 COUNTER = 6]
[U] pid: 6 is running! global_variable: 2
[U] pid: 6 is running! global_variable: 3
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 0, priority = 7
pid = 3, counter = 0, priority = 7
pid = 4, counter = 0, priority = 7
pid = 5, counter = 0, priority = 7
pid = 6, counter = 0, priority = 7
pid = 7, counter = 6, priority = 7
pid = 8, counter = 6, priority = 7
switch from 6 to [PID = 7 PRIORITY = 7 COUNTER = 6]
[U] pid: 7 is running! global_variable: 2
[U] pid: 7 is running! global_variable: 3
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 0, priority = 7
pid = 2, counter = 0, priority = 7
pid = 3, counter = 0, priority = 7
pid = 4, counter = 0, priority = 7
pid = 5, counter = 0, priority = 7
pid = 6, counter = 0, priority = 7
pid = 7, counter = 0, priority = 7
pid = 8, counter = 6, priority = 7
switch from 7 to [PID = 8 PRIORITY = 7 COUNTER = 6]
[U] pid: 8 is running! global_variable: 2
[U] pid: 8 is running! global_variable: 3
SCHEDULE(): INFO OF ALL PROCESSES
pid = 0, counter = 0, priority = 0
pid = 1, counter = 7, priority = 7
```