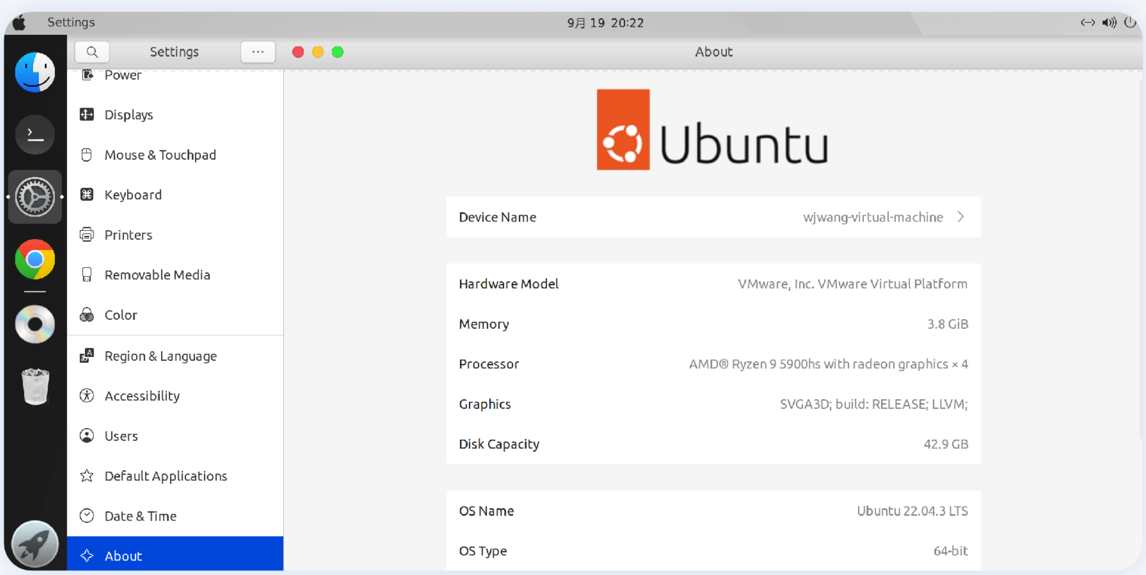
Lab 6: 实现 fork 机制

学号：3210106034
姓名：王伟杰

实验环境：

```
uname -a
Linux wjwang-virtual-machine 6.2.0-33-generic #33~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Thu Sep  7 10:33:52 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
```



调用sys_clone

添加 `sys_clone` 的代码：

```
// syscall.h
#define SYS_CLONE 220
```

在 `trap_handler` 中添加处理 `sys_clone` 的代码：

```c
void trap_handler(unsigned long scause, unsigned long sepc, struct pt_regs *regs) {
    unsigned long temp = 1;
    if(scause & (temp << 63)) { // interrupt
        // ...
    } else {
        switch (scause & ~(temp << 63))
        {
        case 0x8: // ECALL_FROM_U_MODE
            uint64 syscall_id = regs->a7;
            switch (syscall_id)
            {
            // ...
            case SYS_CLONE: // sys_clone
                regs->a0 = sys_clone(regs);
                break;
            }
         // ...
        }
    } // exception
}
```

__ret_from_fork

在 `_traps` 中的 `jal x1, trap_handler` 后面插入一个符号：

```
    .global _traps
_traps:
    ...
   jal x1, trap_handler
    .global __ret_from_fork
__ret_from_fork:
    ... ;利用 sp 从栈中恢复出寄存器的值
    sret
```

实现sys_clone

`sys_clone` 的处理逻辑如下：

- 创建一个新的 task，将的 parent task 的整个页复制到新创建的 `task_struct` 页上，对复制过来的 `task_struct` 重新设置 `pid` ，设置 `thread.ra` 设置为 `__ret_from_fork`
- 根据parent task的 `regs` 位置计算出child task对应的 `regs` 位置，将 `thread.sp` 设置为改地址
- 设置child task的 `pt_regs` ，子进程的 `fork` 返回值 `a0` 设为0， `sepc` 设置为父进程 `sepc + 4` ，使子进程跳过 `fork` 的这一条指令，继续运行下一条指令
- 为child task分配一个根页表，由于之前我们的 `task_init` 中的进程都是共享页表的，所以在这里，我也不创建内核页表的映射，直接使用之前的方式，让它们共享内核页表
- 最后，我遍历父进程的 `vma` ，找到父进程对应 `vma` 中valid的PTE，将这些PTE对应的frame拷贝出一份新的，即child task在用户态会用到的内存

```c
int IsValid(uint64 *pgd, uint64 va) {
    uint64 VPN[3];
    VPN[2] = (va >> 30) & 0x1ff; // 9 bit
    VPN[1] = (va >> 21) & 0x1ff;
    VPN[0] = (va >> 12) & 0x1ff;
    uint64 *pte = pgd;
    for (int j = 2; j > 0; j--) {
        if ((pte[VPN[j]] & 0x1) == 0) {
            return 0;
        }
        else {
            pte = (uint64 *)((pte[VPN[j]] >> 10 << 12) + PA2VA_OFFSET);
        }
    }
    if ((pte[VPN[0]] & 0x1) == 0) {
        return 0;
    }
    return 1;
}

uint64_t sys_clone(struct pt_regs *regs) {
    // 创建一个新的 task
    struct task_struct *child = (struct task_struct *)kalloc();
    memcpy((void*)child,(void*)current,PGSIZE);
    for (int i = 0; i < NR_TASKS; ++i) {
        if (task[i] == NULL) {
            task[i] = child;
            child->pid = i;
            // printk("sys_clone: child->pid = %d\n", child->pid);
            break;
        }
```

```c
    }
    child->thread.ra = (uint64)__ret_from_fork;

    // 利用参数 regs 来计算出 child task 的对应的 pt_regs 的地址
    uint64 offset = (uint64)regs - PGROUNDDOWN((uint64)regs);
    struct pt_regs *child_regs = (struct pt_regs *)((uint64)child + offset);
    child->thread.sp = (uint64)child_regs;
    child_regs->a0 = 0;
    // child_regs->sscratch = regs->sscratch;
    child_regs->sepc = regs->sepc + 4;

    // 为 child task 分配一个根页表
    child->pgd = (pagetable_t)alloc_page();
    memcpy((void *)child->pgd, (void *)swapper_pg_dir, PGSIZE);

    // 根据 parent task 的页表和 vma 来分配并拷贝 child task 在用户态会用到的内存
    for (int i = 0; i < current->vma_cnt; ++i) {
        struct vm_area_struct *vma = &current->vmas[i];
        uint64 now_page = PGROUNDDOWN(vma->vm_start);
        while (now_page < vma->vm_end) {
            if (IsValid(current->pgd, now_page)) {
                // printk("sys_clone: now_page = %lx\n", now_page);
                uint64 page = alloc_page();
                memcpy((void *)page, (void *)(now_page), PGSIZE);
                create_mapping(child->pgd, now_page, page - PA2VA_OFFSET, PGSIZE, (vma->vm_flags & 0b1110) | 0x11); // UXWRV
            }
            now_page += PGSIZE;
        }
    }

    return child->pid;

}
```

编译及测试

第一个main函数：

```
...setup_vm done!
...buddy_init done!
...setup_vm_final done!
...proc_init done!
2022 Hello RISC-V
sstatus = 8000000000006000
sscratch = 0

switch to [PID = 1 COUNTER = 4 PRIORITY = 37]
Page Fault: sepc: 00000000000100e8, scause: 000000000000000c, stval: 00000000000100e8
Page Fault: sepc: 0000000000010158, scause: 000000000000000f, stval: 0000003ffffffff8
[S] New task: 2
Page Fault: sepc: 00000000000101f0, scause: 000000000000000d, stval: 0000000000011978
[U-PARENT] pid: 1 is running!, global_variable: 0
[U-PARENT] pid: 1 is running!, global_variable: 1
[U-PARENT] pid: 1 is running!, global_variable: 2
[INTERRUPT] S mode timer interrupt!
[U-PARENT] pid: 1 is running!, global_variable: 3
[U-PARENT] pid: 1 is running!, global_variable: 4
[INTERRUPT] S mode timer interrupt!
[U-PARENT] pid: 1 is running!, global_variable: 5
[U-PARENT] pid: 1 is running!, global_variable: 6
[INTERRUPT] S mode timer interrupt!
[U-PARENT] pid: 1 is running!, global_variable: 7
[U-PARENT] pid: 1 is running!, global_variable: 8
[INTERRUPT] S mode timer interrupt!

switch to [PID = 2 COUNTER = 4 PRIORITY = 37]
Page Fault: sepc: 000000000001018c, scause: 000000000000000d, stval: 0000000000011978
[U-CHILD] pid: 2 is running!, global_variable: 0
[U-CHILD] pid: 2 is running!, global_variable: 1
[U-CHILD] pid: 2 is running!, global_variable: 2
[INTERRUPT] S mode timer interrupt!
[U-CHILD] pid: 2 is running!, global_variable: 3
[U-CHILD] pid: 2 is running!, global_variable: 4
[INTERRUPT] S mode timer interrupt!
[U-CHILD] pid: 2 is running!, global_variable: 5
[U-CHILD] pid: 2 is running!, global_variable: 6
[INTERRUPT] S mode timer interrupt!
[U-CHILD] pid: 2 is running!, global_variable: 7
[U-CHILD] pid: 2 is running!, global_variable: 8
[INTERRUPT] S mode timer interrupt!
SET [PID = 1 COUNTER = 1]
SET [PID = 2 COUNTER = 4]

switch to [PID = 1 COUNTER = 1 PRIORITY = 37]
[U-PARENT] pid: 1 is running!, global_variable: 9
[U-PARENT] pid: 1 is running!, global_variable: 10
```

第二个main函数：

```
2022 Hello RISC-V
sstatus = 8000000000006000
sscratch = 0

switch to [PID = 1 COUNTER = 4 PRIORITY = 37]
Page Fault: sepc: 00000000000100e8, scause: 000000000000000c, stval: 00000000000100e8
Page Fault: sepc: 0000000000010158, scause: 000000000000000f, stval: 0000003ffffffff8
Page Fault: sepc: 000000000001017c, scause: 000000000000000d, stval: 0000000000011a00
[U] pid: 1 is running!, global_variable: 0
[U] pid: 1 is running!, global_variable: 1
[U] pid: 1 is running!, global_variable: 2
[S] New task: 2
[U-PARENT] pid: 1 is running!, global_variable: 3
[U-PARENT] pid: 1 is running!, global_variable: 4
[U-PARENT] pid: 1 is running!, global_variable: 5
[INTERRUPT] S mode timer interrupt!
[U-PARENT] pid: 1 is running!, global_variable: 6
[U-PARENT] pid: 1 is running!, global_variable: 7
[INTERRUPT] S mode timer interrupt!
[U-PARENT] pid: 1 is running!, global_variable: 8
[U-PARENT] pid: 1 is running!, global_variable: 9
[INTERRUPT] S mode timer interrupt!
[U-PARENT] pid: 1 is running!, global_variable: 10
[U-PARENT] pid: 1 is running!, global_variable: 11
[INTERRUPT] S mode timer interrupt!

switch to [PID = 2 COUNTER = 4 PRIORITY = 37]
[U-CHILD] pid: 2 is running!, global_variable: 3
[U-CHILD] pid: 2 is running!, global_variable: 4
[U-CHILD] pid: 2 is running!, global_variable: 5
[INTERRUPT] S mode timer interrupt!
[U-CHILD] pid: 2 is running!, global_variable: 6
[U-CHILD] pid: 2 is running!, global_variable: 7
[INTERRUPT] S mode timer interrupt!
[U-CHILD] pid: 2 is running!, global_variable: 8
[U-CHILD] pid: 2 is running!, global_variable: 9
[INTERRUPT] S mode timer interrupt!
[U-CHILD] pid: 2 is running!, global_variable: 10
[U-CHILD] pid: 2 is running!, global_variable: 11
[INTERRUPT] S mode timer interrupt!
SET [PID = 1 COUNTER = 1]
SET [PID = 2 COUNTER = 4]

switch to [PID = 1 COUNTER = 1 PRIORITY = 37]
[U-PARENT] pid: 1 is running!, global_variable: 12
[U-PARENT] pid: 1 is running!, global_variable: 13
[INTERRUPT] S mode timer interrupt!
```

第三个main函数：

```
2022 Hello RISC-V
sstatus = 8000000000006000
sscratch = 0

switch to [PID = 1 COUNTER = 4 PRIORITY = 37]
Page Fault: sepc: 00000000000100e8, scause: 000000000000000c, stval: 00000000000100e8
Page Fault: sepc: 0000000000010158, scause: 000000000000000f, stval: 0000003ffffffff8
Page Fault: sepc: 0000000000010174, scause: 000000000000000d, stval: 0000000000011930
[U] pid: 1 is running!, global_variable: 0
[S] New task: 2
[U] pid: 1 is running!, global_variable: 1
[S] New task: 3
[U] pid: 1 is running!, global_variable: 2
[U] pid: 1 is running!, global_variable: 3
[U] pid: 1 is running!, global_variable: 4
[INTERRUPT] S mode timer interrupt!
[U] pid: 1 is running!, global_variable: 5
[U] pid: 1 is running!, global_variable: 6
[INTERRUPT] S mode timer interrupt!
[U] pid: 1 is running!, global_variable: 7
[U] pid: 1 is running!, global_variable: 8
[INTERRUPT] S mode timer interrupt!
[U] pid: 1 is running!, global_variable: 9
[U] pid: 1 is running!, global_variable: 10
[INTERRUPT] S mode timer interrupt!

switch to [PID = 2 COUNTER = 4 PRIORITY = 37]
[U] pid: 2 is running!, global_variable: 1
[S] New task: 4
[U] pid: 2 is running!, global_variable: 2
[U] pid: 2 is running!, global_variable: 3
[U] pid: 2 is running!, global_variable: 4
[INTERRUPT] S mode timer interrupt!
[U] pid: 2 is running!, global_variable: 5
[U] pid: 2 is running!, global_variable: 6
[INTERRUPT] S mode timer interrupt!
[U] pid: 2 is running!, global_variable: 7
[U] pid: 2 is running!, global_variable: 8
[INTERRUPT] S mode timer interrupt!
[U] pid: 2 is running!, global_variable: 9
[U] pid: 2 is running!, global_variable: 10
[INTERRUPT] S mode timer interrupt!

switch to [PID = 3 COUNTER = 4 PRIORITY = 37]
[U] pid: 3 is running!, global_variable: 2
[U] pid: 3 is running!, global_variable: 3
[U] pid: 3 is running!, global_variable: 4
```

更多测试样例

将INIT_TASKS改为3（即初始化两个进程），测试结果如下：

```
switch to [PID = 2 COUNTER = 4 PRIORITY = 88]
[U-PARENT] pid: 2 is running! the 39th fibonacci number is 63245986 and the number @ 961 in the large array is 961

switch to [PID = 4 COUNTER = 4 PRIORITY = 88]
[U-CHILD] pid: 4 is running! the 39th fibonacci number is 63245986 and the number @ 961 in the large array is 961

switch to [PID = 3 COUNTER = 10 PRIORITY = 37]
[U-CHILD] pid: 3 is running! the 37th fibonacci number is 24157817 and the number @ 963 in the large array is 963
[U-CHILD] pid: 3 is running! the 38th fibonacci number is 39088169 and the number @ 962 in the large array is 962
[U-CHILD] pid: 3 is running! the 39th fibonacci number is 63245986 and the number @ 961 in the large array is 961
[U-CHILD] pid: 3 is running! the 40th fibonacci number is 102334155 and the number @ 960 in the large array is 960
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 PRIORITY = 88]
[U-CHILD] pid: 4 is running! the 40th fibonacci number is 102334155 and the number @ 960 in the large array is 960

switch to [PID = 3 COUNTER = 5 PRIORITY = 37]

switch to [PID = 1 COUNTER = 10 PRIORITY = 37]
[U-PARENT] pid: 1 is running! the 37th fibonacci number is 24157817 and the number @ 963 in the large array is 963
[U-PARENT] pid: 1 is running! the 38th fibonacci number is 39088169 and the number @ 962 in the large array is 962
[U-PARENT] pid: 1 is running! the 39th fibonacci number is 63245986 and the number @ 961 in the large array is 961
[U-PARENT] pid: 1 is running! the 40th fibonacci number is 102334155 and the number @ 960 in the large array is 960

switch to [PID = 2 COUNTER = 10 PRIORITY = 88]
[U-PARENT] pid: 2 is running! the 40th fibonacci number is 102334155 and the number @ 960 in the large array is 960
```

思考题

1. *参考 task_init 创建一个新的 task，将的 parent task 的整个页复制到新创建的 task_struct 页上。*这一步复制了哪些东西?

   在这一步中，新创建的 `task_struct` 页复制了 parent task 的整个页。这包括了 parent task 的所有内存状态，如 `state` / `counter` / `priority` / `pid` / `thread` / `vma_cnt` / `vmas` 等等，以及parent task内核栈的所有信息（如 `pt_regs` ），虽然之后我们会重新设置 `pid` 和 `thread` 中的内容，但是大部分状态是不变的。

2. *将 thread.ra 设置为 `__ret_from_fork`，并正确设置 `thread.sp`。*仔细想想，这个应该设置成什么值？可以根据 child task 的返回路径来倒推。

   `thread.sp` 应该设置为child task的 `pt_regs` 的地址，目前栈中有一个 `pt_regs` ， `thread.sp` 指向 child task 的栈顶部，即现在的 `child_regs` 地址。

3. *利用参数 regs 来计算出 child task 的对应的 pt_regs 的地址，并将其中的 a0, sp, sepc 设置成正确的值。*为什么还要设置 sp?

   我之前的 `pt_regs` 中并没有保存 `sp` ，所以不需要设置。若保存了 `sp` ，则需要设置为当前内核栈，即 `child_regs` 地址