Operating system

Lab 5

Author: 吴杭 李克成

Date: 2024/12/24

Autumn-Fall 2024-2025 Semester

Table of Contents

Chapter 1: 实验流程

1.1) 4.3 实现 fork 的系统调用

首先根据实验文档中的内容搭好框架，这里不再展开。

1.1.1) 拷贝内核栈

直接根据 current（当前在运行的线程的 task struct）来深拷贝这一页的内容，就能把内核栈拷贝到新的进程中。其他再根据实验文档来修改 task_struct 相关的内容。

```
uint64_t do_fork(struct pt_regs *regs) {
    // copy kernel stack
    struct task_struct *_task = (struct task_struct *)kalloc();
    memcpy(_task, current, PGSIZE);
    // assign new pid
    _task->pid = nr_tasks++;
    // assign new page directory
    _task->pgd = alloc_page();
    // set mm.mmap
    _task->mm.mmap = NULL;
    ...
}
```

1.1.2) 创建子进程页表

内核页表每个线程都是一致的，所以直接拷贝就行。

```
    // copy the swapper_pg_dir into the new task's pgd
    memcpy(_task->pgd, swapper_pg_dir, PGSIZE);
```

接下来需要遍历父进程的所有 vma，子进程作为父进程的副本，此时的运行状态理论上要和父进程一致，所以需要把父进程的所有 vma 都复制一份，添加到子进程的 vma 链表中。同时，父进程的 vma 对应的页表如果存在，说明此时这一页已经被创建，所以需要把这一页的内容拷贝到子进程的页表中。

下面是拷贝 vma 的代码，主要的逻辑就是复制一份 vma 的结构，然后执行链表的 insert 操作即可。

```
    // iterate the vma list of the current task
    struct vm_area_struct *vma = current->mm.mmap;
    while (vma != NULL) {
        // create a new vma
        struct vm_area_struct *new_vma = (struct vm_area_struct *)kalloc();
        memcpy(new_vma, vma, sizeof(struct vm_area_struct));
        // link the new vma to the new task's mm
        new_vma->vm_mm = &(_task->mm);
        new_vma->vm_next = _task->mm.mmap;
        new_vma->vm_prev = NULL;
        // link the new vma to the new task's mm
        if (_task->mm.mmap == NULL) {
            _task->mm.mmap = new_vma;
        } else {
```

```
            // struct vm_area_struct *tmp = _task->mm.mmap;
            _task->mm.mmap->vm_prev = new_vma;
            _task->mm.mmap = new_vma;
        }
        // if the corresponding page table entry exists, copy the content
        check_and_copy_pages(current->pgd, new_vma->vm_start, new_vma->vm_end, _task-
>pgd, new_vma->vm_flags);
        // move to the next vma
        vma = vma->vm_next;
    }
```

下面是拷贝 vma 对应的页表项的代码，主要的逻辑是通过传入的 va_start 和 va_end 来遍历经过的每一页，然后通过页表项的 vpn 来找到对应的页表项，然后判断对应页表项的 PRIV_V，如果存在，就给子进程创建一个新的页，再通过 vma 记录的页权限来 creat_mapping，并且把父进程的页深拷贝一份到子进程的页中。这样就保证了父进程和子进程现在用到的资源是完全一致的。

```
// utils: check and copy the content of the page table entry
void check_and_copy_pages(uint64_t *pgd, uint64_t va_start, uint64_t va_end, uint64_t
*new_pgd, uint64_t vm_flags) {
    // notice va is page aligned
    for (uint64_t va = PGROUNDDOWN(va_start); va < va_end; va += PGSIZE) {
        // get the page table entry
        uint64_t vpn0 = (va >> 12) & 0x1ff;
        uint64_t vpn1 = (va >> 21) & 0x1ff;
        uint64_t vpn2 = (va >> 30) & 0x1ff;

        // check if the first page table entry is valid
        if (pgd[vpn2] & PRIV_V) {
            uint64_t *pgtbl1 = get_pgtable(pgd, vpn2);
            // check if the second page table entry is valid
            if (pgtbl1[vpn1] & PRIV_V) {
                uint64_t *pgtbl0 = get_pgtable(pgtbl1, vpn1);
                // check if the third page table entry is valid
                if (pgtbl0[vpn0] & PRIV_V) {
                    // if yes, deep copy the content of the page
                     // notice only the machine mode has the privilege to access the
physical address, so we use memcpy for the virtual page address here
                    char* child_process_page = alloc_page();
                    uint64_t priv_r = (vm_flags & VM_READ) ? PRIV_R : 0;
                    uint64_t priv_w = (vm_flags & VM_WRITE) ? PRIV_W : 0;
                    uint64_t priv_x = (vm_flags & VM_EXEC) ? PRIV_X : 0;
                     create_mapping(new_pgd, va, VA2PA((uint64_t)child_process_page),
PGSIZE, PRIV_U | priv_r | priv_w | priv_x | PRIV_V);
                    // copy the content of current page to the child page
                    memcpy(child_process_page, (char *)va, PGSIZE);
                }
            }
        }
    }
}
```

1.1.3) 处理进程返回逻辑

对于父进程的返回，文档里面已经写得很清楚了，在运行完 do_fork 之后就正常返回子进程的 pid 即可，我们无需做任何的修改。

而子进程可以认为是下一次被调度的时候才会返回，并且调度的过程中 PC 实际上是在 `__switch_to` 的位置，在 `__switch_to` 中，会把子进程的 `task_struct` 给 load 出来，然后根据其中设置的 ra 来跳转到下一个执行的地址。为了让子进程和父进程在 fork 时候的运行状态一致，我们应该让子进程像父进程那样返回到 trap_handler 返回的位置，也就是 `_traps` 中 `call trap_handler` 之后的位置加一个标记 `__ret_from_fork`，然后把子进程的 thread.ra 设置为这个位置即可。

接下来我们处理子进程的一系列 `sp`，首先是 `_task->thread.sp`，由于我们在 `__switch_to` 返回到 `__ret_from_fork` 的时候，马上需要根据此时的 `sp` 来 load 子进程存在内核栈中的值，而内核栈的位置可以通过 `do_fork` 中 `regs` 和 `current` 的相对位置以及 `_task` 的值来得到。

接下来是 `_task->thread.sscratch` 的值，理论上这个寄存器在此时应当存储子进程用户态栈的地址。而这个地址实际上保存在内核栈 `regs` 中存好的 `sscratch` 中。

另外我们还需要思考子进程的内核栈 `child_pt_regs` 中的 `sp` 的值。这个值根据我们前面的分析，实际上会在 `__ret_from_traps` 被 load 到 `sp` 中，观察 `_traps` 中的代码，就会发现他应当和 `_task->thread.sp` 的值相同，也就是和 `child_pt_regs` 的值相同。

最后我们设置子进程的返回值，也就是 `child_pt_regs` 中存好的 `a0`，将其设置为 0。以及设置子进程的 `sepc`，也存在 `sepc` 中，将其的值加 4 即可。

同时由于我们在 `task_struct` 中储存了 `satp`，所以也要作相应的更新。

```
    // struct pt_regs* test_regs = (struct pt_regs *)((uint64_t)current + PGSIZE -
sizeof(struct pt_regs));
   struct pt_regs* child_pt_regs = (struct pt_regs *)((uint64_t)_task + (uint64_t)regs-
(uint64_t)current);
    child_pt_regs->general_regs[1] = (uint64_t)child_pt_regs; // set child_pt_regs->sp
    child_pt_regs->sepc += 4; // set the sepc of the child process
    _task->thread.sscratch = regs->sscratch; // the sscratch of the child process
    _task->thread.sp = (uint64_t)child_pt_regs; // the sp of the child process
    _task->thread.ra = (uint64_t)__ret_from_fork; // the ra of the child process
    // set the return value of the child process
    child_pt_regs->general_regs[9] = 0;
    // set the satp of the chlid process
    uint64_t ppn = VA2PA((uint64_t)(_task->pgd)) >> 12;
    _task->thread.satp = ppn | (SATP_MODE_SV39 << 60);
    // add _task to the task list
    task[_task->pid] = _task;
    // return the new task's pid
    return _task->pid;
```

1.2) COW 实现

首先把原来子进程复制父进程页表中已经分配的页面的逻辑改成给引用的 page 的计数+1，但是不进行 memcpy 的操作，这里要注意的是，传给 get_page 的地址是内核态的虚拟地址，所以需要把用户态的虚拟地址通过页表查询出对应的物理页，然后得到再通过 PA2VA 转化为内核态的虚拟地址，然后传给 get_page

```
// check and copy pages
if (pgd[vpn2] & PRIV_V) {
  uint64_t *pgtbl1 = get_pgtable(pgd, vpn2);
  // check if the second page table entry is valid
  if (pgtbl1[vpn1] & PRIV_V) {
    uint64_t *pgtbl0 = get_pgtable(pgtbl1, vpn1);
    // check if the third page table entry is valid
    if (pgtbl0[vpn0] & PRIV_V) {
      // icnrease the page count of the referenced page
      uint64_t test = (pgtbl0[vpn0] & ~((1 << 10) - 1)) << 2;
      get_page((uint64_t *)PA2VA(test));
      // set the PTE_W of the parent process as 0
      pgtbl0[vpn0] &= ~PRIV_W;
      uint64_t priv_r = (vm_flags & VM_READ) ? PRIV_R : 0;
      uint64_t priv_x = (vm_flags & VM_EXEC) ? PRIV_X : 0;
      create_mapping(new_pgd, va,
                     (uint64_t)((pgtbl0[vpn0] & ~((1 << 10) - 1)) << 2),
                     PGSIZE, PRIV_U | priv_r | priv_x | PRIV_V);
```

然后在 trap.c 的 do_page_fault 中再增加一段逻辑处理 COW，具体逻辑是 alloc 一个新的 page，然后把原来 pgtbl 中记录的 page 的计数减一，并且需要先把 pgtbl 原来的页表项先给置为 Invalid（否则在 create_mapping 中不会创建新的映射），然后 create_mapping 就可以了。

```
if (priv_w) {
  // get the pgtbl entry
  uint64_t vpn0 = (bad_addr >> 12) & 0x1ff;
  uint64_t vpn1 = (bad_addr >> 21) & 0x1ff;
  uint64_t vpn2 = (bad_addr >> 30) & 0x1ff;
  uint64_t *pgtbl1 = get_pgtable(current->pgd, vpn2);
  uint64_t *pgtbl0 = get_pgtable(pgtbl1, vpn1);
  // check if the entry is valid but not writable. if yes, do the cow
  if ((pgtbl0[vpn0] & PRIV_V) && !(pgtbl0[vpn0] & PRIV_W)) {
    // create a new page
    uint64_t *page = alloc_page();
    // copy the content of the page
    uint64_t pa = (pgtbl0[vpn0] & ~((1 << 10) - 1)) << 2;
    memcpy(page, (void *)PA2VA(pa), PGSIZE);
    // decrease the page count
    put_page((void *)PA2VA(pa));
    // before create mapping, set the old page table entry's valid bit as 0
    pgtbl0[vpn0] &= ~PRIV_V;
    // create a new mapping
    create_mapping(current->pgd, (uint64_t)PGROUNDDOWN(bad_addr),
                   VA2PA((uint64_t)page), PGSIZE,
```

```
                    PRIV_U | priv_r | priv_w | priv_x | PRIV_V);
      Log("cow page fault at 0x%lx, create a new page at 0x%lx", bad_addr,
          VA2PA((uint64_t)page));
      return;
    }
  }
```

1.3) 运行结果

运行 `make run TEST=FORK1`

```
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 12, sepc = 0x100e8
[trap.c,22,do_page_fault] page fault at 0x100e8
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x802d1000
[trap.c,86,trap_handler] trap: scause = 15, sepc = 0x101ac
[trap.c,22,do_page_fault] page fault at 0x3fffffff8
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x802de000
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 13, sepc = 0x10240
[trap.c,22,do_page_fault] page fault at 0x12000
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x802e1000
[trap.c,86,trap_handler] trap: scause = 12, sepc = 0x1114c
[trap.c,22,do_page_fault] page fault at 0x1114c
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x802e2000
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x1121c
[U-PARENT] pid: 1 is running! global_variable: 0
switch to [PID = 2 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 13, sepc = 0x101e8
[trap.c,22,do_page_fault] page fault at 0x12000
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x802e3000
[trap.c,86,trap_handler] trap: scause = 12, sepc = 0x1114c
[trap.c,22,do_page_fault] page fault at 0x1114c
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x802e4000
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x1121c
[U-CHILD] pid: 2 is running! global_variable: 0
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x1121c
[U-PARENT] pid: 1 is running! global_variable: 1
switch to [PID = 2 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x1121c
[U-CHILD] pid: 2 is running! global_variable: 1
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
switch to [PID = 2 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x1121c
[U-CHILD] pid: 2 is running! global_variable: 2
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
```

Figure 1: “fork1”

运行 `make run TEST=FORK2`

```
2024 ZJU Operating System
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 12, sepc = 0x100e8
[trap.c,22,do_page_fault] page fault at 0x100e8
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x802d2000, 0x802d3000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 15, sepc = 0x101ac
[trap.c,22,do_page_fault] page fault at 0x3ffffffff8
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x802d5000, 0x802d6000), perm = 0x17
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 13, sepc = 0x101d4
[trap.c,22,do_page_fault] page fault at 0x12000
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x802d8000, 0x802d9000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 12, sepc = 0x11320
[trap.c,22,do_page_fault] page fault at 0x11320
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x802d9000, 0x802da000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 13, sepc = 0x10470
[trap.c,22,do_page_fault] page fault at 0x14008
[vm.c,91,create_mapping] mapping [0x14000, 0x15000) to [0x802da000, 0x802db000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U] pid: 1 is running! global_variable: 0
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U] pid: 1 is running! global_variable: 1
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U] pid: 1 is running! global_variable: 2
[trap.c,86,trap_handler] trap: scause = 15, sepc = 0x10230
[trap.c,22,do_page_fault] page fault at 0x13008
[vm.c,91,create_mapping] mapping [0x13000, 0x14000) to [0x802db000, 0x802dc000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x802df000, 0x802e0000), perm = 0x17
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x802e3000, 0x802e4000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x802e6000, 0x802e7000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x802e7000, 0x802e8000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x13000, 0x14000) to [0x802e8000, 0x802e9000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x14000, 0x15000) to [0x802e9000, 0x802ea000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-PARENT] pid: 1 is running! Message: ZJU OS Lab5
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-PARENT] pid: 1 is running! global_variable: 3
switch to [PID = 2 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-CHILD] pid: 2 is running! Message: ZJU OS Lab5
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-CHILD] pid: 2 is running! global_variable: 3
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
```

Figure 2: “fork2”

```
[vm.c,91,create_mapping] mapping [0x14000, 0x15000) to [0x802e9000, 0x802ea000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-PARENT] pid: 1 is running! Message: ZJU OS Lab5
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-PARENT] pid: 1 is running! global_variable: 3
switch to [PID = 2 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-CHILD] pid: 2 is running! Message: ZJU OS Lab5
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-CHILD] pid: 2 is running! global_variable: 3
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-PARENT] pid: 1 is running! global_variable: 4
switch to [PID = 2 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-CHILD] pid: 2 is running! global_variable: 4
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-PARENT] pid: 1 is running! global_variable: 5
switch to [PID = 2 PRIORITY = 7 COUNTER = 7]
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
switch to [PID = 2 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-CHILD] pid: 2 is running! global_variable: 5
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-PARENT] pid: 1 is running! global_variable: 6
switch to [PID = 2 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x113f0
[U-CHILD] pid: 2 is running! global_variable: 6
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
```

Figure 3: “fork2”

运行 `make run TEST=FORK3`

```
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 12, sepc = 0x100e8
[trap.c,22,do_page_fault] page fault at 0x100e8
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x802d1000, 0x802d2000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 15, sepc = 0x101ac
[trap.c,22,do_page_fault] page fault at 0x3ffffffff8
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x802d4000, 0x802d5000), perm = 0x17
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 13, sepc = 0x101cc
[trap.c,22,do_page_fault] page fault at 0x12000
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x802d7000, 0x802d8000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 12, sepc = 0x11174
[trap.c,22,do_page_fault] page fault at 0x11174
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x802d8000, 0x802d9000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 1 is running! global_variable: 0
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x802dc000, 0x802dd000), perm = 0x17
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x802e0000, 0x802e1000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x802e3000, 0x802e4000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x802e4000, 0x802e5000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x802e8000, 0x802e9000), perm = 0x17
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x802ec000, 0x802ed000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x802ef000, 0x802f0000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x802f0000, 0x802f1000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 1 is running! global_variable: 1
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x802f4000, 0x802f5000), perm = 0x17
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x802f8000, 0x802f9000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x802fb000, 0x802fc000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x802fc000, 0x802fd000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 1 is running! global_variable: 2
switch to [PID = 2 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x80300000, 0x80301000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x80303000, 0x80304000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x80304000, 0x80305000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x80306000, 0x80307000), perm = 0x17
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 2 is running! global_variable: 1
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
```

Figure 4: “fork3”

```
switch to [PID = 2 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x80300000, 0x80301000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x80303000, 0x80304000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x80304000, 0x80305000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x80306000, 0x80307000), perm = 0x17
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 2 is running! global_variable: 1
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x8030c000, 0x8030d000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x8030f000, 0x80310000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x80310000, 0x80311000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x80312000, 0x80313000), perm = 0x17
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 2 is running! global_variable: 2
switch to [PID = 3 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 3 is running! global_variable: 1
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x80318000, 0x80319000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x8031b000, 0x8031c000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x8031c000, 0x8031d000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x8031e000, 0x8031f000), perm = 0x17
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 3 is running! global_variable: 2
switch to [PID = 4 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 4 is running! global_variable: 2
switch to [PID = 5 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 5 is running! global_variable: 1
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10134
[trap.c,97,trap_handler] clone syscall
[vm.c,91,create_mapping] mapping [0x3ffffff000, 0x4000000000) to [0x80324000, 0x80325000), perm = 0x17
[vm.c,91,create_mapping] mapping [0x10000, 0x11000) to [0x80328000, 0x80329000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x11000, 0x12000) to [0x8032b000, 0x8032c000), perm = 0x1f
[vm.c,91,create_mapping] mapping [0x12000, 0x13000) to [0x8032c000, 0x8032d000), perm = 0x1f
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 5 is running! global_variable: 2
switch to [PID = 6 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
```

Figure 5: “fork3”

```
[U] pid: 5 is running! global_variable: 2
switch to [PID = 6 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 6 is running! global_variable: 2
switch to [PID = 7 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 7 is running! global_variable: 2
switch to [PID = 8 PRIORITY = 7 COUNTER = 6]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 8 is running! global_variable: 2
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 7 COUNTER = 7]
SET [PID = 3 PRIORITY = 7 COUNTER = 7]
SET [PID = 4 PRIORITY = 7 COUNTER = 7]
SET [PID = 5 PRIORITY = 7 COUNTER = 7]
SET [PID = 6 PRIORITY = 7 COUNTER = 7]
SET [PID = 7 PRIORITY = 7 COUNTER = 7]
SET [PID = 8 PRIORITY = 7 COUNTER = 7]
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 1 is running! global_variable: 3
switch to [PID = 2 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 2 is running! global_variable: 3
switch to [PID = 3 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 3 is running! global_variable: 3
switch to [PID = 4 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 4 is running! global_variable: 3
switch to [PID = 5 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 5 is running! global_variable: 3
switch to [PID = 6 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 6 is running! global_variable: 3
switch to [PID = 7 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
[U] pid: 7 is running! global_variable: 3
switch to [PID = 8 PRIORITY = 7 COUNTER = 7]
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x10100
[trap.c,86,trap_handler] trap: scause = 8, sepc = 0x11244
```

Figure 6: “fork3”

Chapter 2: 思考题

Chpater 3: 心得体会

3.1) 遇到的问题

1. 在考虑子进程返回的 sp 设置的时候，一开始认为`_task->thread.sp`load 到 `sp` 中的值会在子进程进入到`__ret_from_fork` 后马上被内核栈中 load 出来的值给覆盖掉，所以认

为_task->thread.sp 不需要设置，后来才意识到在__ret_from_fork 中需要是通过_task->thread.spload 出来的 sp 的值来找到内核栈的位置，所以这个值是需要设置的。

3.2) 心得体会

这次实验让我对 fork 机制有了更加深刻的理解，尤其是对于 fork 之后，子进程和父进程返回值不同是怎么实现的，这一点之前困扰了我很久，在做了实验之后就很清晰了。这次 do_fork 中的程序虽然复杂，但是文档写的比较详细，按照上面的来也不会漏掉需要处理的结构。并且需要思考的部分也给了很清楚的提示，体验感很好。

Declaration

We hereby declare that all the work done in this lab 5 is of our independent effort.