浙江大学

本科实验报告

课程名称：	操作系统
实验名称：	实现 fork 机制
姓　　名：	蔡雨谦
学　　院：	计算机学院
系：	计算机系
专　　业：	计算机科学与技术
学　　号：	3210102466
指导教师：	寿黎但

2023 年　　12 月　　25 日

浙江大学实验报告

一、 实验目的

- 为 task 加入 fork 机制，能够支持通过 fork 创建新的用户态 task 。

二、 实验环境

- Ubuntu 22.04.2 LTS Windows Subsystem for Linux 2

三、 操作方法与实验步骤

1. __ret_from_fork

在 _traps 中的 jal x1, trap_handler 后面插入以下内容，使 fork 的程序返回我们想要的位置。

```
    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
```

2. task_init

只初始化 IDLE 进程和一个主进程，剩余进程赋为 NULL。

```
for (int i = 2; i < NR_TASKS; i++)
{
    task[i] = NULL;
}
```

3. sys_clone

寻找一个空闲的进程。

```
int pid = 2;
//找到一个空闲进程
while (task[pid] != NULL)
{
    pid++;
    if (pid >= NR_TASKS)
    {
        return -1;
    }
}
```

复制父进程的页面内容。

```
task[pid] = (struct task_struct *)kalloc();
char *child = (char *)task[pid];
char *parent = (char *)(current);
for (int i = 0; i < PGSIZE; i++)
{
    child[i] = parent[i];
}
```

设置进程的结构体信息。

```
task[pid]->state = TASK_RUNNING;
task[pid]->counter = 0;
task[pid]->priority = rand();
task[pid]->pid = pid;
task[pid]->thread.ra = (uint64)(__ret_from_fork);
task[pid]->thread.sp = (uint64)task[pid] + PGSIZE;
task[pid]->thread.sepc = regs->sepc + 4;
task[pid]->thread.sscratch = regs->sscratch;
task[pid]->thread.sstatus = regs->sstatus;
```

设置子进程的特殊寄存器。由于子进程的返回值为 0，a0 设置为 0。在处理中断时，用户栈和内核栈进行过交换，这里将子进程的 sp 设置为 sscratch。

```
child_regs->a0 = 0;//子进程的返回值为0
uint64 sscratch = csr_read(sscratch);
child_regs->sp = sscratch;//用户栈在trap里和内核栈交换了
child_regs->sepc = regs->sepc + 4;
//复制内核页表
```

复制内核页表。

```
task[pid]->pgd = (unsigned long *)kalloc();
for (int i = 0; i < 512; i++)
{
    task[pid]->pgd[i] = swapper_pg_dir[i];
}
```

复制父进程的用户栈。

```
unsigned long user_stack = kalloc();
create_mapping(task[pid]->pgd, USER_START, user_stack - PA2VA_OFFSET, PGSIZE, 0x1f);
char *child_stack = (char *)(user_stack);
char *parent_stack = (char *)(((regs->sscratch) >> 12 << 12));
for (int i = 0; i < PGSIZE; i++)
{
    child_stack[i] = parent_stack[i];
}
//设置子进程的VMA
```

为子进程设置 VMA。

```
for (int i = 0; i < current->vma_cnt; i++)
{
    struct vm_area_struct *vma = &(current->vmas[i]);
    do_mmap(task[i]->vmas, vma->vm_start, vma->vm_end - vma->vm_start, vma->vm_flags,
            vma->vm_content_offset_in_file, vma->vm_content_size_in_file);
}
```

4. syscall

Syscall 添加 else 分支即可。

```
else if(regs->a7==SYS_CLONE)
{
    regs->a0=sys_clone(regs);
    regs->sepc+=4;
}
else
```

四、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?

这一步复制了父进程的内核态的各种数据，包括内核栈，寄存器，thread_struct 的信息等。

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。

task[pid] + PGSIZE。

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?

因为用户栈和内核栈在 trap 中进行过交换，sscratch 寄存器中保存的才是用户栈。

五、讨论心得

后面几次实验都做得非常艰难，要达到的目的倒是清楚，但具体到地址的各种转化和寄存器的值的变动就容易弄不清方向。而且前期的实验没有做好深入的理解，到后面许多问题堆积起来就很难解决。我的 lab4 就没有理解透彻没能跑通测试，加入 page fault 和 fork 机制后变得更加混乱，只能勉强按照实验指导写出代码，正确性也无从验证，过程相当艰难。